HP Universal CMDB

for the Windows and Solaris operating systems

Software Version: 8.00

Integrations

Document Release Date: January 2009
Software Release Date: January 2009

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.

Third-Party Web Sites

HP provides links to external third-party Web sites to help you find supplemental information. Site content and availability may change without notice. HP makes no representations or warranties whatsoever as to site content or availability.

Copyright Notices

Trademark Notices

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

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

Oracle® is a registered US trademark of Oracle Corporation, Redwood City, California.

Unix® is a registered trademark of The Open Group.

Documentation Updates

This guide's title page 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

You can 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 Support Online 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 HP Software 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 find more information about access levels, go to:

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

To register for an HP Passport ID, go to:

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

Table of Contents

Welcome to This Guide

This guide describes the various integrations available for HP Universal CMDB.

This chapter includes:

How This Guide Is Organized

The guide contains the following parts:

Part I The HP Universal CMDB APIs

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

Part II Federation and Reconciliation

Explains how to define adapters to include data in the CMDB from other sources, in such a way that the source of the data retains control of the data.

Part III Integrations

Describes how to embed applets into external applications using direct links.

Who Should Read This Guide

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

- HP Universal CMDB administrators
- HP Universal CMDB end users
- HP Universal CMDB integration developers

Readers of this guide should be knowledgeable about navigating and using enterprise applications, and be familiar with HP Universal CMDB and enterprise monitoring and management concepts.

Getting More Information

For a complete list of all online documentation included with HP Universal CMDB, additional online resources, information on acquiring documentation updates, and typographical conventions used in this guide, see the the *HP Universal CMDB Deployment Guide* PDF.

Part I

The HP Universal CMDB APIs

1

Introduction to APIs

This chapter lists the APIs that are included with HP Universal CMDB.

This chapter includes:

Concepts

APIs Overview

The following APIs are included with HP Universal CMDB:

- **UCMDB 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 "The HP Universal CMDB Web Service API" on page 15.
- **UCMDB Java API.** Explains how third-party or custom tools can use the Java API to extract data and calculations and to write data to the UCMDB (Universal Configuration Management database). For details, see "The HP Universal CMDB Java API" on page 91.
- **DDM Web Service.** Explains how third-party or custom tools can use the HP Discovery and Dependency Mapping Web Service to manage Discovery and Dependency Mapping (DDM). For details, see "The HP Discovery and Dependency Mapping Web Service API" in *Discovery and Dependency Mapping Guide*.

2

The HP Universal CMDB Web Service API

This chapter explains how third-party or custom tools can use the HP Universal CMDB Web Service API to extract data and calculations and to write data to the UCMDB (Universal Configuration Management database).

Use this chapter in conjunction with the UCMDB schema documentation, available in the online Documentation Library.

This chapter includes:

Concepts

Tasks

Reference

➤ Use Cases on page 52

➤ Examples on page 54

➤ UCMDB General Parameters on page 85

➤ UCMDB Output Parameters on page 89

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

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 *Model Management*.

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

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 HP Business Availability Center:

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

  For details, see "Security Manager Window" in *Model Management*.

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

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:

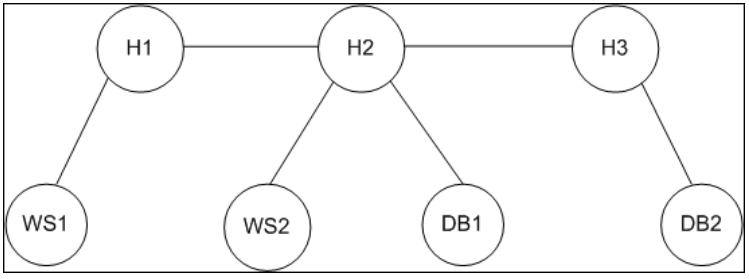
\\<HP Universal CMDB root directory>\UCMDBServer\j2f\AppServer\webapps\site.war\amdocs\eng\doc_lib\Integrations\CMDB_Schema\webframe.html

Returning Unambiguous Topology Map Elements

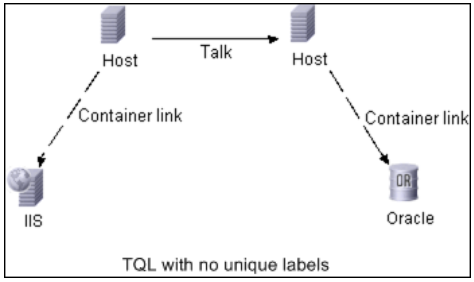
Query methods that return the data in topology or topologyMap elements search the system for a match of a TQL query. The following diagrams illustrate how the resulting topology and topologyMap structures are affected by the use of unique labels in the query.

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
```

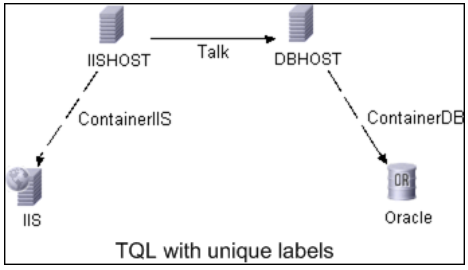
TopologyMap Response

```
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.

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 UCMDB populates one or more of the error message, error code, and exception message fields. If there is no error, the results of the invocation are returned.

SOAP programmers can access the WSDL at:

http://<server>[:port]/axis2/services/UcmdbService?wsdl

The port specification is only necessary for non-standard installations. Consult your system administrator for the correct port number.

The URL for calling the service is:

http://<server>[:port]/axis2/services/UcmdbService

For examples of connecting to the UCMDB, see “Use Cases” on page 52.

Query the UCMDB

The UCMDB is queried using the APIs described in “UCMDB Query Methods” on page 31.

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

For examples of the use of the query methods, see "Query Example" on page 58.

This section includes the following topics:

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 45 and "releaseChunks" on page 46.

Specifying Properties to Return

CIs and relations generally have many properties. Some methods that return collections or graphs of these items accept input parameters that specify which property values to return with each item that matches the query. The 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 UCMD with the update APIs. For details of the API methods, see "UCMDB Update Methods" on page 47.

For examples of the use of the update methods, see "Update Example" on page 75.

This section includes the following topics:

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 27.

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 47 and "UCMDB Query Methods" on page 31.

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 *Model Management*.

For examples of the use of the class model methods, see "Class Model Example" on page 79.

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

- "getClassAncestors" on page 29
- "getAllClassesHierarchy" on page 30
- "getCmdbClassDefinition" on page 30

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 85.
className	The type name. For details, see "Type Name" on page 87.

Output

Parameter	Comment
classHierarchy	A collection of pairs of class names and parent class name. .
comments	For internal use only.

getAllClassesHierarchy

The getAllClassesHierarchy method retrieves the entire class model tree.

Input

Parameter	Comment
cmdbContext	For details, see “CmdbContext” on page 85.

Output

Parameter	Comment
classesHierarchy	A collection of pairs of class name and parent class name.
comments	For internal use only.

getCmdbClassDefinition

The getCmdbClassDefinition method retrieves information about the specified class.

If you use getCmdbClassDefinition to retrieve the key attributes, you must also query the parent classes up to the base class. getCmdbClassDefinition identifies as key attributes only those attributes with the ID_ATTRIBUTE set in the class definition specified by className. Inherited key attributes are not recognized as key attributes of the specified class. Therefore, the complete list of key attributes for the specified class is the union of all the keys of the class and of all its parents, up to the root.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 85.
className	The type name. For details, see "Type Name" on page 87.

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 81.

UCMDB Query Methods

This section provides information on the following methods:

- "executeTopologyQueryByName" on page 32
- "executeTopologyQueryByNameWithParameters" on page 33
- "executeTopologyQueryWithParameters" on page 34
- "getChangedCIs" on page 34

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 18.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 85.
queryName	The name of the TQL in the UCMDB with which to retrieve the map.
queryTypedProperties	A collection of sets of properties to retrieve to items of a specific Configuration Item Type.

Output

Parameter	Comment
topologyMap	For details, see “TopologyMap” on page 90.

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
cmdbContext	For details, see “CmdbContext” on page 85.
queryName	The name of the parameterized TQL in the UCMDB for which to get the map.
parameterizedNodes	The conditions each node must meet to be included in the query results.
queryTypedProperties	A collection of sets of properties to retrieve to items of a specific Configuration Item Type.

Output

Parameter	Comment
topologyMap	For details, see “TopologyMap” on page 90.
chunkInfo	For details, see: “ChunkInfo” on page 90, “Processing Large Responses” on page 24.

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 85.
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 90.
chunkInfo	For details, see "ChunkInfo" on page 90 and "Processing Large Responses" on page 24.

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 85.
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.

 getCINeighbours

The getCINeighbours method returns the immediate neighbors of the specified CI.

For example, if the query is on the neighbors of CI A, and CI A contains CI B which uses CI C, CI B is returned, but CI C is not. That is, only neighbors of the specified type are returned.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 85.
ID	The ID of the CI with which to retrieve the neighbors. This must be a real UCMDB ID.

Parameter	Comment
neighbourType	The CIT name of the neighbors to retrieve. Neighbors of the specified type and of types derived from that type are returned. For details, see “Type Name” on page 87.
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 27.
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 27

Output

Parameter	Comment
topology	For details, see “Topology” on page 89.
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 85.
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 27.
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 90, "Processing Large Responses" on page 24.

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 85.

Parameter	Comment
type	The class name. For details, see "Type Name" on page 87.
properties	The data to be returned on each configuration item. 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 26.

Output

Parameter	Comment
CIs	Collection of CI elements.
chunkInfo	For details, see: "ChunkInfo" on page 90, "Processing Large Responses" on page 24.

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	Date 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. 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. **Note:** The name ChangedDuring is kept to preserve backward compatibility. In previous versions, the operator was used only with create and modify time properties.
Equal	String and numerical
EqualIgnoreCase	String
Greater	Numerical
GreaterEqual	Numerical
In	String, numerical, and list 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.

Operator	Type of Condition/Comments
InList	List The condition's value and the property's value are lists. 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.
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.

Operator	Type of Condition/Comments
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. sampleII extends the CI with two attributes, level and grade. This example sets up a query for the properties of sampleII that were inherited from sample by specifying them by name.

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

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 85.
type	The class name. For details, see "Type Name" on page 87. The type can be any of the types defined using the CI Type Manager. For details, see "CI Type Manager" in *Model Management*.
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 26.
conditions	A collection of name-value pairs and the operators that relate one to the other. For example, host_hostname like QA.
conditionsLogicalOperator	➤ **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 90 and “Processing Large Responses” on page 24.

 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 85.
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 specifed 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 85.
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 85.
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 90 and "Query the UCMDB" on page 22.

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 58.

Input

Parameter	Comment
cmdbContext	For details, see "CmdbContext" on page 85.
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 90.
comments	For internal use only.

Example of Handling 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.sizeCIList() ; 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 85.
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

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 updateExisiting is **false** and ignoreValidation is **true**, the add operation is performed without validating the CIs.

If updateExisiting 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
cmdbContext	For details, see "CmdbContext" on page 85.
updateExisting	Set to *true* to update items that already exist in the UCMDB. Set to *false* to throw an exception if any item already exists.
CIsAndRelationsUpdates	The items to update or create. For details, see "CIsAndRelationsUpdates" on page 27.
ignoreValidation	If is 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 47.
comments	For internal use only.

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
cmdbContext	For details, see "CmdbContext" on page 85.
CIsAndRelationsUpdates	The items to delete. For details, see "CIsAndRelationsUpdates" on page 27

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 85.
CIsAndRelationsUpdates	The items to update. For details, see "CIsAndRelationsUpdates" on page 27.
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 47.

UCMDB Impact Analysis Methods

This section provides information on the following methods:

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 85.
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 89.
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 85.
identifier	The key to the server response that was returned by calculateImpact.
relation	A Relation based on one of the ShallowRelations returned by calculateImpact in the impactTopology 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 85.
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

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

```
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 getCIsByTypeDemo() {
       GetCIsByType request = new GetCIsByType();
       //set cmdbcontext
       CmdbContext cmdbContext = getContext();
       request.setCmdbContext(cmdbContext);
       //set CIs type
       request.setType("anyType");
       //set CIs 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.setCIsTypedProperties(properties);
        try {
            GetCIsByIdResponse response =
                getStub().getCIsById(request);
            CIs cis = response.getCIs();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getFilteredCIsByTypeDemo() {
        GetFilteredCIsByType request = new GetFilteredCIsByType();
        CmdbContext cmdbContext = getContext();
        //set cmdbcontext
        request.setCmdbContext(cmdbContext);
        //set CIs type
        request.setType("anyType");
        //sets Filter conditions
        Conditions conditions = new Conditions();
        IntConditions intConditions = new IntConditions();
        IntCondition intCondition = new IntCondition();
        IntProp intProp = new IntProp();
        intProp.setName("int_attr1");
```

```
intProp.setValue(100);
intCondition.setCondition(intProp);
intCondition.setIntOperator
    (IntCondition.intOperatorEnum.Greater);
intConditions.addIntCondition(intCondition);
```

```
conditions.setIntConditions(intConditions);
request.setConditions(conditions);
//set logical operator for conditions
request.setConditionsLogicalOperator
    (GetFilteredCIsByType.conditionsLogicalOperatorEnum.AND);
//set CIs properties to be retrieved
CustomProperties customProperties =
    new CustomProperties();
PredefinedProperties predefinedProperties =
    new PredefinedProperties();
SimplePredefinedProperty simplePredefinedProperty =
    new SimplePredefinedProperty();
simplePredefinedProperty.setName
    (SimplePredefinedProperty.nameEnum.NAMING);
```

```
SimplePredefinedPropertyCollection
    simplePredefinedPropertyCollection =
        new SimplePredefinedPropertyCollection();
simplePredefinedPropertyCollection.
    addSimplePredefinedProperty
        (simplePredefinedProperty);
predefinedProperties.setSimplePredefinedProperties
    (simplePredefinedPropertyCollection);
customProperties.setPredefinedProperties
    (predefinedProperties);
```

```
request.setProperties(customProperties);
try {
    GetFilteredCIsByTypeResponse response =
        getStub().getFilteredCIsByType(request);
    TopologyMap map =
        getTopologyMapResultFromCIs
            (response.getCIs(), response.getChunkInfo());
```

```
        } catch (RemoteException e) {
            //handle exception
        } catch (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 getCINeighboursDemo() {
    GetCINeighbours request = new GetCINeighbours();
    //set cmdbcontext
    CmdbContext cmdbContext = getContext();
    request.setCmdbContext(cmdbContext);
    // set CI id
    ID id = new ID();
    id.setBase("cmdbobjectidCIT1");
    request.setID(id);
    //set neighbour type
    request.setNeighbourType("neighbourType");
    //set Neighbours CIs propeties to be retrieved
    TypedPropertiesCollection properties =
        new TypedPropertiesCollection();
    TypedProperties typedProperties1 = new TypedProperties();
    typedProperties1.setType("neighbourType");
    CustomTypedProperties customProperties1 =
        new CustomTypedProperties();
    PredefinedTypedProperties predefinedProperties1 =
        new PredefinedTypedProperties();
```

```
QualifierProperties qualifierProperties =
    new QualifierProperties();
qualifierProperties.addQualifierName("ID_ATTRIBUTE");
predefinedProperties1.setQualifierProperties(qualifierProperties);
customProperties1.setPredefinedTypedProperties
    (predefinedProperties1);
typedProperties1.setProperties(customProperties1);
properties.addTypedProperties(typedProperties1);
request.setCIProperties(properties);
```

```
TypedPropertiesCollection relationsProperties =
    new TypedPropertiesCollection();
TypedProperties typedProperties2 = new TypedProperties();
typedProperties2.setType("relationType");
CustomTypedProperties customProperties2 =
    new CustomTypedProperties();
```

```
PredefinedTypedProperties predefinedProperties2 =
    new PredefinedTypedProperties();
SimpleTypedPredefinedProperty simplePredefinedProperty2 =
    new SimpleTypedPredefinedProperty();
simplePredefinedProperty2.setName
```

```
    (SimpleTypedPredefinedProperty.nameEnum.CONCRETE);
SimpleTypedPredefinedPropertyCollection
    simplePredefinedPropertyCollection2 =
        new SimpleTypedPredefinedPropertyCollection();
simplePredefinedPropertyCollection2.
    addSimpleTypedPredefinedProperty
        (simplePredefinedProperty2);
predefinedProperties2.
    setSimpleTypedPredefinedProperties
        (simplePredefinedPropertyCollection2);
customProperties2.setPredefinedTypedProperties
    (predefinedProperties2);
typedProperties2.setProperties(customProperties2);
relationsProperties.addTypedProperties(typedProperties2);
request.setRelationProperties(relationsProperties);
```

```
        try {
            GetCINeighboursResponse response =
                getStub().getCINeighbours(request);
            Topology topology = response.getTopology();
    } catch (RemoteException e) {
            //handle exception
        } catch (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())){
```

```
                CIs cisTOAdd = ciNode.getCIs();
                CIs cis =
                    mergeCIsGroups
                    (topologyMap.getCINodes().getCINode(j).getCIs(),
                        cisTOAdd);
                topologyMap.getCINodes().getCINode(j).setCIs(cis);
                alreadyExist = true;
            }
        }
        if(!alreadyExist) {
            topologyMap.getCINodes().addCINode(ciNode);
        }

    }
```

```
    for(int i=0 ; i < newMap.getRelationNodes().sizeRelationNodeList() ; i++ ) {
        RelationNode relationNode =
            newMap.getRelationNodes().getRelationNode(i);
        boolean alreadyExist = false;
        if(topologyMap.getRelationNodes() == null) {
            topologyMap.setRelationNodes(new RelationNodes());
        }
```

```
            for(int j=0 ;
                    j < topologyMap.getRelationNodes().sizeRelationNodeList() ;
                    j++) {
                RelationNode relationNode2 =
                    topologyMap.getRelationNodes().getRelationNode(j);
                if(relationNode2.getLabel().equals(relationNode.getLabel())){
                    Relations relationsTOAdd = relationNode.getRelations();
                    Relations relations =
                        mergeRelationsGroups
                        (topologyMap.getRelationNodes().
                            getRelationNode(j).getRelations(),
                            relationsTOAdd);
                    topologyMap.getRelationNodes().
                        getRelationNode(j).setRelations(relations);
                    alreadyExist = true;
                }
            }
```

```
            if(!alreadyExist) {
                topologyMap.getRelationNodes().addRelationNode(relationNode);
            }
        }

        return topologyMap;

    }
```

```
    private Relations mergeRelationsGroups(Relations relations1, Relations
relations2) {
        for(int i=0 ; i < relations2.sizeRelationList() ; i++) {
            relations1.addRelation(relations2.getRelation(i));
        }
        return relations2;
    }
```

```
    private CIs mergeCIsGroups(CIs cis1, CIs 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;
```

```
/**
 * User: hbarkai
 * 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

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 *Model Management*. For information about identifying the key attributes from within the API client application, see "getCmdbClassDefinition" on page 30.

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 *Model Management*.

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 88 and "Relation" on page 88.

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 *Model Management*.

The props element is a CIProperties collection. For details, see "CIProperties" on page 87.

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 UCMBD 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 29.

For details, see "CI Type Manager" in *Model Management*.

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 87.

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

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 89).
- relationNodes. This is one or more RelationNode (see "RelationNode" on page 90).

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 24.

3

The HP Universal CMDB Java API

This chapter explains how third-party or custom tools can use the HP Universal CMDB Java API to extract data and calculations and to write data to the UCMDB (Universal Configuration Management database).

Use this chapter in conjunction with the API Javadoc, available in the online Documentation Library.

This chapter includes:

Concepts

Tasks

Reference

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 Java API

The HP Universal CMDB Java 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 *Model Management*.

Users of the HP Universal CMDB Java API should be familiar with:

- The Java programming language
- HP Universal CMDB

This section includes the following topics:

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 can use the Impact Analysis methods for change analysis.

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 more information, see “Create an Integration User” on page 95.

General Structure of 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 UcmdbServiceProvider interface. The client communicates with the server over HTTP.

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
getClassModelService	Information about types of CIs and Relations
getImpactAnalysisService	Analysing the effect of a change in the IT universe
getTopologyQueryService	Getting information about the IT universe
getTopologyUpdateService	Changing the information in the IT universe

Retrieve the API Jar File

Get the ucmdb-api.jar from a BAC or UCMDB server installation. Extract the jar file from ucmdb-api.war, found in the AppServer\webapps directory on the server. Inside the war archive, ucmdb-api.jar is located in theWEB-INF\lib directory.

Compile and run your application with ucmdb-api.jar in the classpath.

Create an Integration User

Applications written with this API set log on with an integration user.

To create an integration user:

1. Log on to the JMX Agent console.
   a. The URL is http://<UCMDB host>:8080/jmx-console.
   b. The default user and password are both "admin".
2. On the JMX Agent View page, locate the section labeled "MAM".
3. Click **service=MAM Security Services**.
4. On the JMX MBean View, locate "java.lang.String createIntegrationUser()"
   a. Fill in the userName and password fields.
   b. Click **Invoke**.
5. Either click **Back to MBean View** to create more users, or close the JMX Agent Console.
6. Log on the the UCMDB as an administator.
7. From the **Settings** tab, run **Package Manager**.
8. Click the **New** icon.
9. Enter a name for the new package, and click **Next**.
10. In the Resource Selection tab, under Settings, click **Integration Users**.
11. Select a user or users that you created using the JMX Agent console.
12. Click **Next** and then **Finish**. Your new package appears in the Package Name list in the Package Manager.
13. Deploy the package to the users who will run the API applications.

    For details, see “Deploy a Package” on page 466.

HP Universal CMDB Java API Reference

For full documentation on the available APIs, refer to the HP UCMDB Java 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\UCMDB_JavaAPI\index.html

Use Cases

The following use cases assume two systems:

- HP Universal CMDB server
- A third-party system that contains a repository of configuration items

This section includes the following topics:

Populating the 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:

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 demonstate 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_l
abel");
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());
}
```

4

The Discovery and Dependency Mapping Web Service API

For details, see "The HP Discovery and Dependency Mapping Web Service API" in *Discovery and Dependency Mapping Guide*.

Part II

Federation and Reconciliation

5

Introduction to Federated CMDB

This chapter provides information on the Federated CMDB functionality.

This chapter includes:

Concepts

Tasks

Reference

Federated CMDB – Overview

CMDB implementations often involve federation, which is the inclusion of data in the CMDB from other sources, in such a way that the source of the data retains control of the data.

You use federated CMDB to answer the following types of questions:

- Which hosts in a specific application (for example, SAP) have changed more than a certain number of times in a certain time period?
- Which application names have changed more than a certain number of times in a certain time period?
- Which hosts in the model have changed more than a certain number of times in a certain time period?

Adapters

You set up adapters to work with data that is federated from different CMDB sources, using the HP Universal CMDB API. For details, see "The HP Universal CMDB Web Service API" in *Reference Information*.

You can use the following, predefined adapters to federate different CMDB sources:

- **CmdbChangesAdapter.** Select to define a source adapter that queries the UCMDB for changes. The adapter runs on federated TQLs and the UCMDB History database/schema. (For details on limitations, see "The CmdbChanges Adapter" on page 272.)
- **CmdbHistoryAdapter.** Select to define an adapter that retrieves data from a UCMDB History database. Note: the History Adapter does not support replication, that is, you cannot use this adapter to define a replication job (to copy data from one data store to another one).
- **CmdbRmiAdapter.** Select to define an adapter that uses an RMI API to retrieve data from an external CMDB data store. Note: This adapter supports federated replication only by default. To support federated queries, contact HP Software Support.

- **CmdbSoapAdapter.** Select to define an adapter that uses a SOAP API to retrieve data from an external CMDB data store. Note: This adapter does not support federated queries and can be used only as a target in federated replication.
- **ServiceDeskAdapter.** Select to define an adapter that 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. For details on working with this adapter, see Chapter 8, "The HP ServiceCenter/Service Manager Adapter."
- **GenericDBAdapter.** Platform for configuring a database adapter that integrates with any type of relational database and enables running federated TQLs against the databases. For details on working with this adapter, see Chapter 6, "The Generic Database Adapter."

  The Federation Framework SDK acts as a mediator between HP Universal CMDB and the adapters; and as a container for the federated adapters. For details, see "The Federation Framework SDK" on page 207.

  For details on choosing an adapter when creating a data store, see "Data Stores Tab" on page 115.

Retrieving Data from Multiple Data Stores

During FTQL calculation, you can retrieve data for the same CIT from several data stores. The data can be retrieved from the local UCMDB and from an external data store, or from several external data stores.

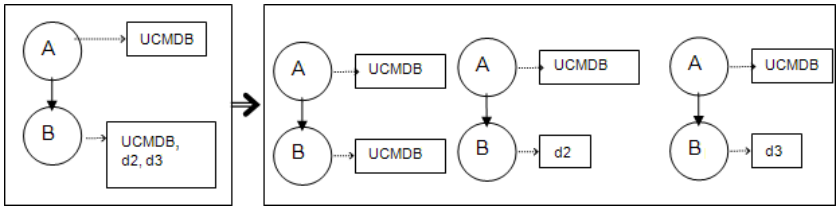
The data can be retrieved by the same adapter from several locations or by several adapters from several locations.

Each CI that is retrieved from an external data store includes an attribute (Created By) to show from which data store the CI has been retrieved.

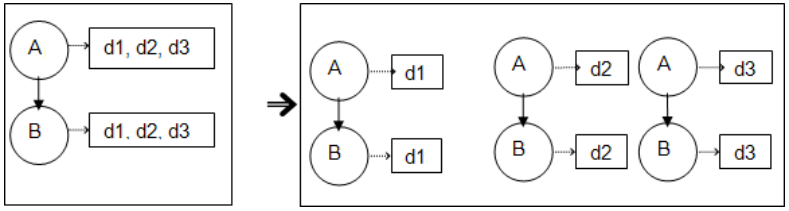
For details on mapping information, see "Mapping Information" on page 111.

Limitations

- When a virtual link exists between two data stores, HP Universal CMDB supports mapping in the following cases only:
  - The UCMDB data store lies at one end of the link and multiple data stores lie at the other end. The Cartesian product is calculated for A's data store (UCMDB) and B's data stores (UCMDB, d2, d3).



  - The same data stores lie at both ends of the link. The link is an internal link of each data store and no mapping is required.



- Data that is retrieved from several data stores is not reconciled. If two data stores include the same CIT, the CIT appears twice in HP Universal CMDB. If two data stores include the same attribute, only one of the attributes is federated with the UCMDB.

Retrieving Attributes from an External Data Store

- You can retrieve the attributes of a CI from an external data store, when the core CI data is stored in the UCMDB.
- The core data store must be the UCMDB.
- The CIT must be located in a data store for its attributes to be defined.
- The same attributes can be retrieved from multiple data stores.

For details on retrieval options, see the CI Type Retrieval Mode field in the "Supported CI Types Tab" on page 120.

- You cannot map a CIT to an external CIT and to the external CIT's attributes.
- A CIT can support external attributes if the adapter (that is federating the CIT data) supports mapping information (reconciliation) for this CIT.

Use Cases

- You need to discover the SMS or Altiris desktops in your system. The desktop CIT is a core CIT and is already synchronized with the UCMDB. However, you do not want to store all the desktop data in the UCMDB as this is inefficient and unnecessary. It is enough to store core attributes such as name and MAC address in the UCMDB, and to define the other details of the desktops as external attributes in two data stores: SMS and Altiris.
- VMware creates virtual machines that contain a virtual machine monitor (hypervisor) that allocates hardware resources dynamically and transparently. Multiple operating systems can run concurrently on a single physical computer. Since the allocation resources (for example, memory) are dynamic, DDM cannot discover these resources (DDM runs once every 24 hours and the resource data can change hourly). To enable HP Universal CMDB to always be updated with real-time data, the solution is to divide the data into two: the core data of the virtual hosts should be discovered and placed in the UCMDB; the resource attributes should be retrieved from the external source. In this use case, the data for these attributes is retrieved from two data stores: UCDMB and VMware.

Mapping Information

Federated queries should use the Mapping Engine to unify the shallow CIs (the core attributes of a CI) from the core data store with the attributes from the external data store.

For details on the Mapping Engine, see "Federation Framework Flow for FTQL" on page 212.

For details on selecting attributes to be included in the federation, see "Supported CI Types Tab" on page 120.

Work with Federated Data – Workflow

This section explains how to set up and work with data that is federated from different CMDB sources, using the HP Universal CMDB application.

This section includes the following topics:

1 Prerequisites

Set up the adapter. For details, see "Add an Adapter for a New External Data Store" on page 230. For details on existing adapters, see "Adapters" on page 108.

2 Create a Data Store

Access **Admin** > **Settings** > **Federated CMDB**. Click the **Add** button to open the New Data Store dialog box. For details, see "New Data Store Wizard" on page 117.

3 Replicate the Data Store

Use the data store for replication. Access **Admin** > **Settings** > **Federated CMDB** > **Replication Jobs** tab. Click the **Add** button to open the Replication Job dialog box. For details, see "Replication Job Dialog Box" on page 123.

4 Build a View

For details, see "View Manager Overview" in *Model Management*.

5 View Instances in IT Universe Manager

For details, see "IT Universe Manager Overview" in *Model Management*.

6 View Reports

For details, see "Topology Report Manager" in *Model Management*.

Change the Encrypted Password of a Federated Adapter

When defining a new data store adapter, a user must enter a password. This password is encrypted by default and encryption is entirely transparent to the user.

The following procedures explain how to generate a new key file and change an existing key file. HP Universal CMDB includes a default key so that these procedures are optional.

Note: During the upgrade process from a version with unencrypted passwords, the passwords are encrypted.

The encryption key is located at **C:\hp\UCMDB\UCMDBServer\j2f\fcmdb\fkey.bin**.

To generate a new key file:

1 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 HP Universal CMDB is installed. You may have to log in with the user name and password.

2 Click the **Topaz** > **service=FCmdb Config Services** link.

3 In the JMX MBEAN View page, locate the following operation: **java.lang.String generateNewKeyFile()**.

4 In the **customer id** field, enter **1**.

5 Click **Invoke**. A message is displayed informing you that a new key file is generated and has been loaded into the application.

To change the key file:

1 Prepare a key file and note the location.

2 In the JMX MBEAN View page, locate the following operation: **java.lang.String importKeyFile()**.

3 In the **customerId** field, enter **1**; in the **newKeyFileLocation** field, enter the full path to the new key.

4 Click **Invoke**. A message is displayed.

Federated CMDB User Interface

This section describes:

Data Stores Tab

Description	Enables you to edit existing data stores or to create new stores. **To access: Admin > Settings > Federated CMDB > Data Stores tab**.
Important Information	Data Stores includes the following tabs: ➤ **Properties.** Enables you to configure an adapter that accesses external data sources. For details, see "Properties Tab" on page 118. ➤ **Supported CI Types.** Enables you to choose which CITs are to be supported by the data store. For example, if a federated TQL (FTQL) includes a node that represents a specific CIT, the instances of this CIT are accepted from this external data store. For details, see "Supported CI Types Tab" on page 120. ➤ **Supported Queries.** Enables you to choose which queries should be supported by this data store for data replication. For details, see "Supported Queries Tab" on page 122.

The following elements are included (unlabeled GUI elements are shown in angle brackets):

GUI Element (A-Z)	Description
	Add a data store. For details, see "New Data Store Wizard" on page 117.
	Delete a data store.
	Click to save changes.
 **Refresh**	Click to refresh the page.
	Click to reload, if changes have been made to the adapter.

GUI Element (A-Z)	Description
<List of Defined Adapters>	**Name.** The name you give to the data store. **Adapter.** The type of adapter for this data store.
View	For details, see: ➤ "Data Stores Tab" on page 115 ➤ "Supported CI Types Tab" on page 120

Federated CMDB Window

Description	Enables you to define external data stores and create replication jobs. **To access: Admin** > **Settings** > **Federated CMDB**.
Important Information	You create a data store with the New Data Store wizard. For details, see "New Data Store Wizard" on page 117.
Included in Tasks	"Work with Federated Data – Workflow" on page 112
Useful Links	➤ "Data Stores Tab" on page 115 ➤ "Replication Jobs Tab" on page 126 ➤ "Adapters" on page 108

New Data Store Wizard

Description	Enables you to define a data store that uses an adapter to access external data sources. **To access: Admin** > **Settings** > **Federated CMDB** > **Data Stores tab** > click the **Add** button.
Important Information	Depending on the adapter's function, use one of the following adapters to define a data store: ➤ **CmdbChangesAdapter**. The adapter supports federated replication only and can be used as a source, therefore, you must fill in the **Properties** tab and the **Supported Queries** tab. ➤ **CmdbHistoryAdapter**. The adapter supports federated TQL queries only, therefore you must fill in the **Properties** tab and the **Supported CI Types** tab. ➤ **CmdbRmiAdapter**. The adapter supports both federated replication and federated queries, therefore you must fill in the **Properties** tab, the **Supported CI Types** tab, and the **Supported Queries** tab. (By default, this adapter supports replication only.) ➤ **CmdbSoapAdapter**. The adapter supports federated replication only and can be used as a target, therefore you must fill in the **Properties** tab. ➤ **ServiceDeskAdapter**. The adapter supports federated TQL queries only, therefore you must fill in the **Properties** tab and the **Supported CI Types** tab. ➤ **GenericDBAdapter**.
Included in Tasks	"Work with Federated Data – Workflow" on page 112
Useful Links	➤ "Retrieving Data from Multiple Data Stores" on page 109 ➤ "Retrieving Attributes from an External Data Store" on page 110 ➤ "Adapters" on page 108

Properties Tab

Description	Enables you to select the type of adapter, and to define the connection information for the external data store you are defining. **To access: Admin** > **Settings** > **Federated CMDB** > **Data Stores** tab. Click the **New Data Store** button.
Important Information	All enabled fields are mandatory.
Useful Links	"Adapters" on page 108

The following elements are included (unlabeled GUI elements are shown in angle brackets):

GUI Element(A–Z)	Description
Adapter	Choose the adapter to retrieve the external data from an external data store.
Test connection button	Click to test that you have entered valid information.
Customer ID	For HP Universal CMDB, enter **1**. For HP Software-as-a-Service, enter the customer ID number.
Host	Enter the name of the machine to which the adapter must connect.
Name	Enter a name to identify the data store. The name must be unique in the HP Universal CMDB version.

GUI Element(A–Z)	Description
Next button	Click to continue defining the adapter. **Note:** If you did not test the connection, it is automatically tested when you click **Next**. If the adapter does not support a federated query and replication as a source (that is, the adapter supports replication only as a target, for example, the SOAP adapter), the following message is displayed:  Click **OK**, then click **Finish**.
Password	Enter the password needed to access the external data store to which you want to connect. If you set up the system to accept an encrypted password, the password you enter in this field is encrypted and saved to the database. For details on encrypted passwords, see “Change the Encrypted Password of a Federated Adapter” on page 113.
Port	Enter the port through which you are accessing the external data store (if required).
URL	Use this field to define a specific URL to connect to the external data store.
User	Enter the user name needed to access the external data store to which you want to connect.

Supported CI Types Tab

Description	Enables you to choose which CITs or attributes are to be supported by the data store. For example, if a federated TQL (FTQL) includes a node that represents a specific CIT, the instances of this CIT are accepted from this external data store. **To access: Admin** > **Settings** > **Federated CMDB** > **Data Stores** tab. Click the **Supported CI Types** tab.
Important Information	This page is displayed when the selected adapter supports federated queries.
Included in Tasks	"Work with Federated Data – Workflow" on page 112
Useful Links	"Retrieving Attributes from an External Data Store" on page 110

The following elements are included (unlabeled GUI elements are shown in angle brackets):

GUI Element (A-Z)	Description
	Represents instance federation, that is, CITs are federated.
	Represents attribute federation, that is, federation is run at the level of the attributes of a CIT.
<List of CITs>	This list includes all CITs that are supported by the adapter. When queried by an FTQL, the CITs you select here are configured to retrieve the data from this external data store. Select the CITs to be supported by this data store.

GUI Element (A-Z)	Description
CI Type Retrieval Mode	➤ **Retrieve CIs of selected CI Type.** All a CI's data, including all its attributes, are retrieved from the data store. **Retrieve CIs of the CI Type from the UCMDB too.** ➤ **Retrieve selected attributes.** The selected attributes are retrieved from the data store. The CIs must already exist in the UCMDB. **Retrieve the attribute from the UCMDB too.** **Note**: ➤ All CITs (and their child CITs) included in a data store definition must use the same retrieval mode. ➤ You cannot select both CITs and attributes for the same data store.
CI Types	The CIT hierarchy. Right-click the CIT to define a new data store of the same data store type.
Select Attributes	You can define which attributes of an external CIT are to be included in the federation: ➤ In the CI Type Retrieval Mode pane, select **Retrieve selected attributes**. ➤ In the Select Attributes list, select the attributes that are to be included in the federation. ➤ Save the changes. **Note**: Attributes are defined in the CIT Manager. For details, see "Add/Edit Attribute Dialog Box" on page 237.

Supported Queries Tab

Description	Displays a list of queries that are supported by the adapter. Enables you to choose which queries should be supported by this data store for data replication. **To access: Admin** > **Settings** > **Federated CMDB** > **Data Stores** tab. Click the **Supported Queries** tab.
Important Information	This page is displayed when an adapter is selected that supports federated replication and can be used as a source.
Included in Tasks	"Work with Federated Data – Workflow" on page 112

The following elements are included (unlabeled GUI elements are shown in angle brackets):

GUI Element (A-Z)	Description
	➤ **Select/Unselect All.** Click to select all queries in the list and click again to clear all queries. ➤ **Switch Selection.** Click to select all non-selected queries or to clear all selected queries. ➤ **Show Selected Queries.** Click to view a list of all selected queries.
<List of query names>	This list includes all queries that are supported by the adapter. During replication job configuration, queries you select here are visible. Select the queries to be included in the configured data store. **Visible.** The query is visible in the Replication Job dialog box for the current data store.

Replication Job Dialog Box

Description	Enables you to replicate data between two data stores. **To access: Admin** > **Settings** > **Federated CMDB** > **Replication Jobs**. Click the **Add** button.
Important Information	➤ For successful replication, if you use CMDB adapters for replication (SOAP or RMI), the CIT class model in the target (the machine to which the data is imported, to be federated) must be identical to the class model in the source (the machine from which the data is exported). ➤ To add specific CIs only from a TQL during a replication job, you must configure the required condition on those CIs. Then, you must configure the same conditions for the relationship TQLs that are linked to the CIs.
Included in Tasks	"Work with Federated Data – Workflow" on page 112

The following elements are included (unlabeled GUI elements are shown in angle brackets):

GUI Element (A-Z)	Description
Name	Enter a name to identify the replication job.
Replication Job Queries	➤ **Active.** Select to use this query in the replication job. ➤ **Name.** The query name. ➤ **Description.** The description of the query. ➤ **Permit Deletion in Target.** Permits deletion of CIs and relationships on the target machine if they have been deleted in the source query result.

GUI Element (A-Z)	Description
Source Data Store	The data store for the data that is to be brought over and replicated to the data on the target machine. Only data stores that have adapters that support replication and can be used as a source are displayed here. Click **Details** to view information about the data store.
Target Data Store	The data store to which the data that is brought in from the source machine should be replicated. Only data stores that have adapters that support replication and can be used as a target are displayed here.

Replication Job Statistics Window

Description	Enables you to view statistics for a selected replication job to verify whether replication is successful. **To access: Admin** > **Settings** > **Federated CMDB** > **Replication Jobs** tab. Select a replication job and click the **Statistics** icon in the toolbar.
Important Information	Click a row to view error messages for a specific replication job.

The following elements are included (unlabeled GUI elements are shown in angle brackets):

GUI Element (A-Z)	Description
Ad hoc	**Yes.** The replication job was run by Scheduler or by the user clicking the **Ad hoc** icon in the Replication Jobs tab.
Full replication	**Yes.** The user chose to retrieve all appropriate data in the source to the target, without taking the last run of the replication job into consideration.
Job Name	Name given to the replication job.

GUI Element (A-Z)	Description
Replication Job Statistics per Query	➤ **Query Name.** The name of the query that is used for replication. ➤ **Status.** Can be SUCCEEDED or FAILED. If the status is FAILED, click the query to display the Replication Job Error dialog box that contains the reason for the failure. For more detailed information, look at the fcmdb.synchronizer.log file, located in the following folder: <HP Universal CMDB root directory>\UCMDBServer \j2f\fcmdb ➤ **Replication Start Time.** The time the replication job started running replication for the current query. ➤ **Replication Stop Time.** The time the replication job stopped running replication for the current query. ➤ **Updated.** The number of CIs and relationships that were updated in the target data store during the last replication job. ➤ **Added.** The number of CIs and relationships that were added in the target data store during the last replication job. ➤ **Removed.** The number of CIs and relationships that were removed in the target data store during the last replication job. **Note:** The CIs and relationships are removed from the target only if **Permit Deletion in Target** is selected in the Replication Job definition. ➤ **Has Error. Yes.** Click to open the Replication Error dialog box that displays the reason for the failure.
Source Data Store	The ID of the source data store.
Status	Can be SUCCEEDED or FAILED.
Target Data Store	The ID of the target data store.

Replication Jobs Tab

Description	Enables you to define the replication jobs that contain the source data store, target data store, and queries, whose results should be replicated. **To access: Admin > Settings > Federated CMDB > Replication Jobs** tab.
Important Information	You can set up a schedule to run a replication job. For details, see "Scheduler Actions" on page 496.

The following elements are included (unlabeled GUI elements are shown in angle brackets):

GUI Element (A-Z)	Description
New Replication Job	Click to add a replication job. For details, see "Replication Job Dialog Box" on page 123.
Delete Selected Items	Click to delete a replication job.
New Data Store	Click to open the New Data Store dialog box. For details, see "New Data Store Wizard" on page 117.
Save	Click to save the changes.
Refresh	Click to refresh the page.
	Click to test that replication is successful: ➤ **Ad hoc diff replication.** The result from the source data store is compared to the result from the last run of the replication job; only changes are sent to the target data store. ➤ **Ad hoc full replication.** All appropriate data in the source is brought to the target, without taking the last run of the replication job into consideration. **Note:** This feature is useful when testing a replication job as you are developing it. During production, you should use the Scheduler to run the replication job. For details, see "Schedule Tab" on page 127.

GUI Element (A-Z)	Description
 **Statistics**	Click to view replication job statistics about the selected replication job. For details, see "Replication Job Statistics Window" on page 124.
	To filter the list of tasks, choose from the list and type in the first letters in the **by** box.
<right-click menu>	For details, see the explanations for the buttons in this table. **Duplicate Replication Job**. Enter a name for the duplicate job and make changes to it as necessary. Save the job.
Tasks Table	**Name.** The name you give to the replication job. **Source Data Store.** The ID of the source data store. **Target Data Store.** The ID of the target data store. **Queries.** The queries that are used for replication in this replication job.

Properties Tab

Description	Enables you to edit a replication job. **To access: Admin > Settings > Federated CMDB > Replication Jobs** tab. Click the **Properties** tab.
Important Information	For details, see "Replication Job Dialog Box" on page 123.

Schedule Tab

Description	Enables you to set a schedule to run replication jobs. **To access: Admin > Settings > Federated CMDB > Replication Jobs** tab. Click the **Schedule** tab.
Useful Links	Chapter 15, "Scheduler"

The following elements are included (unlabeled GUI elements are shown in angle brackets):

GUI Element (A-Z)	Description
	Click to add a schedule. For details, see "Job Definitions Dialog Box" on page 497. For details on setting a replication job action, see "Action Definition Dialog Box" on page 495.
	Click to edit an existing schedule. For details, see "Job Definitions Dialog Box" on page 497.
	Delete a schedule.
Active	The schedule runs at the defined time.
Job Definition	A description of the job.
Last Run Time	The previous time that the schedule ran.
Name	The name of the schedule.
Next Run Time	The date and time that the schedule will next run.
Schedule	The description of the schedule.

6

The Generic Database Adapter

This chapter describes the generic database adapter functionality, usages, and deployment in HP Universal CMDB.

This chapter includes:

Concepts

Tasks

Reference

Database Adapter – Overview

The purpose of this adapter is to enable integration with any kind of relational database management system (RDBMS) and to run a federated TQL query and replication jobs against the database.

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.

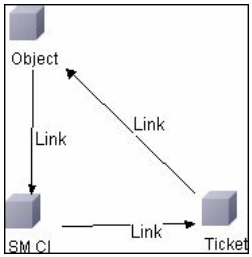
The databases supported by the generic database adapter are Oracle, Microsoft SQL Server, and MySQL.

Non-supported TQL Queries

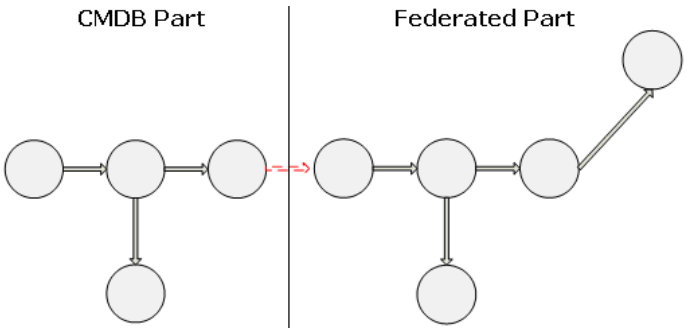
The following limitations exist on the federated CMDB only:

- 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 multinode.

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 144.
- Reconciliation with CMDB id is also supported.

Reconciliation rules take the form of OR and AND conditions. You can define these rules on several different nodes (for example, host is identified by host_name from host AND/OR ip_address from ip).

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.

  host_name OR ip_address. If both the UCMDB and the data source include host_name and they are equal, the hosts are considered as equal. If both have host_name but they are not equal, the hosts are considered not equal without testing the ip_address. If either the UCMDB or the data source is missing host_name, the 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.

  host_name OR ip_address. If there is no match on host_name, 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 host), the mapping of a superset node includes all relevant attributes from all modeled CITs.

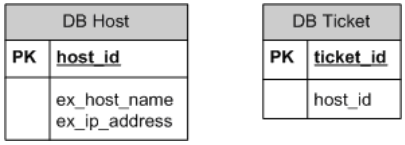
Note: As a result, there is a limitation that all reconciliation attributes in the data source should reside in tables that share the same primary key.

Another limitation states that the reconciliation TQL should have no more than two nodes. For example, the host > ticket TQL has a host in the UCMDB and a ticket in the data source.

To reconcile the results, host_name must be retrieved from the host and/or ip_address must be retrieved from the IP address. A new mapping is made from this federated multinode host towards the database host table and from the ticket to the database ticket. In this case, the multinode is the superset of all attributes needed for reconciliation (host_name + ip_address).

If the host_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 host_id column in the database ticket table is used to connect between the two entities (the defined association can also be made in a host table):



Note: Both tables must be part of the federated RDBMS source and not the CMDB database.

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 206.

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 135.

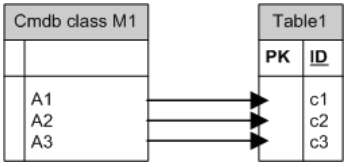
For our purposes, there is no need to create Java classes. The mapping is defined from the CMDB class model CITs to the database tables.

Examples of Object-Relational Mapping

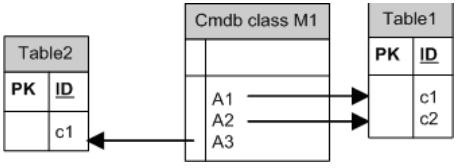
The following examples describe object-relational mapping:

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 raw in table 1.

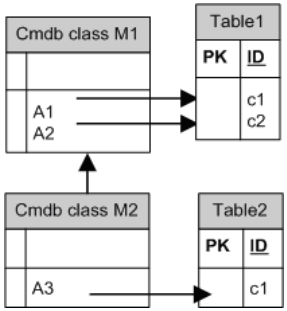


Example of 1 CMDB class mapped to 2 database tables



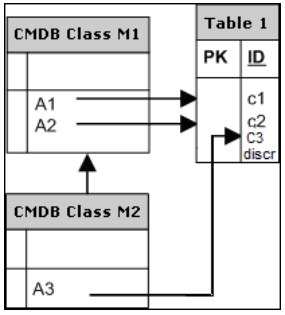
Example of inheritance

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.

Deploy a Database Adapter – Minimal Method

Note: When building an adapter for the first time, you should use this method. 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 later 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 host attribute.
- Demonstrate the Generic Database Adapter capabilities.

This method:

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

This task includes the following steps:

1 Prerequisites

For details, see Prerequisites in "Deploy a Database Adapter – Advanced Method" on page 143.

2 Extract the Database Adapter Configuration File

For details, see "Extract the Database Adapter Configuration File" on page 147.

3 Deploy the Adapter Package

For details, see "Deploy the Adapter Package" on page 150.

4 Deploy the Adapter

For details, see "Deploy the Adapter" on page 151.

5 Create a CI Type

For details, see "Create a CI Type" on page 153.

6 Create the Relationships

For details, see "Create the Relationships" on page 155.

7 Configure the adapter.conf File

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

a Open the following file in a text editor: **<HP Universal CMDB root directory>\j2f\fcmdb\Codebase\MyAdapter\META-INF\adapter.conf**.

b Locate the following line: **use.simplified.xml.config=<true/false>**.

c Change to **use.simplified.xml.config=true**.

8 Configure the simplifiedConfiguration.xml File

In this step you configure the simplifiedConfiguration.xml file by mapping between the CIT in the UCMDB and the fields in the RDBMS table.

a Open the following file in a text editor: **<HP Universal CMDB root directory>\j2f\fcmdb\Codebase\MyAdapter\META-INF\ simplifiedConfiguration.xml**.

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

```
<cmdb-class cmdb-class-name="host" default-table-name="[table_name]">
      <primary-key column-name="[column_name]"/>
      <reconciliation-by-two-nodes connected-node-cmdb-class-name="ip" cmdb-
link-type="contained">
            <or is-ordered="true">
                  <attribute cmdb-attribute-name="host_hostname" column-
name="[column_name]" ignore-case="true"/>
                  <connected-node-attribute cmdb-attribute-name="ip_address"
column-name="[column_name]"/>
            </or>
      </reconciliation-by-two-nodes>
</cmdb-class>

<class cmdb-class-name="[cmdb_class_name]" default-table-
name="[default_table_name]" connected-cmdb-class-name="host" link-class-
name="container_f">
      <foreign-primary-key column-name="[column_name]" cmdb-class-primary-key-
column="[column_name]"/>
      <primary-key column-name="[column_name]"/>
      <attribute cmdb-attribute-name="[cmdb_attribute_name]" column-
name="[column_name]" from-cmdb-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.Generi
cEnumTransformer(generic-enum-transformer-example.xml)" to-cmdb-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.Generi
cEnumTransformer(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>
```

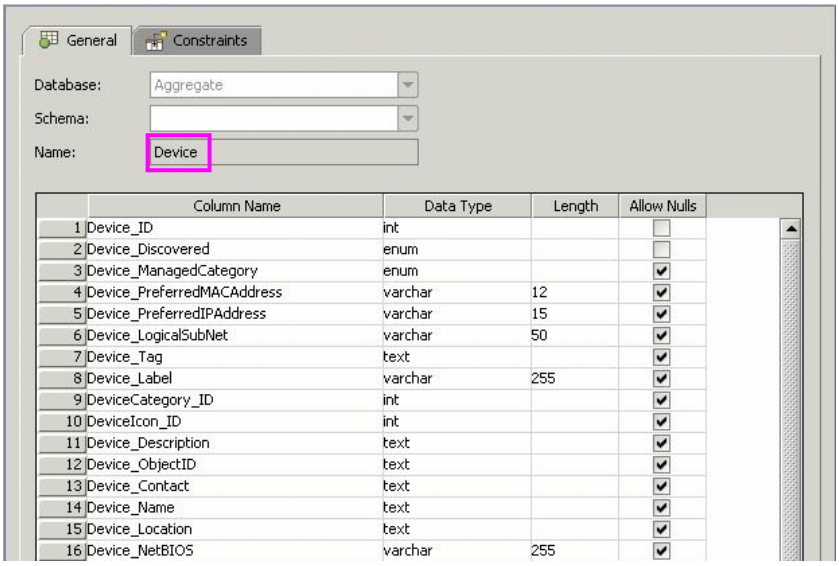
c Make changes to the attributes as follows:

➤ 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="host" default-table-name="Device">
```

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

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



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

- 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" cmdb-link-type="contained">
```

- The reconciliation attribute in Universal CMDB (cmdb-attribute-name) and in the RDBMS (column-name):

```
<connected-node-attribute cmdb-attribute-name="ip_address" 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="host" link-class-name="container_f">
```

- 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"/>
```

d Save the file.

9 Continue with the Procedure

For the rest of the procedure, see the following sections:

- ➤ "Load the Adapter" on page 162
- ➤ "Create a Data Store" on page 163
- ➤ "Create a View" on page 163
- ➤ "Calculate the Results" on page 165
- ➤ "View Results" on page 165
- ➤ "View Reports" on page 167
- ➤ "Enable Log Files" on page 167

Deploy a Database Adapter – Advanced Method

The following procedure describes the complete method of mapping the class model in the UCMDB to tables in an RDBMS.

This task includes the following steps:

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 host name, verify that there is a table that contains a column with host names. If the reconciliation is run according to host cmdb_id, verify that there is a column with CMDB IDs that matches the CMDB IDs of the hosts in the CMDB. For details on reconciliation, see "Reconciliation" on page 131.

ID	HOST_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 host and ticket, there must be a column in the ticket table that contains the host ID, a column in the host table with the ticket ID that is connected to it, or a mapping table whose end1 is the host ID and end2 is the ticket ID. For details on correlation data, see "Hibernate as JPA Provider" on page 134.

The following table shows the foreign key HOST_ID column:

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

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

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

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

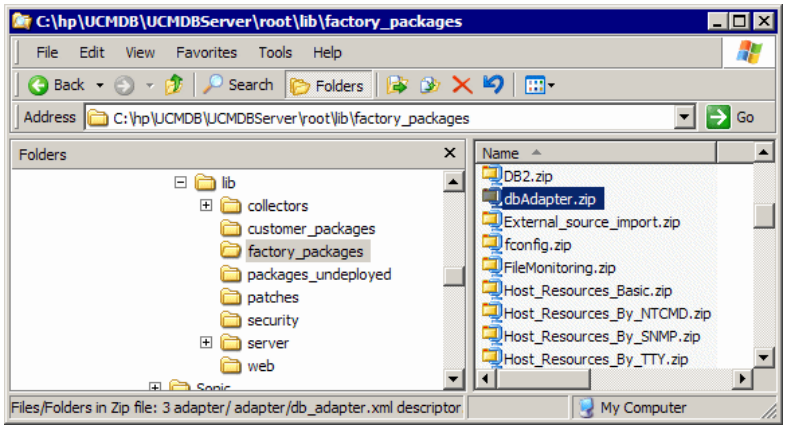
CARD_ID	CARD_TYPE	CARD_NAME
1	Serial Bus Controller	Intel ® 82801EB USB Universal Host Controller
2	System	Intel ® 631xESB/6321ESB/3100 Chipset LPC
3	Display	ATI ES1000
4	Base System Peripheral	HP ProLiant iLO 2 Legacy Support Function

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

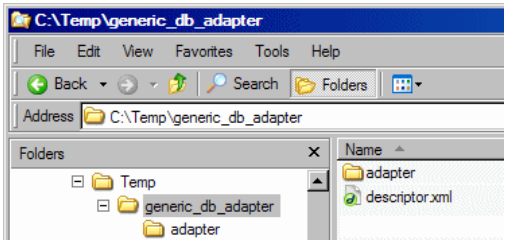
2 Extract the Database Adapter Configuration File

In this step, you locate the generic database adapter package and make a copy of it.

a Locate the **dbAdapter** package in **<HP Universal CMDB root directory>\UCMDBServer\root\lib\factory_packages**.



b Extract the package to a local temporary directory:



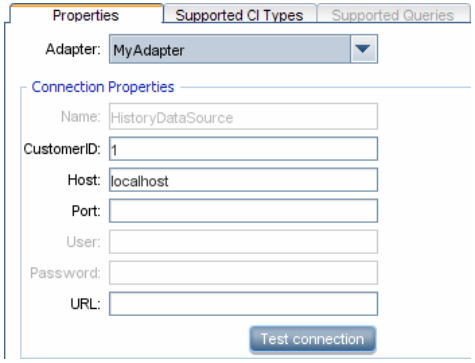
c Open the **adapter\db_adapter.xml** file in a text editor.

This file includes the fields that are needed to connect to the data store. You can use this file as a template (recommended) or you can create a new XML file from scratch.

d Locate the **adapter-id** attribute and replace the name:

```
<adapter-config adapter-id="MyAdapter">
      <class-name>com.mercury.topaz.fcmdb.adapters.dbAdapter.DBAdapter
</class-name>
            <adapter-capabilities>
                  <support-federated-query>
                        <supported-classes>
                        </supported-classes>
                              <topology>
                                    <pattern-topology>
                                    </pattern-topology>
                              </topology>
                  </support-federated-query>
            </adapter-capabilities>
            <fields-to-connect>
                  <field>host</field>
                  <field>customerId</field>
                  <field>port</field>
                  <field>url</field>
            </fields-to-connect>
            <default-mapping-engine>
com.mercury.topaz.fcmdb.adapters.dbAdapter.reconciliation.mapping_engine.
DBMappingEngine</default-mapping-engine>
</adapter-config>
```

There are no restrictions on the name, such as case or special characters. The name you enter here appears in the list of adapters in the Data Stores pane in HP Universal CMDB:

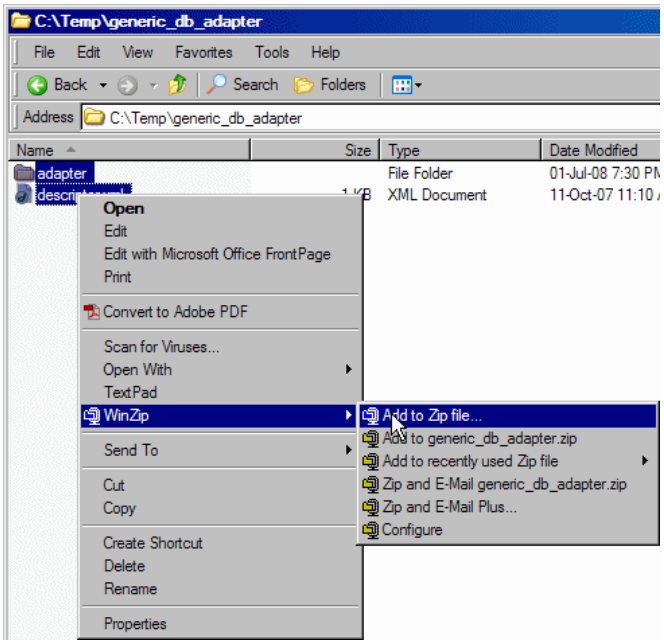


For details, see “Data Stores Tab” on page 115.

Important: For consistency's sake and for ease of use, use this name when saving the file and when defining the adapter.

---

e Create a *.zip file:



f Give the zip file the same name as you gave to the **attribute-id** attribute, as described in step d on page 148.

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

---

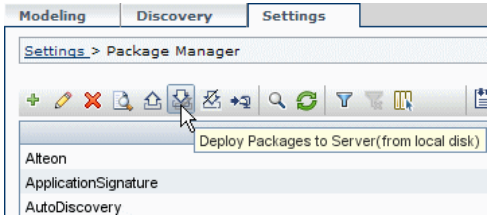
3 Deploy the Adapter Package

In this step you deploy the package in Package Manager.

a Save the new package that you created in the previous step in the following folder: **<HP Universal CMDB root directory>\UCMDBServer \root\lib\customer_packages**.

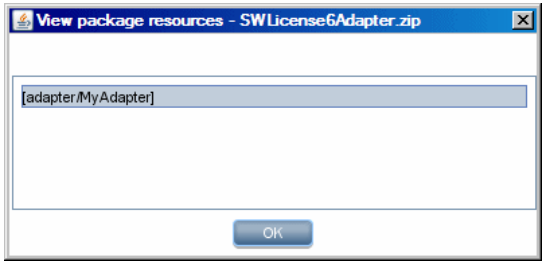
b In HP Universal CMDB, access Package Manager. For details, see "Package Manager Window" on page 483.

c Deploy the adapter: click the **Deploy Packages to Server** icon:



Click **Add** and browse to your adapter package. Click **Open** then **OK** to display the package in the Package Manager.

d Verify that the XML file contents is recognized by Package Manager: select your package in the list and click **View package resources**.

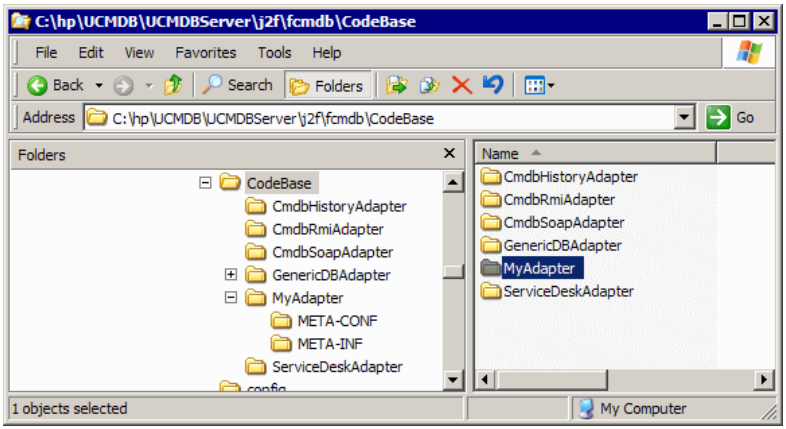


The package contains the adapter XML file only.

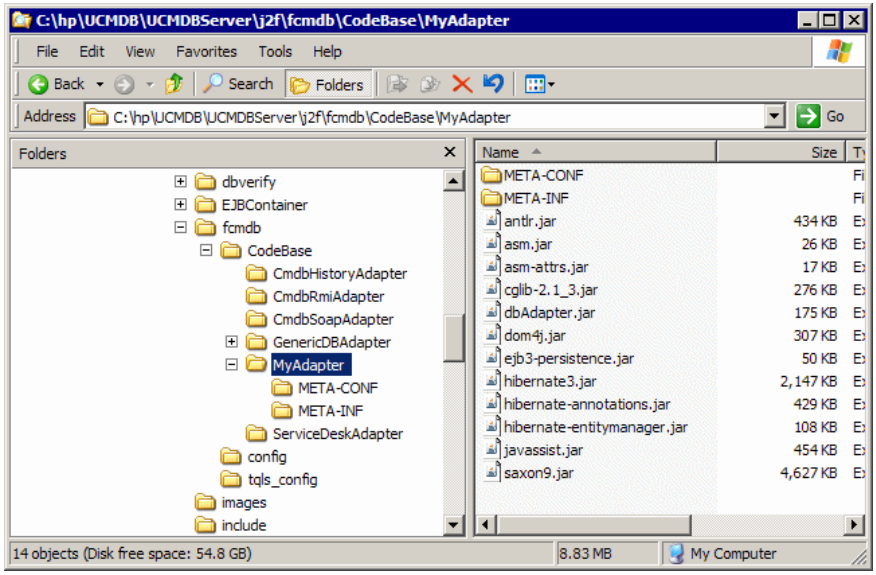
4 Deploy the Adapter

In this step, you create the connection between the logic and the definition of the adapter.

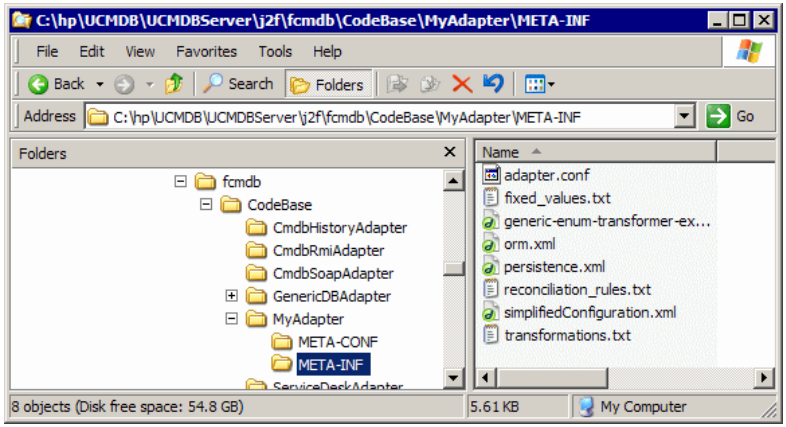
a Copy the generic database adapter folder under the **<HP Universal CMDB root directory>\j2f\fcmdb\Codebase** folder and rename it to the name of your adapter (the same name that you gave to the adapter-id field you changed in step d on page 148):



This folder contains the jar files that execute the federation logic, for example, the adapter name, the queries and classes in Universal CMDB and the fields in the RDBMS that the adapter supports.



b Open the new adapter folder and drill down to the META-INF folder:



This folder contains the files that run the adapter. The **orm.xml** file maps the Universal CMDB class model to the actual columns and tables in the RDBMS.

5 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 HP Universal CMDB, access the CI Type Manager and create a new CI Type. For details, see "Create a CI Type" on page 229.

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 HP Universal CMDB views.

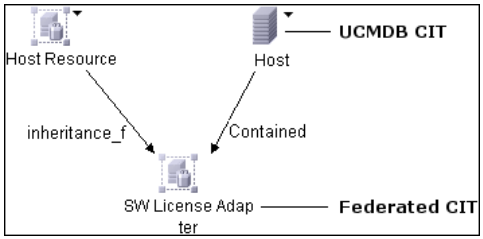
Example of creating a host_card CIT

```
<?xml version="1.0" encoding="UTF-8"?>
<Class class-name="host_card" display-name="Host Card" description="">
 <Class-Qualifiers/>
 <Class-Type>OBJECT</Class-Type>
 <Derived-From class-name="hostresource"/>
 <Attributes>
      <Attribute name="card_class" display-name="Card Class" description=""
type="string">
           <Attribute-Qualifiers/>
      </Attribute>
      <Attribute name="card_vendor" display-name="Card Vendor" description=""
type="string">
           <Attribute-Qualifiers/>
      </Attribute>
      <Attribute name="card_name" display-name="Card Name" description=""
type="string">
           <Attribute-Qualifiers/>
      </Attribute>
 </Attributes>
 <Attribute-Overrides>
      <Attribute-Override name="display_label" is-partially-override="true">
           <Attribute-Qualifiers>
                <Attribute-Qualifier name="CALCULATED_ATTRIBUTE">
                     <Data-Items>
                          <Data-Item name="FUNCTION"
type="string">card_name</Data-Item>
                     </Data-Items>
                </Attribute-Qualifier>
           </Attribute-Qualifiers>
      </Attribute-Override>
 </Attribute-Overrides>
</Class>
```

6 Create the Relationships

In this step you add a relationship between the HP Universal CMDB CIT and the new CIT that represents the data to be federated from the external data source.

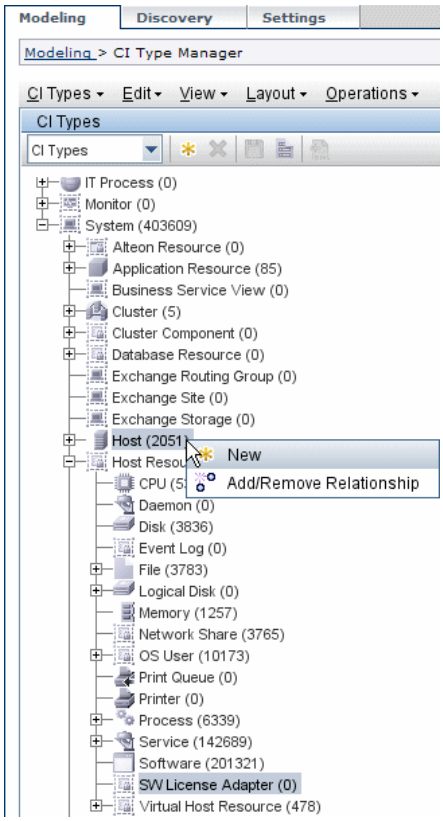
Add appropriate, valid relationships to the new CIT. For details, see "Add/Remove Relationship Dialog Box" on page 240.



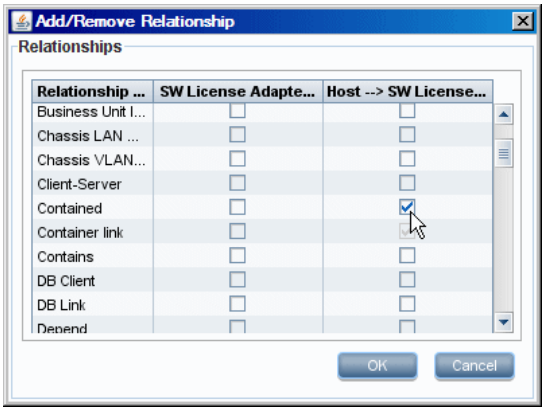
Note: At this stage, you cannot yet view the federated data, as you have not yet defined the method for bringing in the data.

Example of creating a Contained relationship

a In the CIT Manager, select the two CITs:



b Create a **Contained** relationship between the two CITs:



Example of Contained Relationship between host_card and host

```
<?xml version="1.0" encoding="UTF-8"?>
<Valid-Links>
 <Valid-Link>
      <Class-Ref class-name="contained"/>
      <End1 class-name="host"/>
      <End2 class-name="host_card"/>
 </Valid-Link>
</Valid-Links>
```

7 Configure the orm.xml File

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

Tip: We recommend that when building an adapter for the first time, you use the simplified method. The **orm.xml** file that is automatically generated as a result of running that method is a good example that you can use now when working with the advanced method. For details, see "Deploy a Database Adapter – Minimal Method" on page 136.

---

a Open **orm.xml** in an XML or text editor. This file, by default, contains a template that you use to map as many CITs and tables as needed for the federation.

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

For details of naming conventions, see "Naming Conventions" on page 181.

Example of Entity Mapping Between the Class Model and the RDBMS

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

➤ The name and class of the Universal CMDB CIT.

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

➤ 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="host_hostname">
```

➤ 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 153.

```
<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:

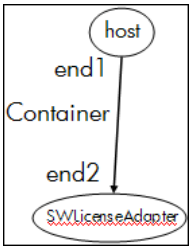
```
<basic name="license_required">
      <column updatable="false" insertable="false"
name="MyAdapter_LicenseRequired"/>
```

8 Map the Relationships

In this step you map relationships in Universal CMDB to the relationships in the RDBMS. For details on relationships, see "Example of Relationship Mapping Between the Class Model and the RDBMS" on page 161.

You make changes to the orm.xml file according to the relationships to be mapped. You define the same relationships in Universal CMDB as exist in the RDBMS, according to the use case. The following relationships can be mapped:

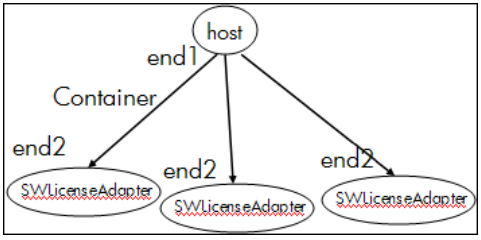
- One to one:



The code for this type of relationship is:

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

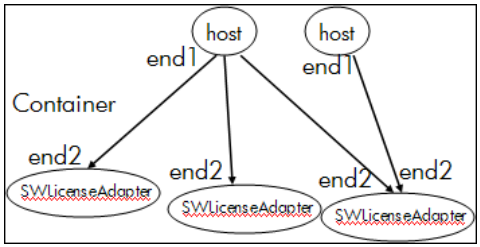
- Many to one:



The code for this type of relationship is:

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

Example of Relationship Mapping Between the Class Model and the RDBMS

➤ The name and class of the Universal CMDB relationship:

```
<entity name="host_contained_MyAdapter"
class="generic_db_adapter.host_contained_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="host" 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"/>
```

9 Configure the reconciliation_rules.txt File

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

a Open **META-INF\reconciliation_rules.text** in a text editor.

b Make changes to the file according to the CIT you are mapping. For example, to map a host CIT, use the following expression:

```
multinode[host] ordered expression[^host_hostname]
```

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.

---

10 Load the Adapter

In this step you load the adapter onto the HP Universal CMDB machine.

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

---

a 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 HP Universal CMDB is installed. You may have to log in with the user name and password.

b Click the **service=Fcmdb Config Services** link under the Topaz section.

c In the JMX MBEAN View page, locate the **loadOrReloadCodeBaseForAdaptorId()** operation.

d In the customerID field, enter **1**.

e In the adaptorId field, enter **MyAdapter**. (This is the name you gave to the adapter.)

f Click **Invoke**.

11 Create a Data Store

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.

a In HP Universal CMDB access the Federated CMDB page (**Settings > Federated CMDB**).

b Create a data source. For details, see "Data Stores Tab" on page 115.

The Data Store dialog box displays all CITs that support federation.

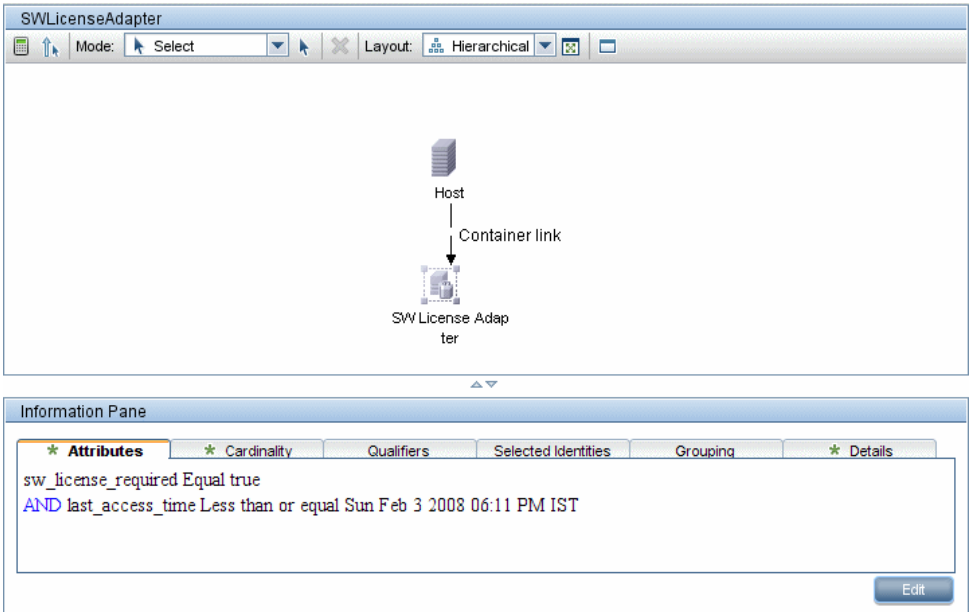
12 Create a View

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

a In HP Universal CMDB access the View Manager (**Admin > Modeling > View Manager**).

b Create a view. For details, see "Create a Template Based View" on page 162.

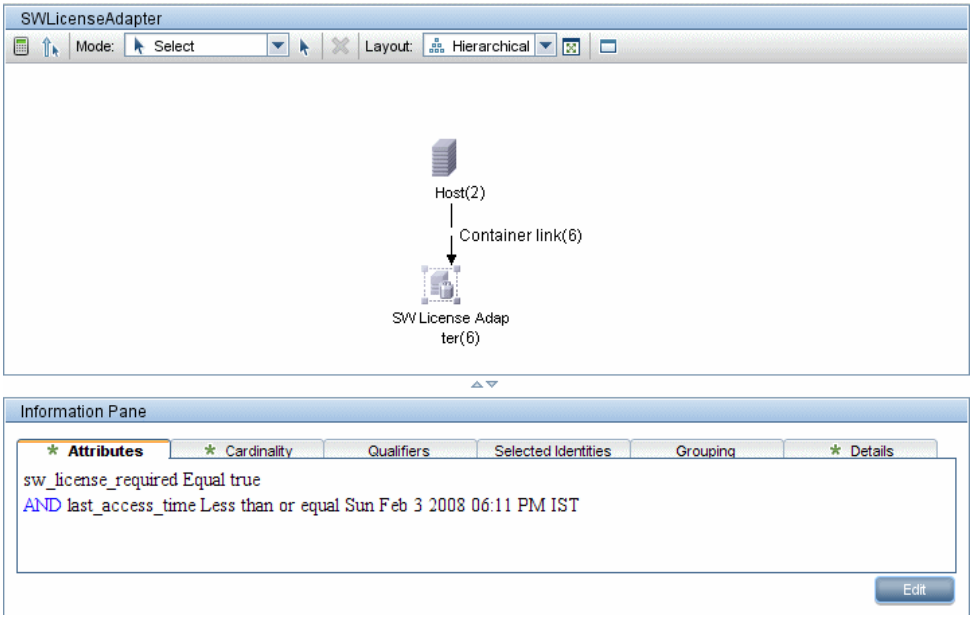
c You can add conditions to the TQL, for example, the last access time is greater than six months:



13 Calculate the Results

In this step you check the results.

a From the View Manager calculate results: click the Calculate TQL result count button.



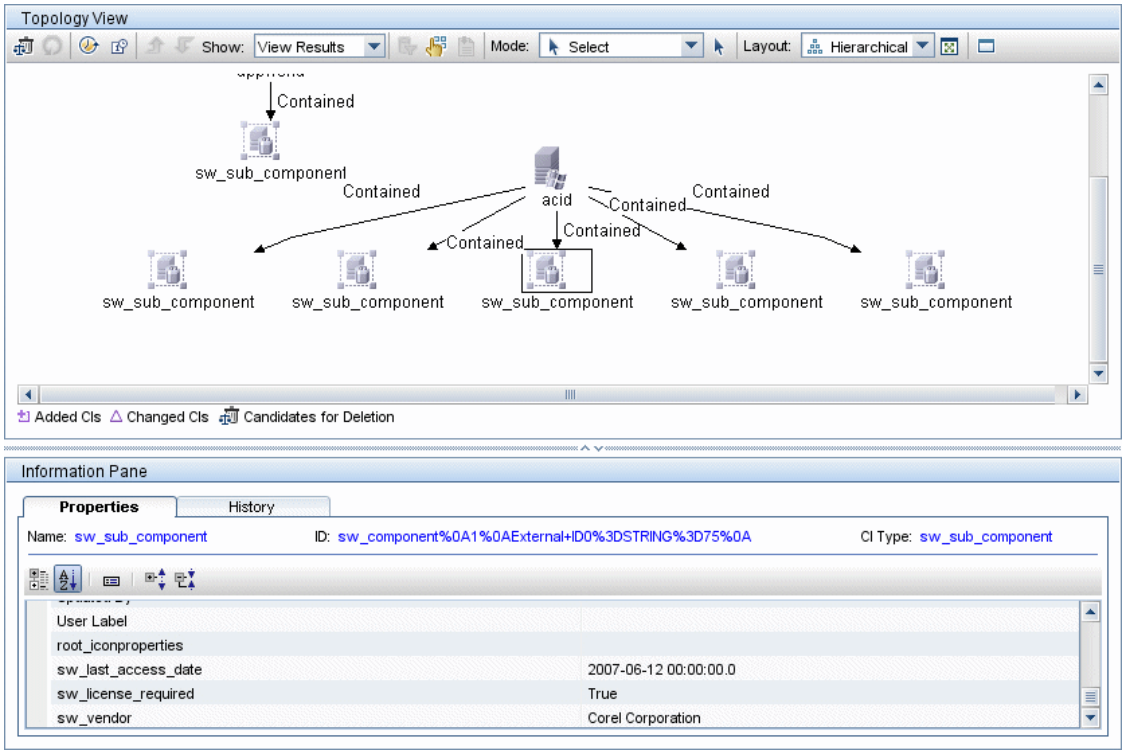
b Click the **Preview** button to view the CIs in the CIT.

14 View 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.

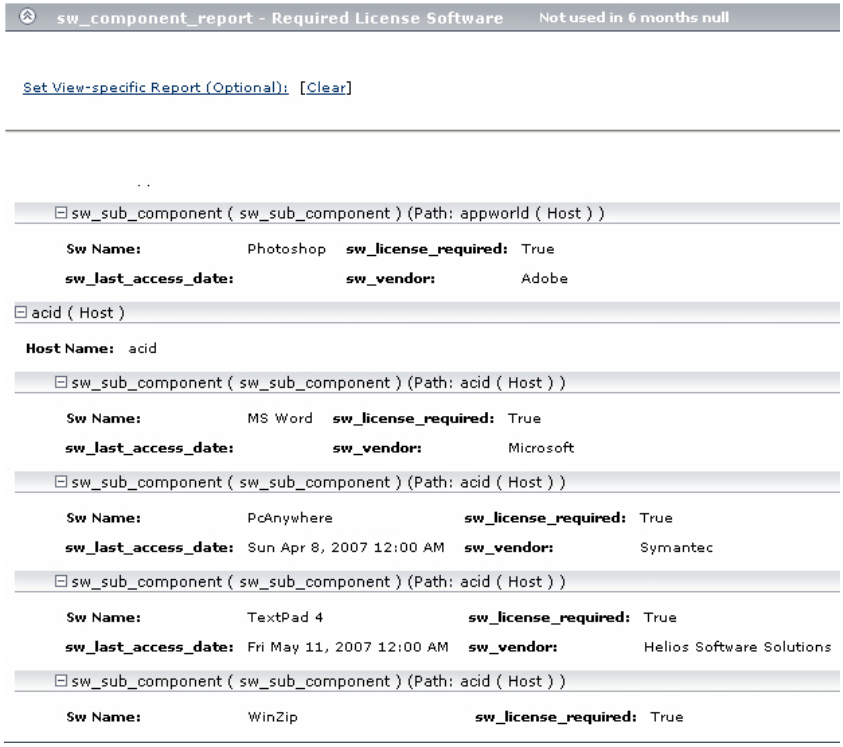
a In HP Universal CMDB access the Topology View (**Application** > **Topology View**).

b The Properties tab displays the results of the federation:



15 View Reports

In this step you view Topology reports. For details, see Chapter 28, “Topology Report.”



16 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 203.

The Federated Database Configuration Files

The files discussed in this section are located under the **<HP Universal CMDB root directory>\UCMDBServer\j2f\fcmdb\CodeBase\GenericDBAdapter\META-INF** folder.

This section includes the following topics:

General Configuration

- **adapter.conf.** The adapter configuration file. For details, see "The adapter.conf File" on page 169.

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 177.
- **reconciliation_rules.txt**. Contains the reconciliation rules. For details, see "The reconciliation_rules.txt File" on page 181.
- **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 182.

Hibernate Configuration

- **persistence.xml.** Used to override out of the box Hibernate configurations. For details, see "The persistence.xml File" on page 183.

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 169.

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 less 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 under the **<HP Universal CMDB root directory>\UCMDBServer\j2f\fcmdb\CodeBase\GenericDBAdapter\META-INF** folder.

This section includes the following topics:

- "Example of the XSD File" on page 170
- "The Template" on page 173

➤ "Limitations" on page 176

Example of the XSD File

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 3 U (http://www.xmlspy.com) by Nimrod (Mercury Interactive) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:element name="generic-DB-adapter-config">
      <xs:complexType>
           <xs:sequence>
                <xs:element ref="CMDB-class" maxOccurs="unbounded"/>
                <xs:element ref="class" maxOccurs="unbounded"/>
           </xs:sequence>
      </xs:complexType>
 </xs:element>
 <xs:element name="class">
      <xs:complexType>
           <xs:sequence>
                <xs:element ref="foreign-primary-key" maxOccurs="unbounded"/>
                <xs:element ref="primary-key" maxOccurs="unbounded"/>
                <xs:element ref="attribute" minOccurs="0" maxOccurs="unbounded"/>
           </xs:sequence>
           <xs:attribute name="CMDB-class-name" type="xs:string" use="required"/>
           <xs:attribute name="default-table-name" type="xs:string" use="required"/>
           <xs:attribute name="connected-CMDB-class-name" type="xs:string" use="required"/>
           <xs:attribute name="link-class-name" type="xs:string" use="required"/>
      </xs:complexType>
 </xs:element>
 <xs:element name="reconciliation-by-single-node">
      <xs:complexType>
           <xs:choice>
                <xs:element ref="attribute"/>
                <xs:element name="or">
                     <xs:complexType>
                          <xs:choice minOccurs="2" maxOccurs="unbounded">
                               <xs:element name="and">
                                    <xs:complexType>
                                         <xs:sequence>
                                              <xs:element ref="attribute" minOccurs="2"
maxOccurs="unbounded"/>
                                         </xs:sequence>
                                    </xs:complexType>
                               </xs:element>
                               <xs:element ref="attribute"/>
                          </xs:choice>
```

```
                        <xs:attribute name="is-ordered" type="xs:boolean" use="optional"
default="false"/>
                    </xs:complexType>
                </xs:element>
                <xs:element name="and">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="attribute" minOccurs="2" maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="primary-key">
        <xs:complexType>
            <xs:attribute name="column-name" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="attribute">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="attribute-type"/>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="attribute-type">
        <xs:attribute name="table-name" type="xs:string" use="optional"/>
        <xs:attribute name="column-name" type="xs:string" use="required"/>
        <xs:attribute name="CMDB-attribute-name" type="xs:string" use="required"/>
        <xs:attribute name="from-CMDB-converter" type="xs:string" use="optional"/>
        <xs:attribute name="to-CMDB-converter" type="xs:string" use="optional"/>
        <xs:attribute name="ignore-case" type="xs:boolean" use="optional" default="false"/>
    </xs:complexType>
    <xs:complexType name="class-type"/>
    <xs:element name="or">
        <xs:complexType>
            <xs:choice minOccurs="2" maxOccurs="unbounded">
                <xs:element ref="and"/>
                <xs:element ref="attribute"/>
                <xs:element ref="connected-node-attribute"/>
            </xs:choice>
            <xs:attribute name="is-ordered" type="xs:boolean" use="optional" default="false"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="and">
```

```
        <xs:complexType>
            <xs:choice minOccurs="2" maxOccurs="unbounded">
                <xs:element ref="attribute"/>
                <xs:element ref="connected-node-attribute"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="reconciliation-by-two-nodes">
        <xs:complexType>
            <xs:sequence>
                <xs:choice>
                    <xs:element ref="attribute"/>
                    <xs:element ref="connected-node-attribute"/>
                    <xs:element ref="or"/>
                    <xs:element ref="and"/>
                </xs:choice>
            </xs:sequence>
            <xs:attribute name="connected-node-CMDB-class-name" type="xs:string" use="required"/>
            <xs:attribute name="CMDB-link-type" type="xs:string" use="required"/>
            <xs:attribute name="link-direction" use="optional" default="main-to-connected">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="main-to-connected"/>
                        <xs:enumeration value="connected-to-main"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
        </xs:complexType>
    </xs:element>
    <xs:element name="connected-node-attribute" type="attribute-type"/>
    <xs:element name="CMDB-class">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="primary-key" maxOccurs="unbounded"/>
                <xs:choice>
                    <xs:element ref="reconciliation-by-single-node"/>
                    <xs:element ref="reconciliation-by-two-nodes"/>
                </xs:choice>
            </xs:sequence>
            <xs:attribute name="CMDB-class-name" type="xs:string" use="required"/>
            <xs:attribute name="default-table-name" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="foreign-primary-key">
        <xs:complexType>
            <xs:attribute name="CMDB-class-primary-key-column" type="xs:string" use="required"/>
```

```
        <xs:attribute name="column-name" type="xs:string" use="required"/>
    </xs:complexType>
 </xs:element>
</xs:schema>
```

The 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="host" 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 host to ip or from ip to host):

```
      <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-link-
type="contained" 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="host_hostname" column-
name="[column_name]" ignore-case="true"/>
                <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="[column_name]"/>
            </or>
        </reconciliation-by-two-nodes>
 </CMDB-class>
 <class CMDB-class-name="[CMDB_class_name]" default-table-
name="[default_table_name]" connected-CMDB-class-name="host" link-class-
name="container_f">
```

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 host 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>
```

Example of Simple Mapping

```
<?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="host" default-table-name="Device">
      <primary-key column-name="Device_ID"/>
      <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-link-
type="contained">
           <or>
                <attribute CMDB-attribute-name="host_hostname" column-name="Device_Name"/>
                <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="Device_PreferredIPAddress"/>
           </or>
      </reconciliation-by-two-nodes>
 </CMDB-class>
 <class CMDB-class-name="host_card" default-table-name="hwCards" connected-CMDB-class-
name="host" link-class-name="contained">
      <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>
 <class CMDB-class-name="sw_sub_component" default-table-name="SWSubComponent" connected-
CMDB-class-name="host" link-class-name="contained">
```

```
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-column="Device_ID"/>
        <primary-key column-name="Device_ID"/>
        <primary-key column-name="Version_ID"/>
        <attribute CMDB-attribute-name="installed_dir" column-
name="SWSubComponent_InstalledDirectory"/>
        <attribute CMDB-attribute-name="license_required" column-
name="SWSubComponent_LicenceRequired"/>
        <attribute CMDB-attribute-name="last_access_time" column-
name="SWSubComponent_LastAccessTimeStamp"/>
        <attribute CMDB-attribute-name="last_access_time_string" column-
name="SWSubComponent_LastAccessTimeStamp"/>
 </class>
 <class CMDB-class-name="host_scsi_device" default-table-name="hwSCSIDevices" connected-
CMDB-class-name="host" link-class-name="contained">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-column="Device_ID"/>
        <primary-key column-name="Device_ID"/>
        <primary-key column-name="hwSCSIDevices_Seq"/>
        <attribute CMDB-attribute-name="scsi_device_name" column-name="hwSCSIDeviceName"/>
        <attribute CMDB-attribute-name="scsi_device_vendor" column-name="hwSCSIDeviceVendor"/>
        <attribute CMDB-attribute-name="scsi_device_type" column-name="hwSCSIDeviceType"/>
 </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 host > 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 host. You cannot bring tickets that belong to more than one host.
- You cannot connect the same class to different types of CMDB CITs. For example, if you define that ticket is connected to host, 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 <**HP Universal CMDB root directory**>**\UCMDBServer\j2f\fcmdb\CodeBase \GenericDBAdapter\META-INF** folder.

This section includes the following topics:

The 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 **name** is the same as **class** and that **class** includes a **generic_DB_adapter.** prefix.

```
<entity name="host" class="generic_DB_adapter.host">
    <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>host</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. The expression is located in the reconciliation_rules.txt file:

```
<basic name="host_hostname">
    <column updatable="false" insertable="false" name="[column_name]"/>
</basic>
<basic name="ip_ip_address">
    <column updatable="false" insertable="false" name="[column_name]"/>
</basic>
</attributes>
</entity>
```

For a single table inheritance with discriminator, map the extending classes as follows:

```
 <entity name="[cmdb_class_name]" class="generic_DBdb_adapter.nt" name="nt">

      <discriminator-value>nt</discriminator-value>
      <attributes/>
 </entity>
 <entity class="generic_DB_adapter.unix" name="unix">
      <discriminator-value>unix</discriminator-value>
      <attributes/>
 </entity>
 <entity name="[CMDB_class_name]"
class="generic_DB_adapter.[CMDB[cmdb_class_name]">
      <table name="[default_table_name]"/>
      <secondary-table name=""/>
      <attributes>
           <id name="id1">
                <column updatable="false" insertable="false" name="[column_name]"/>
                <generated-value strategy="TABLE"/>
           </id>
           <id name="id2">
                <column updatable="false" insertable="false" name="[column_name]"/>
                <generated-value strategy="TABLE"/>
           </id>
           <id name="id3">
                <column updatable="false" insertable="false" name="[column_name]"/>
                <generated-value strategy="TABLE"/>
           </id>
```

The following example shows a 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 **host** and the **linkType** is **container_f**.

```
 <entity name="host_container_f_[CMDB_class_name]"
class="generic_DB_adapter.host_container_f_[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 **host** entity.

many-to-one. Many relationships can be connected to one host.

join-column. The column that contains **end1** IDs (the target entity IDs).

referenced-column-name. The column name in the target entity (**host**) that contain the IDs that are used in the join column.

```
           <many-to-one target-entity="host" 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.
- If an entity has more than one primary key column, you must add an **id** class as a separate entity. The name takes the form IdClassXPK_cmdbClassName where X is replaced with the number of primary key columns and className is replaced with the entity name with which this id class is associated.
- Entity names must match class names even as regards case.
- 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**.

The reconciliation_rules.txt File

This file is used to configure the reconciliation rules.

Each row in the file represents a rule. For example:

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

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, attribute ip_address in the ip class is written ip.ip_address.

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.

The transformations.txt File

This file contains all the converter definitions.

The format is that each line contains a new definition.

The Template

```
entity[[CMDB_class_name]] attribute[[CMDB_attribute_name]]
to_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcmdb.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.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB**. 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 host name.

from_DB_class. The full, qualified name of a class that implements the **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB** 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 host name.

For details, see "Out of the Box Converters" on page 186.

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:

```
host=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.host" name="host">
      <table name="[table_name]"/>
      …
      <inheritance strategy="SINGLE_TABLE"/>
      <discriminator-value>host</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 **host** CIT.

Note that the **host** CIT is also the parent of **nt** and **unix**.

The replication_config.txt File

This file contains a comma-separated list of CI and relationship types whose property conditions are supported by the replication plugin. For details, see "Plugins" on page 190.

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] 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

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: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.fcmdb.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.fcmdb.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.fcmdb.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.fcmdb.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.fcmdb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytesToStringTransformer.

Plugins

The generic database adapter supports the following plugins:

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

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

Configuration Examples

This section gives examples of configurations.

This section includes the following topics:

Use Case

Use case. A TQL is:

host > (container_f) > host_card

where:

host is the UCMDB entity

host_card is the federated database source entity

container_f is the relationship between them

The example is run against the ED database. ED hosts is stored in the Device table and host_card is stored in the hwCards table. In the following examples, host_card is always mapped in the same manner.

Single Node Reconciliation

In this example the reconciliation is run against the host_hostname property.

Simplified Definition

The multinode is the host 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="host" default-table-name="Device">
      <primary-key column-name="Device_ID"/>
      <reconciliation-by-single-node>
           <or>
                <attribute CMDB-attribute-name="host_hostname" column-name="Device_Name"/>
           </or>
      </reconciliation-by-single-node>
 </CMDB-class>
 <class CMDB-class-name="host_card" default-table-name="hwCards" connected-CMDB-class-
name="host" link-class-name="container_f">
      <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 177.

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.host" name="host">
      <table name="Device"/>
      <attributes>
           <id name="id1">
                <column name="Device_ID" insertable="false" updatable="false"/>
                <generated-value strategy="TABLE"/>
           </id>
           <basic name="host_hostname">
                <column name="Device_Name"/>
           </basic>
      </attributes>
 </entity>
 <entity class="generic_DB_adapter.host_card" name="host_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.host_container_f_host_card"
name="host_container_f_host_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="host">
                <join-column name="Device_ID" insertable="false" updatable="false"/>
           </many-to-one>
           <one-to-one name="end2" target-entity="host_card">
                <join-column name="hwCards_Seq" referenced-column-name="hwCards_Seq"
insertable="false" updatable="false"/>
           </one-to-one>
      </attributes>
 </entity>
</entity-mappings>
```

The reconciliation_rules.txt File

For details, see “The reconciliation_rules.txt File” on page 181.

```
multinode[host] expression[host.host_hostname]
```

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 host_hostname and ip_address properties with different variations.

The reconciliation TQL is **host** > **(contained)** > **ip**.

Simplified Definition

The reconciliation is host_hostname OR 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="host" default-table-name="Device">
      <primary-key column-name="Device_ID"/>
      <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-link-
type="contained">
           <or>
                <attribute CMDB-attribute-name="host_hostname" column-
name="Device_Name"/>
                <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="Device_PreferredIPAddress"/>
           </or>
      </reconciliation-by-two-nodes>
 </CMDB-class>
 <class CMDB-class-name="host_card" default-table-name="hwCards" connected-
CMDB-class-name="host" link-class-name="contained">
      <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 host_hostname AND 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="host" default-table-name="Device">
      <primary-key column-name="Device_ID"/>
      <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-link-
type="contained">
           <and>
                <attribute CMDB-attribute-name="host_hostname" column-
name="Device_Name"/>
                <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="Device_PreferredIPAddress"/>
           </and>
      </reconciliation-by-two-nodes>
 </CMDB-class>
 <class CMDB-class-name="host_card" default-table-name="hwCards" connected-
CMDB-class-name="host" link-class-name="contained">
      <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 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="host" default-table-name="Device">
      <primary-key column-name="Device_ID"/>
      <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-link-
type="contained">
           <or>
                <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="Device_PreferredIPAddress"/>
           </or>
      </reconciliation-by-two-nodes>
 </CMDB-class>
 <class CMDB-class-name="host_card" default-table-name="hwCards" connected-
CMDB-class-name="host" link-class-name="contained">
      <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 can be used for both OR and AND and for ip_address alone.

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.host" name="host">
      <table name="Device"/>
      <attributes>
```

```
            <id name="id1">
                <column name="Device_ID" insertable="false" updatable="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <basic name="host_hostname">
                <column name="Device_Name" insertable="false" updatable="false"/>
            </basic>
            <basic name="ip_ip_address">
                <column name="Device_PreferredIPAddress" insertable="false" updatable="false"/>
            </basic>
        </attributes>
    </entity>
    <entity class="generic_DB_adapter.host_card" name="host_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.host_container_f_host_card"
name="host_container_f_host_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="host">
                <join-column name="Device_ID" insertable="false" updatable="false"/>
            </many-to-one>
            <one-to-one name="end2" target-entity="host_card">
                <join-column name="hwCards_Seq" referenced-column-name="hwCards_Seq"
insertable="false" updatable="false"/>
            </one-to-one>
        </attributes>
```

```
 </entity>
</entity-mappings>
```

The reconciliation_rules.txt File

For more information, see “The reconciliation_rules.txt File” on page 181.

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

```
multinode[host] expression[ip.ip_address AND host.host_hostname] end1_type[host]
end2_type[ip] link_type[contained]
```

```
multinode[host] expression[ip.ip_address] end1_type[host] end2_type[ip]
link_type[contained]
```

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 XMLS definitions:

Simplified Definition

There is more than one primary key tag and for each column there is a tag.

```
<?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="host" default-table-name="Device">
      <primary-key column-name="Device_ID"/>
      <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-link-
type="contained">
            <or>
                  <attribute CMDB-attribute-name="host_hostname" column-
name="Device_Name"/>
                  <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="Device_PreferredIPAddress"/>
            </or>
      </reconciliation-by-two-nodes>
 </CMDB-class>
 <class CMDB-class-name="host_card" default-table-name="hwCards" connected-
CMDB-class-name="host" link-class-name="contained">
      <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>
</generic-DB-adapter-config>
```

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 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 177.

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.host" name="host">
      <table name="Device"/>
      <attributes>
           <id name="id1">
                <column name="Device_ID" insertable="false" updatable="false"/>
                <generated-value strategy="TABLE"/>
           </id>
           <basic name="host_hostname">
                <column name="Device_Name"/>
           </basic>
      </attributes>
 </entity>
 <entity class="generic_DB_adapter.host_card" name="host_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>
           <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.host_contained_host_card" name="host_contained_host_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>
        <many-to-one name="end1" target-entity="host">
            <join-column name="Device_ID" insertable="false" updatable="false"/>
        </many-to-one>
        <one-to-one name="end2" target-entity="host_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 host_hostname 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

```
<?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="host" default-table-name="Device">
      <primary-key column-name="Device_ID"/>
      <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-link-
type="contained">
           <or>
                <attribute CMDB-attribute-name="host_hostname" column-
name="Device_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)"/>
                <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="Device_PreferredIPAddress"/>
           </or>
      </reconciliation-by-two-nodes>
 </CMDB-class>
 <class CMDB-class-name="host_card" default-table-name="hwCards" connected-
CMDB-class-name="host" link-class-name="contained">
      <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

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[host] attribute[host_hostname]
to_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.Generic
EnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcmdb.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

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

Open the following file in a text editor: **<HP Universal CMDB root directory>**\j2f\conf\core\Tools\log4j\fcmdb\fcmdb.gdba.properties.

The default log level is **ERROR**:

```
#loglevel can be any of DEBUG INFO WARN ERROR FATAL
loglevel=ERROR
```

➤ To increase the log level for all log files, change **loglevel=ERROR** to **loglevel=DEBUG** or **loglevel=INFO**.

➤ To change the log level for a specific file, change the specific **log4j** category line accordingly. For example, to change the log level of fcmdb.gdba.dal.sql.log to **INFO**, change

```
log4j.category.fcmdb.gdba.dal.SQL=${loglevel},fcmdb.gdba.dal.SQL.appender
```

to:

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

Log Locations

The log files are located in the following directory: **<HP Universal CMDB root directory>\j2f\log\fcmdb**.

➤ **Fcmdb.gdba.log**

The adapter lifecycle log. Gives details about when the adapter started or stopped, and which CITs are supported by this adapter.

Consult for initiation errors (adapter load/unload).

➤ **fcmdb.log**

Consult for exceptions.

➤ **cmdb.log**

Consult for exceptions.

➤ **Fcmdb.gdba.mapping.engine.log**

The mapping engine log. Gives details about the reconciliation TQL 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.hibrnate.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 134.

External References

For details on the JavaBeans 3.0 specification, see http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.

Troubleshooting and Limitations

Note the following limitations:

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 host 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 Host 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.

7

The Federation Framework SDK

This chapter provides information on the Federation Framework functionality, which uses an API to retrieve information from federated sources.

This chapter includes:

Concepts

Tasks

Reference

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."

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 HP Universal 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 includes the following topics:

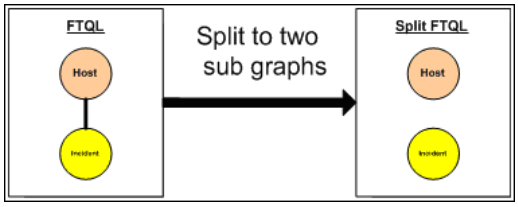
Federation on the Fly

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

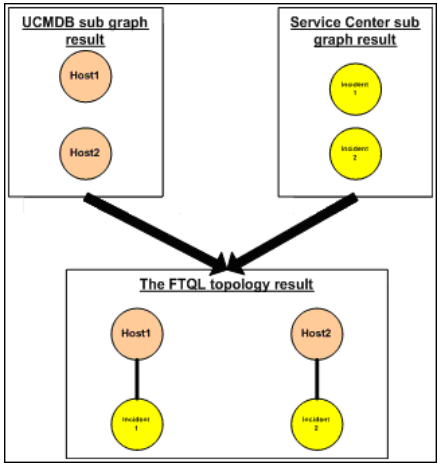
A federated TQL query uses adapters that represent external data stores, to create appropriate external relationships between different external data store CIs and HP Universal CMDB CIs.

Example of Federation on the Fly Flow

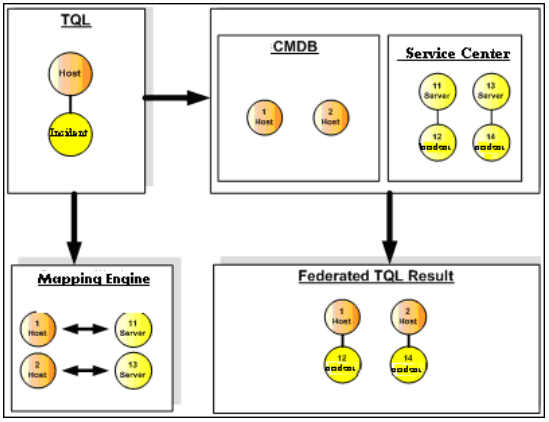
1 The Federation Framework splits a Federated TQL (FTQL) into several subgraphs, where all nodes in a subgraph refer to the same data store. Each subgraph is connected to the other subgraphs by a virtual relationship (but itself contains no virtual relationships).



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 HP Universal CMDB, 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. Retrieves the topology result with signatures from the source data store.
2. Compares the new results with the previous results.
3. Retrieves a full layout (that is, all CI properties) of CIs and relationships, for changed results only.
4. Updates 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 the Federated CMDB that represents external data (data that is not saved in HP Universal CMDB). 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 includes the following topics:

- "Adapter Lifecycle" on page 211
- "Adapter assist Methods" on page 212

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

Reconciliation data. The rule for matching between CIs of the specified type that are received from UCMDB and 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 UCMDB 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 matching 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 some external CI and is used by Mapping Engine to connect between the external CIs to the UCMDB CIs.

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

Mapping engine. A component that identifies relations between CIs from different data stores that have virtual relationship among them. The identification is performed by reconciling between UCMDB Reconciliation objects and external CIs 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 Universal CMDB. The type of this end will be reconciliation type. For the calculation of the two sub graphs, a virtual relationship can start from any end node.

FTQL Adapter

The FTQL adapter will be requested to bring two kinds of data from external data store: external CIs data and reconciliation objects that belong to external CIs.

External CIs data. The external data that does not exist in UCMDB. It is the target data of the external data store for the UCMDB.

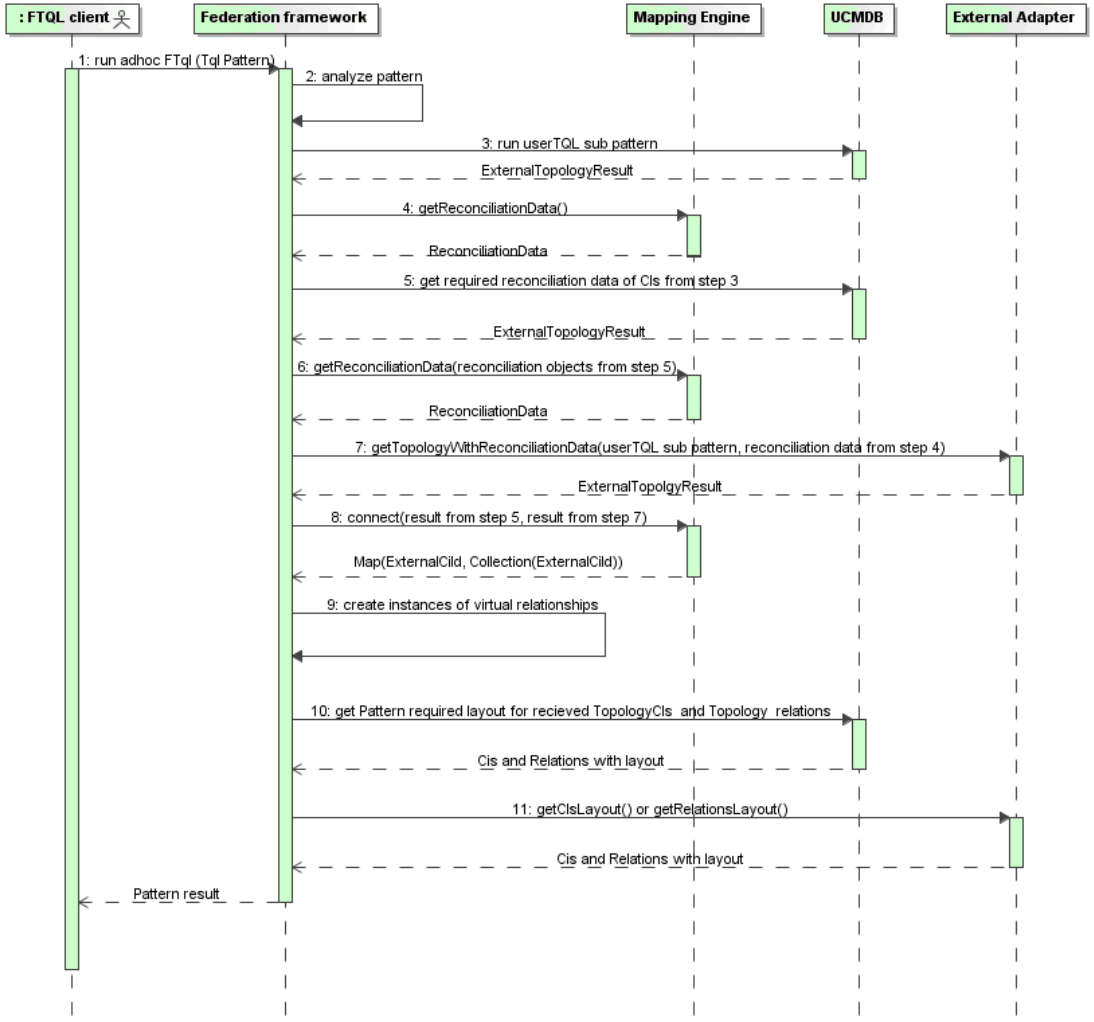
Reconciliation objects data. The auxiliary data that is used by the federation framework to connect between UCMDB CIs and external data. Each reconciliation object should refer to an External CI. The type of the reconciliation objects is the type (or subtype) of one of the virtual relationship end which data is retrieved from UCMDB. Reconciliation objects should fit to reconciliation data the adapter receive. It can be one of three types: IdReconciliationObject, PropertyReconciliationObject, and TopologyReconciliationObject.

Flow Diagrams

The following two diagrams illustrate the Federation Framework, the Universal CMDB, the adapter, and the Mapping Engine. The FTQL in the example diagrams has only one virtual relationship, so that only the Universal CMDB and one external data store are involved in the FTQL. In the first diagram the calculation begins at the Universal CMDB and in the second diagram at the external adapter. Each step in the diagram has reference 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, the Universal CMDB, the adapter, and the Mapping Engine. The FTQL in the example diagram has only one virtual relationship, so that only the Universal CMDB and one external data store are involved in the FTQL.



The numbers in this image are explained below:

1 The Federation Framework receives a call for a FTQL calculation.

2 The Federation Framework analyzes the pattern, finds the virtual relationship, and divides the original TQL into two sub-patterns–one for the Universal CMDB and one for the external data store.

3 The Federation Framework requests the topology of the sub-TQL from the Universal CMDB.

4 After receiving the topology results, the Federation Framework calls the appropriate Mapping Engine for the current virtual relationship and requests reconciliation data. The reconciliationObject parameter is empty at this stage, that is, no condition should be added to reconciliation data in this call. The returned reconciliation data defines what data is needed to match between the reconciliation CIs from UCMDB and external data store. The reconciliation data can be one of three following types:

- **IdReconciliationData.** Reconciliation is done by ID.
- **PropertyReconciliationData.** Reconciliation is done by properties of one CI.
- **TopologyReconciliationData.** Reconciliation is done by topology (for example, to reconcile host CIs, the IP address of **IP** is required too).

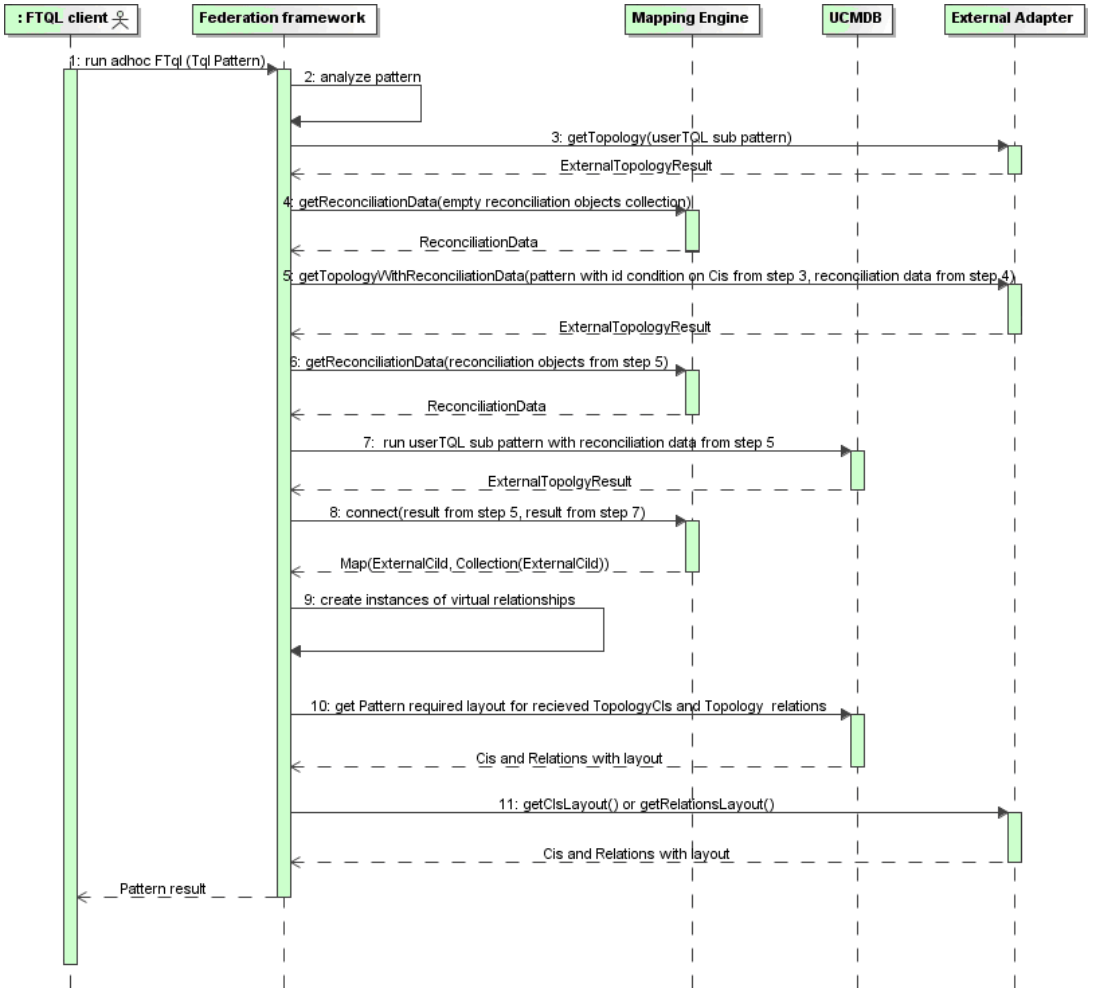
5 The Federation Framework requests reconciliation data for the CIs of the virtual relationship ends that were received in step 3 from the Universal CMDB.

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 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 as a parameter the reconciliation data from step 6.

8. The Federation Framework calls the Mapping Engine to connect between the received results. The firstResult parameter is the external topology result received from UCMDB in step 5 and the secondResult 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 (the Universal CMDB in our 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



1 The Federation Framework receives a call for a FTQL calculation.

2 The Federation Framework analyzes the pattern, finds the virtual relationship, and divides the original TQL into two sub-patterns - one for the Universal CMDB and one for the external data store.

3. The Federation Framework requests the topology of the sub-TQL from the External Adapter. The returned ExternalTopologyResult is not supposed to contain any reconciliation object, since the reconciliation data is not part of the request.

4. After receiving the topology results, the Federation Framework calls the appropriate Mapping Engine with the current virtual relationship and requests reconciliation data. The reconciliationObjects parameter is empty at this state - it means that no condition should be added to reconciliation data in this call. The returned reconciliation data defines what data is needed to match between the reconciliation CIs from UCMDB and external data store. The reconciliation data can be one of three following types:

   - IdReconciliationData. Reconciliation is done by ID
   - PropertyReconciliationData. Reconciliation is done by properties of one CI.
   - TopologyReconciliationData. Reconciliation is done by 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 getTopologyWithReconciliationData() method in the External Adapter, where the requested topology is one node topology with CIs received in step 3 as ID condition and reconciliation data from the step 4.

6. The Federation Framework calls the Mapping Engine to get the reconciliation data. In this state (in 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 with reconciliation data from the step 6 from the UCMDB.

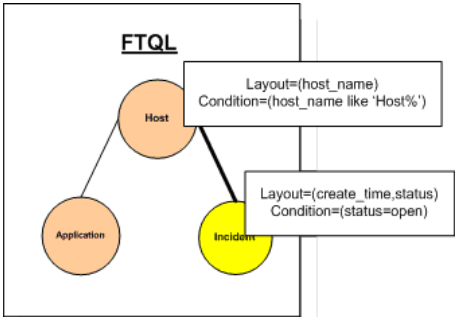
8 The Federation Framework calls the Mapping Engine to connect between the received results. The firstResult parameter is the external topology result received from Extermal Adapter at step 5 and the secondResult parameter is the external topology result received from the UCMDB at step 7. The Mapping Engine returns a map where External CI ID from the first data store (the external data store in our case) is mapped to the External CI IDs from the second (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

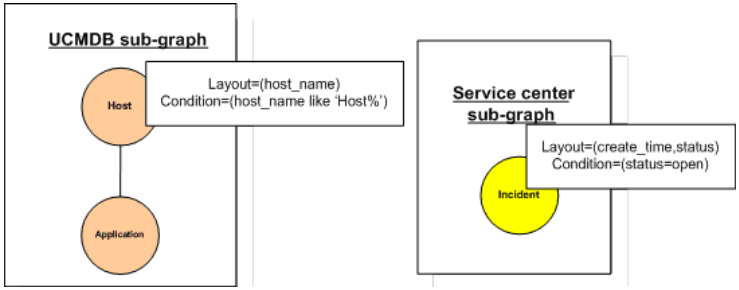
This example explains how to view all open incidents on specific hosts. The ServiceCenter data store is the external data store. The host instances are stored in the Universal CMDB, and the incident instances are stored in ServiceCenter. We assume that to connect the incident instances to the appropriate 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 the Universal CMDB.



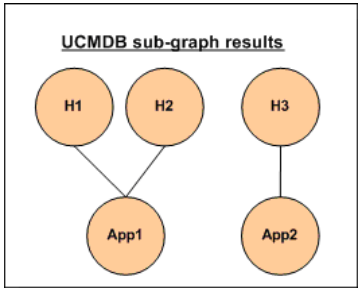
Note: For attribute federation, the adapter's getTopology method is called. The reconciliation data is adapted in the user TQL (in this case, the CI element).

---

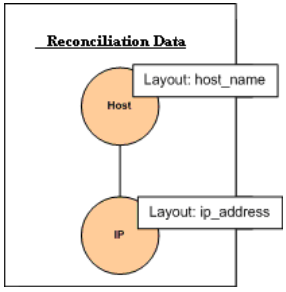
1 After analyzing the pattern, the Federation Framework recognizes the virtual relationship between Host and Incident and splits the FTQL into two subgraphs:



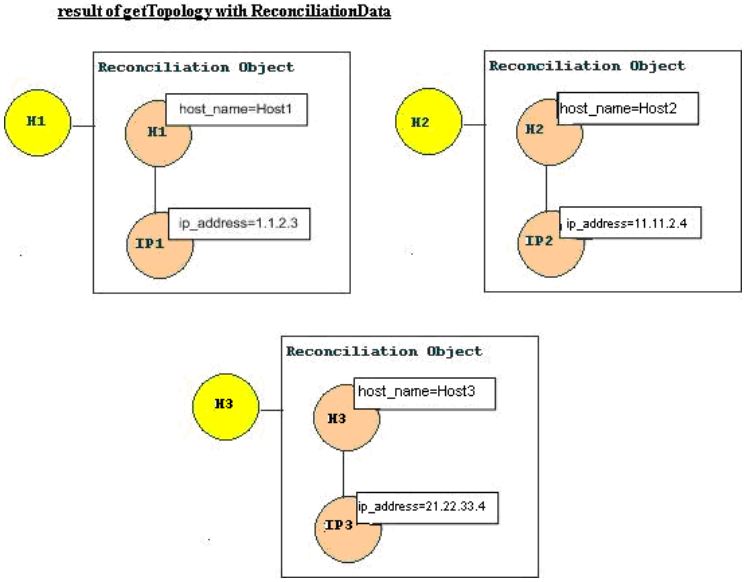
2 The Federation Framework runs the the Universal CMDB subgraph to request the topology, and receives the following results:



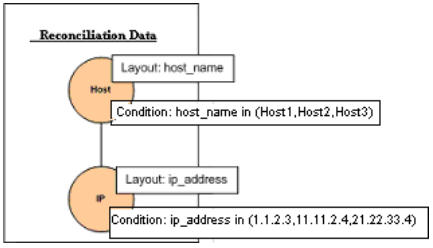
3 The Federation Framework requests from the appropriate Mapping Engine the reconciliation data for the first data store (HP Universal CMDB) that contains the information to connect between received data from two data stores. The reconciliation data in this case is:



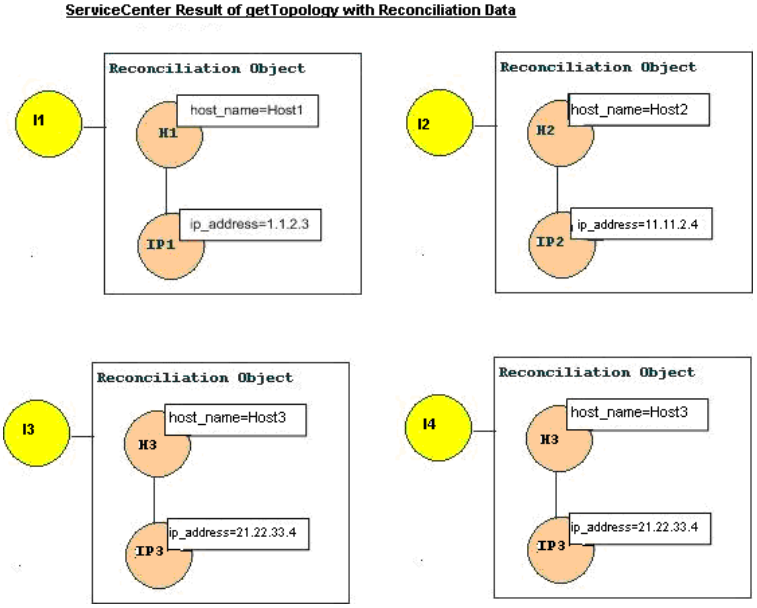
4 The Federation Framework creates one node topology query with Host node and ID conditions on it from the previous result (host_id in H1, H2, H3), and runs this query with required reconciliation data on HP Universal CMDB. The result includes Host CIs that are relevant to the ID condition and the appropriate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



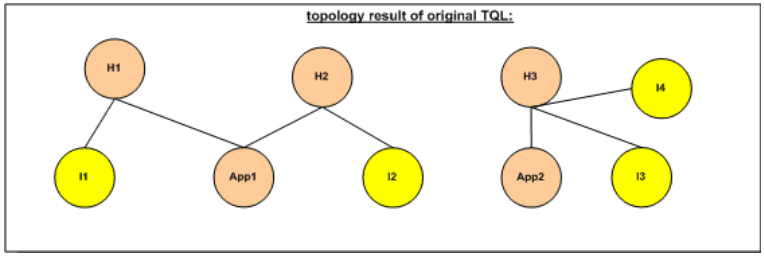
5 The reconciliation data for ServiceCenter should contain a condition for host_name and ip_address that is derived from the reconciliation objects received from HP Universal CMDB:



6 The Federation Framework runs the ServiceCenter subgraph with the reconciliation data to request the topology and appropriate reconciliation objects, and receives the following results:



7 The result after connection in Mapping Engine and creating virtual relationships is:



8 The Federation Framework requests the original TQL layout for received instances from HP Universal CMDB and ServiceCenter.

Federation Framework Flow for Replication

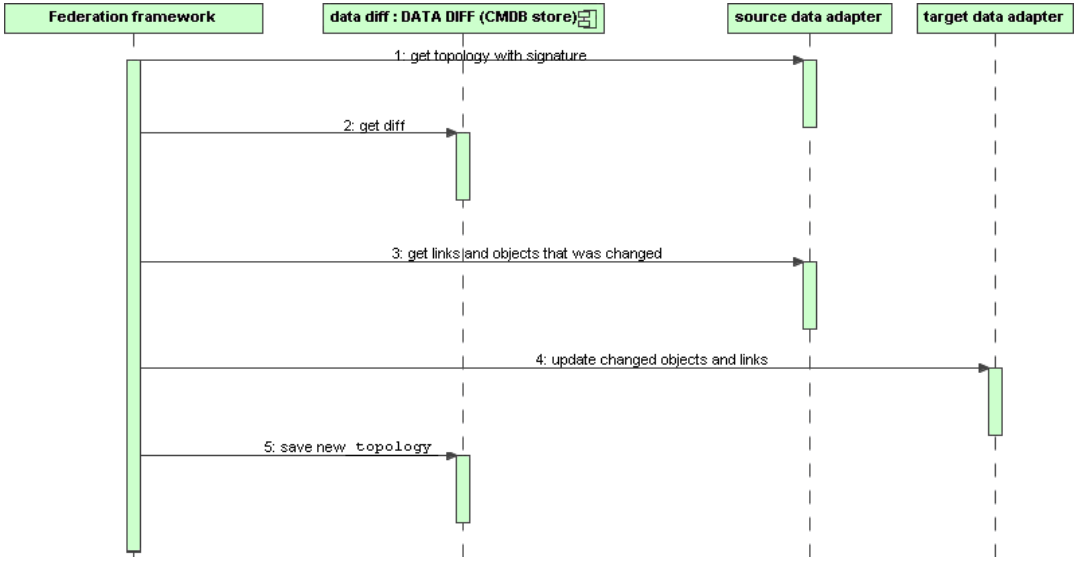
This section includes the following topics:

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. "CI signature" helps detect whether a CI has changed without retrieving and comparing all CI properties. Both the CI and "CI signature" are provided by the appropriate adapter. The adapter is responsible to change CI signature when 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 it represents with the received data.
5. After the update, the Federation Framework saves the last query result.

The HP Release Control Federation Adapter

The HP Release Control Federation adapter supports the retrieval of data from HP Release Control. Every request to HP Release Control to calculate a federated query is made through this adapter. The adapter supports the **Planned Change** CI type. You use **service desk** links to create the query.

The following use cases describe how the adapter can be employed:

- A user needs to display **planned changes to any CI** within a specific time frame.
- A user needs to display **planned changes to specified system CIs**.

  In this case, HP Universal CMDB retrieves changes that directly change system CIs and does not retrieve changes that indirectly affect system CIs.
- A user needs to display **planned changes to a specified business CI**.

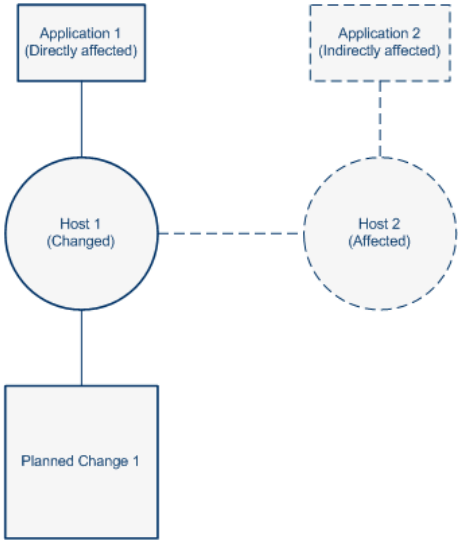
  In this case, HP Universal CMDB retrieves changes that directly affect business CIs and does not retrieve changes that indirectly affect business CIs.

In all of the above cases, HP Universal CMDB retrieves parent changes and independent tasks. HP Universal CMDB does not retrieve tasks that are included in a parent request.

Example

The following example illustrates some of the use cases. Assume there is one planned change in HP Release Control, **Planned Change 1.**

- **Planned Change 1 is** performed on **Host 1**.
- **Application 1** runs on **Host 1** and is therefore directly affected by the change.
- **Host 2** is connected to **Host 1** and may be affected by **Planned Change 1** but no actual change is made to **Host 2**.
- **Application 2** runs on **Host 2** and may be indirectly affected by the change.



If the user runs a query to retrieve planned changes to **Host 1** or **Application 1,** HP Universal CMDB displays **Planned Change 1**, because this change directly affects those CIs.

If the user runs a query to retrieve planned changes to **Host 2** or **Application 2,** HP Universal CMDB does not display any changes because there are no changes that directly affect those CIs.

This section includes the following topics:

Configuring the Federation Adapter

This section describes how to configure the HP Release Control Federation adapter to work with HP Universal CMDB.

To configure the adapter:

1. Deploy the following file in the Package Manager:
   - **<Release Control installation directory>\conf\uCmdb -<version_number>-extensions\federation\rc_federation.zip**

     For details, see Chapter 14, "Package Manager."
2. Copy the **<Release Control installation directory>\conf\uCmdb -<version-number>-extensions\federation\CcmChangeAdapter** directory to the following location:

   <UCMDB installation directory>\UCMDBServer\j2f\fcmdb\CodeBase
3. Reload the **CcmChangeAdapter** in the HP Universal CMDB JMX console.
4. In HP Universal CMDB go to **Settings > Federated CMDB** and configure the **CcmChangeAdapter**. Enter the following required details:
   - **Name.** The logical name of the adapter.
   - **Host.** The URL of HP Release Control.
   - **User.** The user name of an administrator user in HP Release Control.
   - **Password.** The password of the administrator user specified above.

Retrieving Planned Change Attributes

HP Universal CMDB contains a list of the **planned change** attributes in the selected in the **CI Type Manager Attributes** tab. HP Universal CMDB retrieves **planned change** attributes from HP Release Control using the following rule:

HP Universal CMDB converts all underscores (_) in the attribute names and converts them to hyphens (-) and searches through the HP Release Control fields for matching fields. The list of fields in HP Release Control is located in the Administrator module **Fields** tab.

In addition, specific attribute properties are mapped to specific fields in the **convertfields.properties** file, located in the **CcmChangeAdapter** directory. You can map additional attributes to fields HP Release Control in this file.

Adding Custom Fields to the Federation Adapter

This section explains how to add custom fields to the Federation adapter.

To add new custom fields to the Federation adapter:

1. In HP Release Control, add the relevant fields in the Administrator module **Fields** tab. For more information about adding custom fields, refer to the *HP Release Control Installation and Configuration Guide*.
2. In HP Universal CMDB, locate the **planned change** CI type and add the new attribute names.
   - ➤ Use the same name for the attribute as you used for the custom field that you created in HP Release Control. However, if you used a hyphen (-) in the field name, substitute the hyphen for an underscore (_) in the name of the attribute.
   - ➤ To use an attribute name that is different from the custom field name, you can map the attribute name to a specific field name in the **convertfields.properties** file, located in the **CcmChangeAdapter** directory.

Adapter Interfaces

This section includes the following topics:

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.

PatternTopology 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.
- **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.

Add an Adapter for a New External Data Store

This task explains how to define an adapter to support a new external data source.

This task includes the following steps:

- "Model Supported Adapter Classes for CIs and Relationships in the CMBD Class Model" on page 231
- "Define Valid Relationships for Virtual Relationships" on page 231

1 Model Supported Adapter Classes for CIs and Relationships in the CMBD Class Model

As an adapter developer, you should:

- have knowledge of the hierarchy of the HP Universal CMDB CI types to understand how external CITs are related to the HP Universal CMDB CITs
- model the external CITs in the CMDB class model
- add the definitions for new CI types and their relationships
- define valid relationships in the CMDB class model for the valid relationships between adapter inner classes. (The CITs can be placed at any level of the CMDB 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 CMDB class model, see "CI Type Manager" on page 225.

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 HP Universal CMDB 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 241.
- **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

Example of an 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 Implement the Mapping Engine

If your adapter supports federated query, you should support a Mapping Engine implementation for each virtual relationship. The implementation class should implement the Mapping Engine interface.

6 Add Jars Required for Implementation to the Class Path

To implement your classes, add the following jars to your class path:

- federation_api.jar

7 Deploy the Adapter

a Deploy the adapter package. For general details on deploying a package, see "Deploy a Package" on page 466.

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 the 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 **<HP Universal CMDB root directory>\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 **<HP Universal CMDB root directory>\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" on page 466.

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 **<HP Universal CMDB root directory>\UCMDBServer\j2f\fcmdb \CodeBase\<adapter id>** folder.
- Call the next JMX method with the appropriate customer ID and adapter ID:
  FCmdb Config Services > **loadOrReloadCodeBaseForAdapterId**.

Example of Adding a New Adapter

This example illustrates how to add an adapter for ServiceCenter.

1 The relevant ServiceCenter entities are Incident, Problem, and RFC. For each entity, create the CIT definition. Decide where to place these definitions in the HP Universal CMDB 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 Determine to which CITs in HP Universal CMDB 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.

a 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>
```

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-attributes-
supported="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-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>
                     <topology>
                          <one-node-topolgy/>
                     </topology>
                </<support-federated-query>
           </adapter-capabilities>
                <fields-to-connect>
                     <field>host</field>
                     <field>port</field>
                     <field>user</field>
                </fields-to-connect>
</adapter-config>
```

9 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 the reconciliation is done by topology.

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. 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 pattern 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.

8

The HP ServiceCenter/Service Manager Adapter

This chapter provides information on the HP ServiceCenter/Service Manager Adapter, version 1.0. The Adapter is compatible with HP Universal CMDB, version 7.0 or later, HP ServiceCenter, version 6.2, and HP Service Manager, version 7.0 (following changes to the WSDL configuration).

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

This chapter includes:

Concepts

Tasks

Adapter Usage

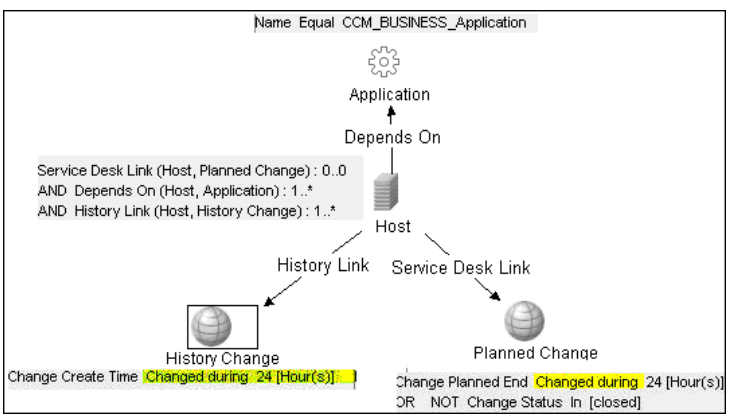
The ServiceCenter/Service Manager Adapter supports the retrieval of data from HP ServiceCenter and HP Service Manager. This adapter connects to, and receives data from, ServiceCenter/Service Manager using the Web Service API. Every request to ServiceCenter/Service Manager to calculate a federated query is made through this adapter.

The Adapter 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 249). Note that there are no internal relationships between Adapter-supported CITs.

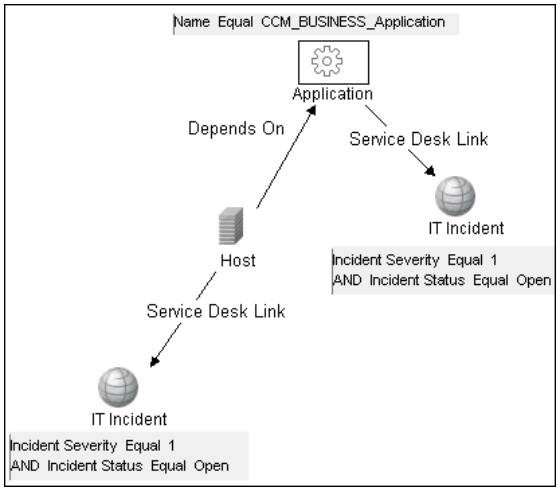
The modeling of the supported CITs and virtual relationships is supplied with the Adapter. You can add attributes to a CIT (for details, see "Add an Attribute to the ServiceCenter/Service Manager CIT" on page 262).

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 Adapter configuration file **serviceDeskConfiguration.xml** is located in the following directory:

<HP Universal CMDB root directory>\UCMDBServer\j2f\fcmdb\CodeBase\ServiceDeskAdapter.

This file contains three parts:

1 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 246.

2 The second part, defined by the reconciliationClassConfigurations element, contains reconciliation data information for appropriate UCMDB CITs. For details, see "Reconciliation Data Configuration" on page 249.

3 The third part, defined by the globalConnectorConfig element, includes the global configuration for a specific connector implementation. For details, see "Global Configuration" on page 253.

Important: The adapter is delivered without a default configuration file. Before defining a data store, you must prepare the appropriate file, according to the version of ServiceCenter/Service Manager you are working with:

➤ Locate the **...\fcmdb\CodeBase\ServiceDeskAdapter** folder.

This folder contains three configuration files:
serviceDeskConfiguration.xml.6.xx for ServiceCenter version 6.xx
serviceDeskConfiguration.xml.7.0x for Service Manager version 7.0x
serviceDeskConfiguration.xml.7.1x for Service Manager version 7.1x

➤ Delete the suffix of the appropriate configuration file. For example, if you are working with Service Manager 7.0x, locate the **serviceDeskConfiguration.xml.7.0x** file and delete **.7.0x**, so that the new name of the file is **serviceDeskConfiguration.xml**.

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 UCMDB 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 UCMDB 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>RetrieveIncidentKeysListRequest</request_name>
           <response_name>RetrieveIncidentKeysListResponse</response_name>
</keys_action_info>
```

```
<properties_action_info>
        <request_name>RetrieveIncidentListRequest</request_name>
        <response_name>RetrieveIncidentListResponse</response_name>
      </properties_action_info>
    </class_configuration> ]]>
      </classConnectorConfiguration>
    </ucmdbClassConfiguration>
```

Adding an Attribute to a CIT

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

1. In **serviceDeskConfiguration.xml**, add an attributeMapping element to the appropriate ucmdbClassConfiguration element.
2. Verify that ServiceCenter/Service Manager externalizes this attribute in its Web Service API.
3. Save **serviceDeskConfiguration.xml**.
4. Send a call to the JMX to reload the adapter: **FCmdb Config Services** > **loadOrReloadCodeBaseForAdapterId**, using the appropriate customer ID and the ServiceDeskAdapter adapter ID.

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.fcmdb.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.fcmdb.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>SerialNumber</reconciliationById>
```

In this example, the ServiceDesk field SerialNumber 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="host_hostname"
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="host_hostname"
serviceDeskAttributeName="NetworkName"
converterClassName="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.converter.PropertyValueConverterToUpperCase"/>
             <reconciliationAttribute ucmdbClassName=”ip”
ucmdbAttributeName="ip_address" serviceDeskAttributeName="NetworkAddress" />

</AND>
  </reconciliationData>
```

In this example, the reconciliation rule follows this format: host.host_hostname= NetworkName and ip.ip_address= 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="host_dnsname"
serviceDeskAttributeName="NetworkDNSName" priority=”2” />
<AND priority=”1” >
          <reconciliationAttribute ucmdbAttributeName="host_hostname"
serviceDeskAttributeName="NetworkName"
converterClassName="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.converter.PropertyValueConverterToUpperCase"/>
            <reconciliationAttribute ucmdbClassName=”ip”
ucmdbAttributeName="ip_address" serviceDeskAttributeName="NetworkAddress" />

</AND>
</OR
  </reconciliationData>
```

In this example the reconciliation rule follows this format: (host.host_dnsname= NetworkDNSName OR (host.host_hostname= NetworkName and ip.ip_address= NetworkAddress)). Since the AND element takes a priority attribute of value 1, the (host.host_hostname= NetworkName and ip.ip_address= 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.

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.conver
ter.PropertyValueConverterToUpperCase"/>
    </OR>
</reconciliationData>
```

You should also change the value of the concreteMappingImplementationClass attribute of the reconciliationClassConfiguration element to:

```
="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.mapping.impl.HostIpMappin
gEngine"
```

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 pattern that the Service Manager is working with.

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

   If the date pattern 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 pattern 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.

Deploy the Adapter

This section describes a typical deployment of the adapter.

This section includes the following steps:

1 "Deploy the ServiceDesk Adapter" on page 255

   a "Extract the Adapter Implementation Files and Deploy the Package" on page 255

   b "Add a ServiceCenter/Service Manager External Data Source" on page 256

c "Configure HP ServiceCenter 6.2" on page 256 (when connecting to HP ServiceCenter)

d "Configure HP Service Manager 7.0" on page 259 (when connecting to HP Service Manager)

2 "Add an Attribute to the ServiceCenter/Service Manager CIT" on page 262

a "Add an Attribute to the HP Universal CMDB Model" on page 262

b "Export Attributes from HP ServiceCenter by Changing the Configuration" on page 263 (when connecting to HP ServiceCenter)

c "Export Attributes from HP Service Manager by Changing the Configuration" on page 265 (when connecting to HP Service Manager)

d "Modify the Adapter Configuration File" on page 268

e "Load the Changes" on page 269

Deploy the ServiceDesk Adapter

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

This section includes the following steps:

- "Extract the Adapter Implementation Files and Deploy the Package" on page 255
- "Add a ServiceCenter/Service Manager External Data Source" on page 256
- "Configure HP ServiceCenter 6.2" on page 256
- "Configure HP Service Manager 7.0" on page 259

1 Extract the Adapter Implementation Files and Deploy the Package

a Verify the location of the following folder and file:

- ServiceDeskAdapter
- serviceDeskAdapter.zip

b Move the **serviceDeskAdapter.zip** package to the following directory: **<HP Universal CMDB root directory>\UCMDBServer\root\lib\packages**.

c Deploy the serviceDeskAdapter.zip package: Log in to HP Universal CMDB and access the Package Manager (**Admin > Settings** > **Package Manager**). Select the package and click the **Deploy** button.

For details on deploying packages, see "Deploy a Package" on page 470.

d Move the **ServiceDeskAdapter** folder to the following directory:

<HP Universal CMDB root directory>\UCMDBServer\j2f\fcmdb \CodeBase

2 Add a ServiceCenter/Service Manager External Data Source

In this step, you add an external data store.

a In HP Universal CMDB, access the Federated CMDB window: **Admin** > **Settings** > **Federated CMDB**.

b Click the button to add a data store. In the Data Store dialog box that opens, choose the **ServiceDeskAdapter** and fill in the mandatory fields.

For help with this dialog box, see "Data Stores Tab" on page 115.

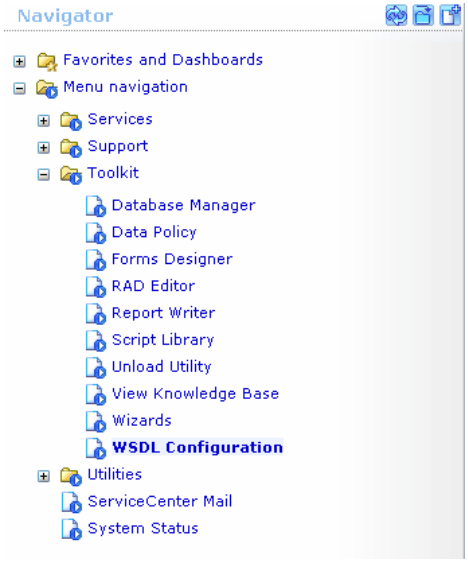
c Continue to "Configure HP ServiceCenter 6.2" on page 256 or "Configure HP Service Manager 7.0" on page 259.

3 Configure HP ServiceCenter 6.2

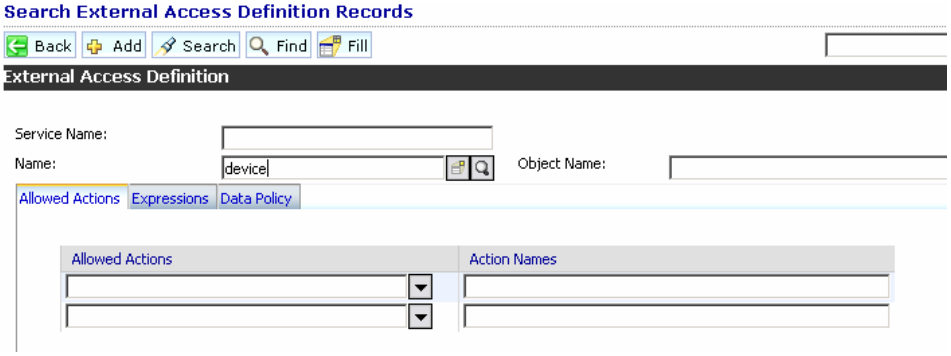
If you are connecting to HP ServiceCenter, perform the following procedure.

a Open HP ServiceCenter, then the ServiceCenter client.

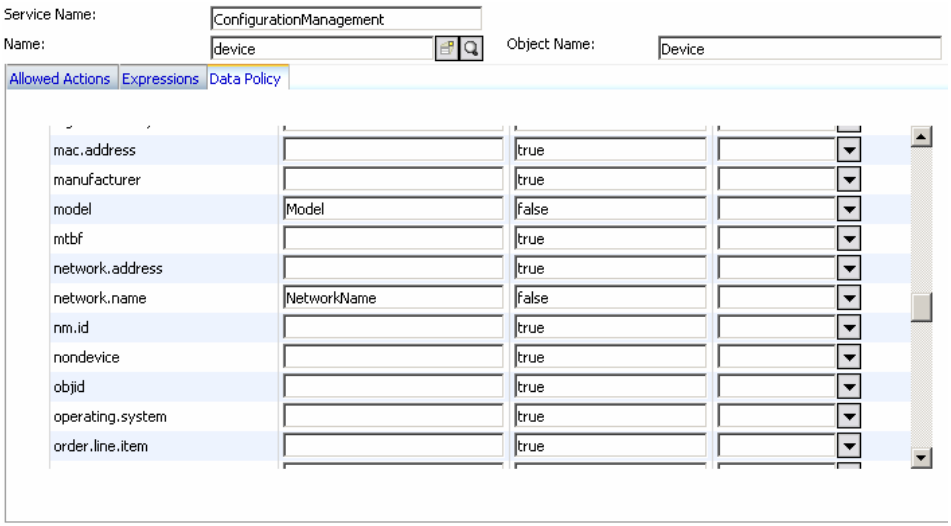
b Display **WSDL Configuration** in the Navigator (**Main Menu** > **Menu navigation** > **Toolkit**):



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



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.

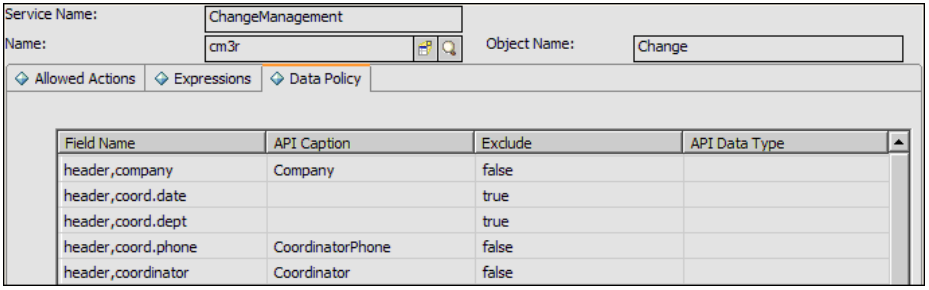


e After saving, click the **Cancel** button.

f In the Object Name field type **Change** and press **Enter**.

g Select the Data Policy tab and ensure that:

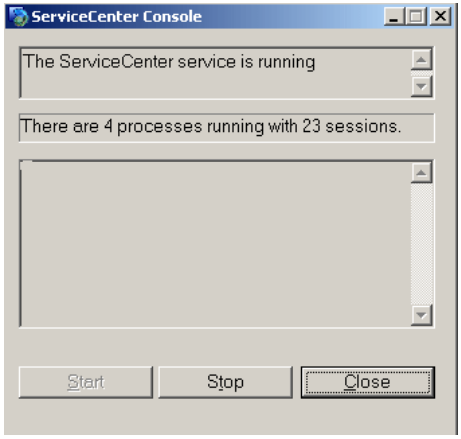
➤ The **header,coordinator** attribute is not empty (its value should be **Coordinator**). Change the value to **false**.



➤ 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**.

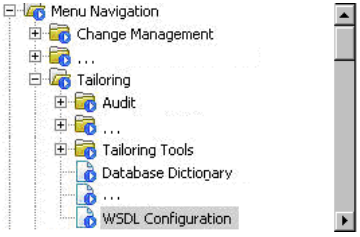
k Continue to "Add an Attribute to the HP Universal CMDB Model" on page 262.

4 Configure HP Service Manager 7.0

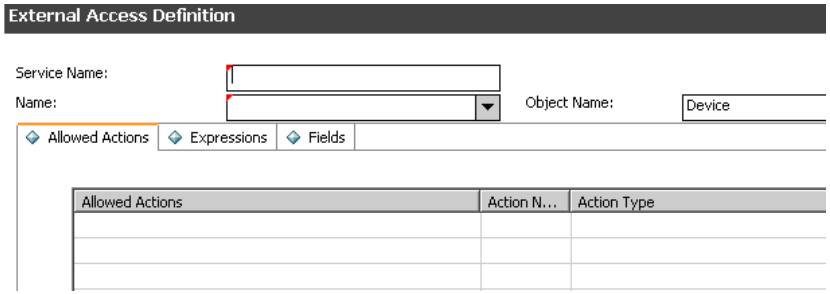
If you are connecting to HP Service Manager, perform the following procedure.

a Open the HP Service Manager client.

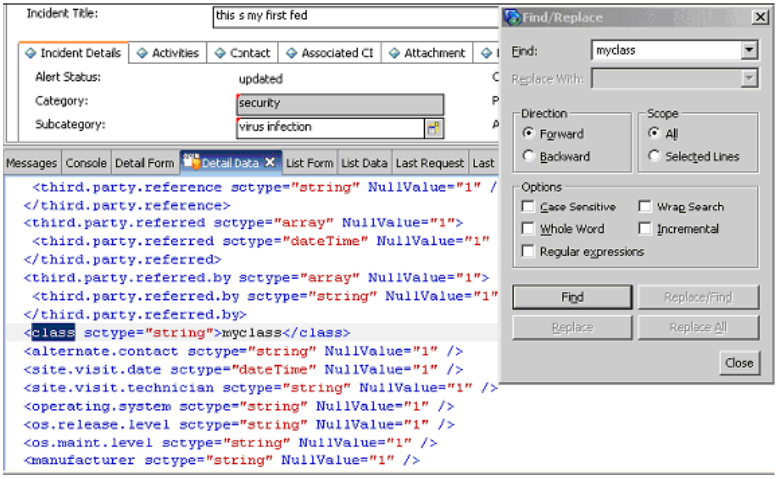
b Display **WSDL Configuration** in the Navigator (**Main Menu** > **Menu navigation** > **Tailoring**):



c In the Object Name field, enter **Device** and press ENTER.



d Select the **Fields** tab and ensure that the network.name attribute appears in the Field list with **NetworkName** as its caption. If this attribute does not appear in the Field list, add it and save your changes.



e After saving, click the **Cancel** button.

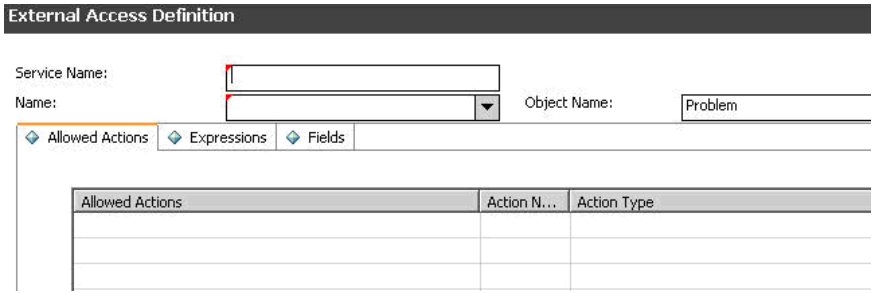
f In the Object Name field type **Change** and press **Enter**.

g Select the Fields tab and ensure that:

- The **header,coordinator** attribute appears in the Field list with **Coordinator** as its caption. If this attribute does not appear in the Field list, add it.
- The **header,orig.operator** attribute appears in the Field list with **OpenedBy** as its caption. If this attribute does not appear in the Field list, add it.

h Save the changes.

i To support Problem federation, access **WSDL Configuration** in the Navigator (**Main Menu** > **Menu navigation** > **Tailoring**) and enter **Problem** in the Object Name field. Press ENTER.



j Click the **Fields** tab, add the following fields, and save your changes:

Field	Caption	Type
id	ID	
brief.description	BriefDescription	
status	Status	
expected.resolution.time	ExpectedResolutionTime	DateTimeType
category	Category	
initial.impact	InitialImpact	
severity	Severity	
priority.code	PriorityCode	
assignment	Assignment	
logical.name	CI	
affected.ci(ci.device.name)	CiDeviceName	

k Continue to “Add an Attribute to the HP Universal CMDB Model” on page 262.

Add an Attribute to the ServiceCenter/Service Manager CIT

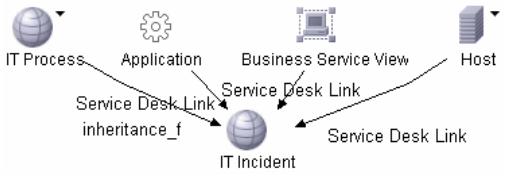
This section explains how to retrieve additional data from ServiceCenter/Service Manager by adding an attribute.

This section includes the following steps:

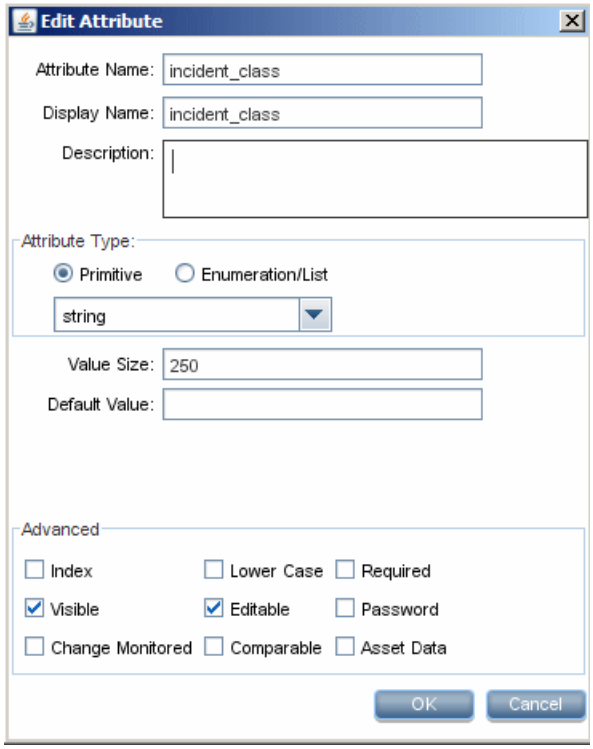
- "Add an Attribute to the HP Universal CMDB Model" on page 262
- "Export Attributes from HP ServiceCenter by Changing the Configuration" on page 263
- "Export Attributes from HP Service Manager by Changing the Configuration" on page 265
- "Modify the Adapter Configuration File" on page 268
- "Load the Changes" on page 269

1 Add an Attribute to the HP Universal CMDB Model

a Add the new attribute to HP Universal CMDB: Edit the Incident CIT: Select **Admin** > **Modeling** > **CI Type Manager**. In View Explorer, select **IT Process** > **IT Incident**.



b Select the Attribute tab and add the new attribute:



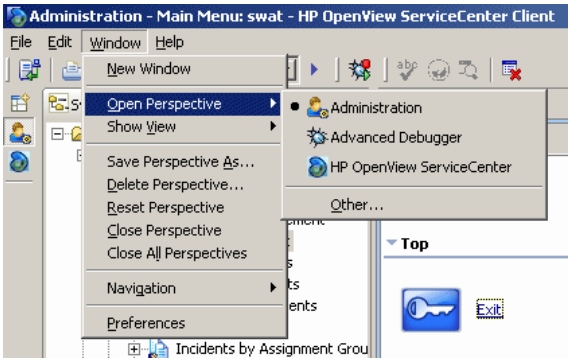
c Continue to "Export Attributes from HP ServiceCenter by Changing the Configuration" on page 263 or "Export Attributes from HP Service Manager by Changing the Configuration" on page 265.

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**:

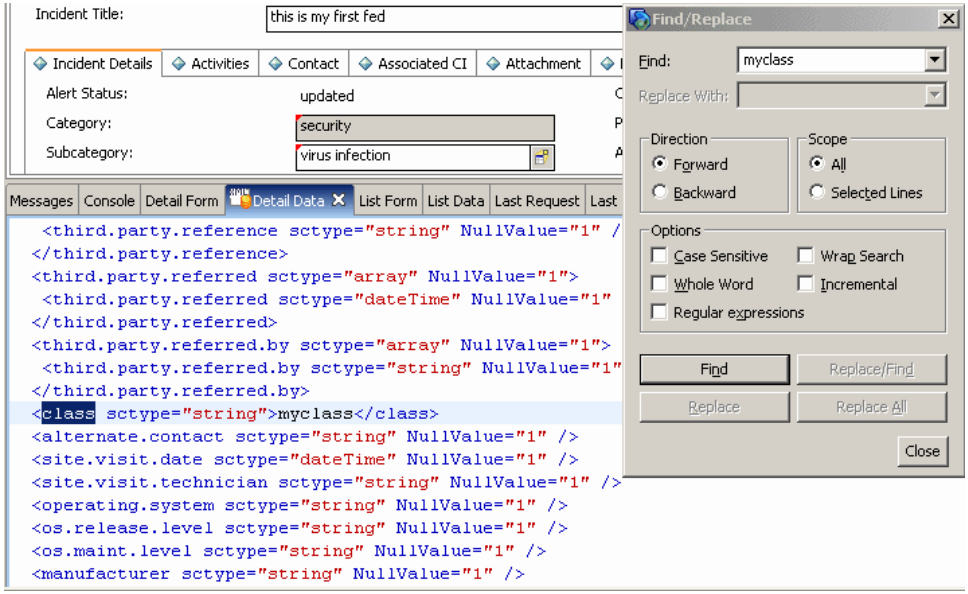


c Select **Incident Management** > **All Open Incidents**, and select one of the incidents you created.

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

---

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.

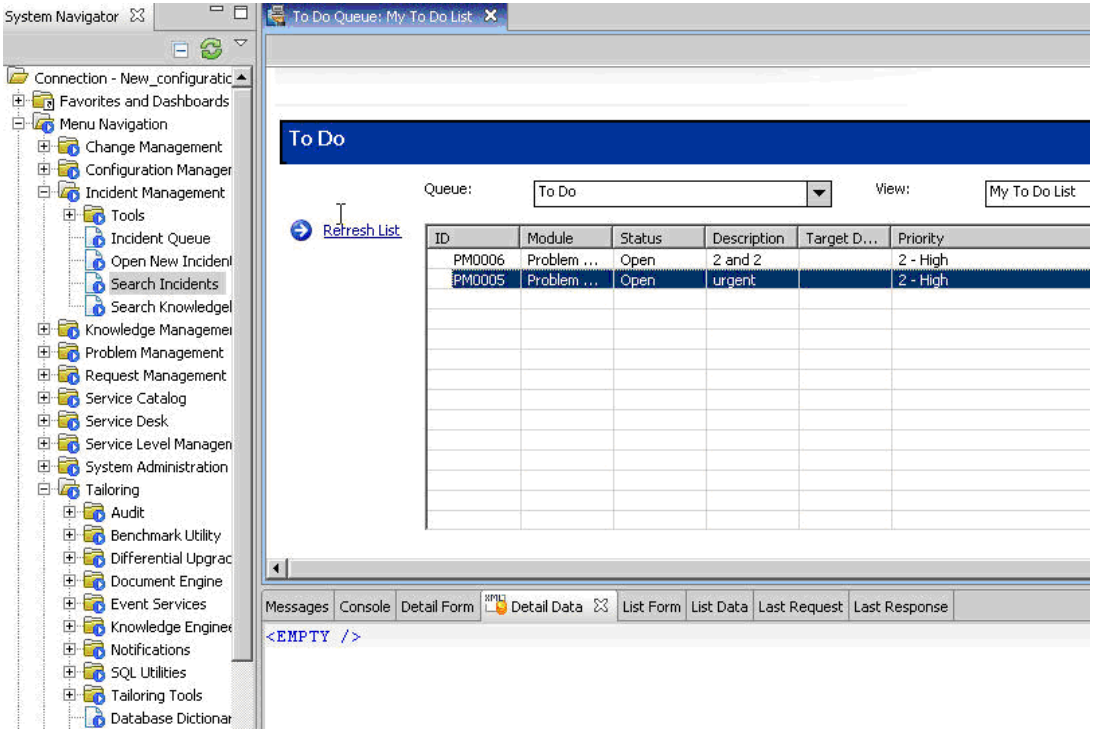
h Click **Stop** and then **Start**.

i Continue to “Modify the Adapter Configuration File” on page 268.

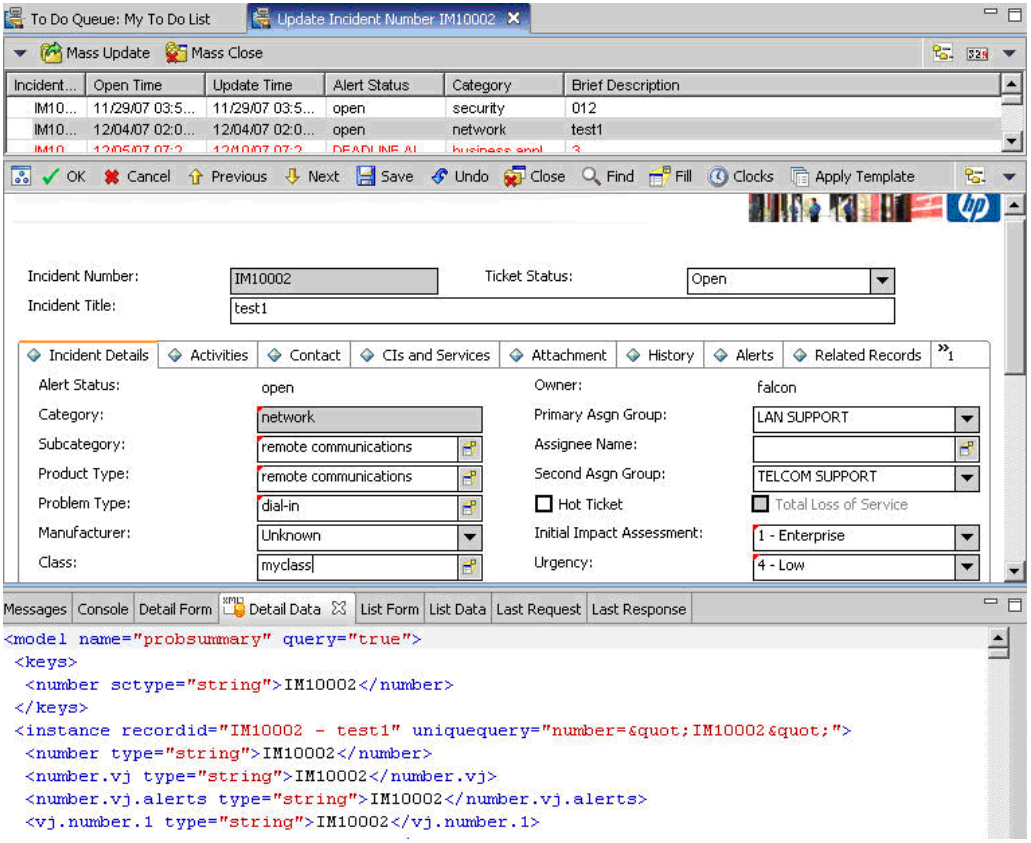
3 Export Attributes from HP Service Manager by Changing the Configuration

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

a In the HP Service Manager client, restore the bottom right pane by clicking the **Restore** button. Click the **Detail Data** tab.

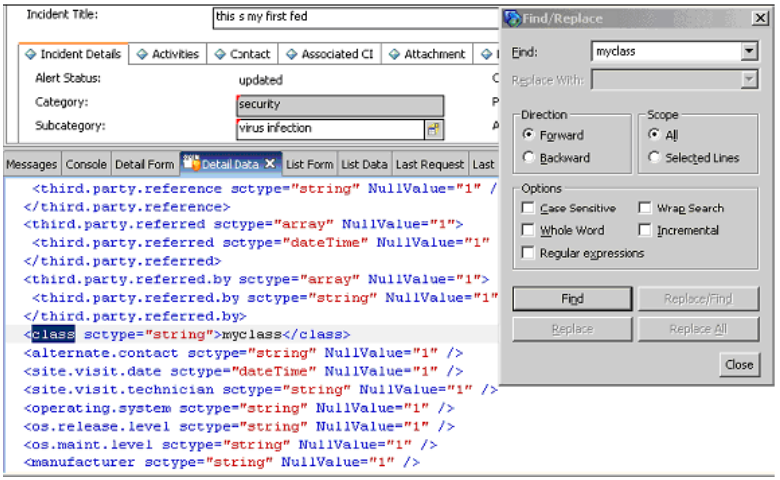


b Open one of the incidents you created: Select **Incident Management** > **Search Incidents**. Click the search button (you can filter the fields to limit the search).



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 **Incident** 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 Restart the HP Service Manager 7.00 Server service.

h Continue to "Modify the Adapter Configuration File" on page 268.

4 Modify the Adapter Configuration File

Perform this procedure for all configurations.

a Edit the **ServiceDeskConfiguration.xml** file in

<HP Universal CMDB root directory>\UCMDBServer\j2f\fcmdb \CodeBase\ServiceDeskAdapter

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 Continue to "Load the Changes" on page 269.

5 Load the Changes

Perform this procedure to load changes.

a Launch the Web browser and enter the following address:

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

where <machine name or IP address> is the machine on which HP Universal CMDB is installed.

You may have to log in with the administrator's user name and password.

b Click the **Topaz** > **service=Fcmdb Config Services** link.

c In the JMX MBEAN View page, locate the following operation: **loadOrReloadCodeBaseForAdapterId()**.

d In the customerID field, enter **1**. In the AdapterId field, enter the name of the Adapter folder (ServiceDeskAdapter). Click **Invoke**.

9

Troubleshooting and Limitations

This chapter includes troubleshooting and limitations for the Federated CMDB functionality.

This chapter includes:

Reference

Federated CMDB Troubleshooting and Limitations

This section includes the following topics:

All Adapters

- When changes are made in View Manager and these changes affect the results of a TQL, federated CIs in the view are not updated. This is because federated TQLs are calculated ad-hoc only and are not updated when a view is recalculated. To update the federated CIs, select the view in View Explorer and click the **Rebuild View** button. (Note that the recalculation may take a long time.) For details, see "Browse Mode" on page 671.
- When configuring a local UCMDB data store, if your RMI adapter also supports federated queries, do not choose CITs in the CITs Supported by Adapter dialog box. (You should add an adapter for the local UCMDB data store only if you want to use it for replication jobs).
- Do not choose a CIT to be supported by an external data store if instances of this CIT exist in the local UCMDB, as this can lead to state inconsistency. For example, if there are instances of the CPU CIT in the local UCMDB, you must not choose the CPU when defining an external data store, even if the selected adapter supports it.
- When configuring a replication job between two CMDBs, verify that the class model is the same in the two CMDBs.

The RMI Adapter

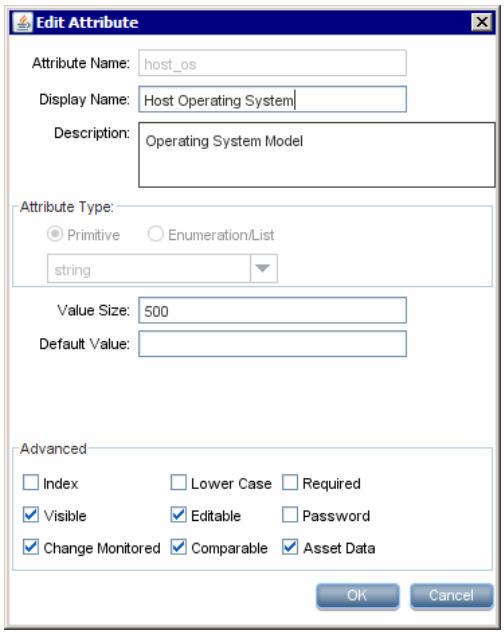
- When using the RMI adapter, the version of the CMDB running on the configured host must be the same as the version of the UCMDB.

The CmdbChanges Adapter

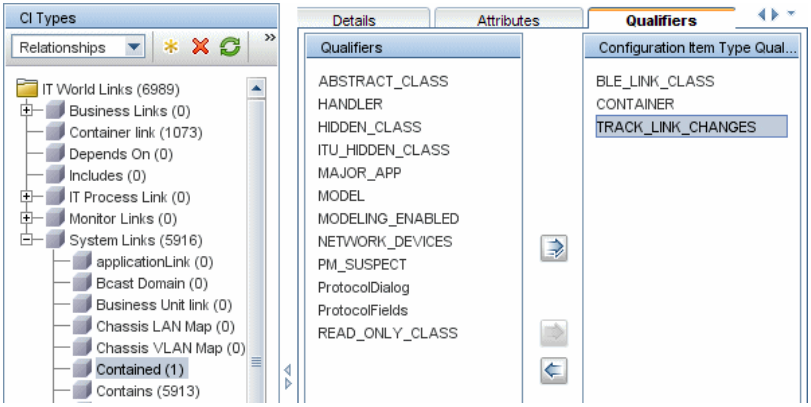
- The adapter does not support any property condition beside the root_class IN … condition on the Root node.
- A query should contain one CI that is labeled **Root**, or one or more relations that are labeled **Root**.

  The root node is the main CI that is synchronized; the other nodes are the contained CIs of the main CI. For example, when synchronizing hosts, the host node is labeled as root and the host resources are not root.
- The TQL graph must not have cycles.
- The TQL must contain only the Root CI and optional CIs that are directly connected to it.

- A query that is used to synchronize relations should have the cardinality 1...* and an OR condition between the relations.
- The adapter does not support compound relations.
- All CI attributes to be replicated must have the **Change Monitored** check box selected (STATIC qualifier) so that they can be written to the History database.



➤ Each relation must include the **TRACK_LINK_CHANGES** qualifier so that it can be written to the History database:



10

Introduction to Reconciliation

This chapter provides information on Reconciliation.

This chapter includes:

Concepts

Reconciliation – Overview

Reconciliation is a process of identifying and matching entities from different data stores (DDM, DDMi, ticketing, and so on). It is designed to avoid duplicate CIs.

The following sections include details of the out-of-the-box reconciliation rules:

Host Reconciliation Rules

- Any host in UCMDB is identified either by an IP address or a strong ID value (such as a MAC address or hardware ID):

  Hosts identified by an IP address are considered incomplete (the Host Is Complete attribute has a false value).

Hosts identified by a strong ID are considered complete. If a host has several MAC addresses, DDM uses the lowest MAC address as the identifier.

- An incomplete host can be reconciled with a complete host by its IP address, or with an incomplete host that has the same ID.
- A complete host can be reconciled with an incomplete host by its IP address, or with a complete host that has the same ID.
- When an incomplete host exists in the CMDB and a complete host is reported, the complete host replaces the incomplete host and the topology (related CIs) that is connected to the incomplete host is moved to the new, complete host.
- When a complete host exists in the CMDB and an incomplete host is reported, the properties of the incomplete host are copied to the complete host that is already in the CMDB.

Note: Properties are updated according to the conflict resolution policy.

Cluster Reconciliation Rules

- A Cluster Server is a complete host.
- If an incomplete host is identified by its IP with two complete hosts, and one of these hosts is a cluster, the incomplete host is identified with the cluster only.
- If a Cluster Server and a complete host are connected to the same VIP (virtual IP), all software elements that include the VIP in the application_ip attribute move from the physical complete host to the Cluster Server.

Software Element Reconciliation Rules

- Software Element CITs are considered to be either of a strong type or a weak type:
  - **Strong Type Software Element CITs.** These CITs are specialized (they are derived from the Software Element CIT) and include more attributes, different identification rules, a display label, and so on.
  - **Weak Type Software Element CITs.** These CITs are CI instances of the Software Element CI Type itself. They include basic configuration attributes only. Also, only one instance of each software component type can exist in the model. That is, you cannot model two different Oracle instances on a single host using a weak type software element.
- A reconciliation mechanism handles the mapping of weak type software elements to their corresponding strong type, once they are discovered. This mechanism enables DDM to detect the existence of software by one method, and report additional details about the software when it is discovered using another method, knowing the two CIs will eventually be merged in the UCMDB.
- The reconciliation mechanism relies on a shared attribute value between the weak type and strong type: the **Name** attribute (data_name). All software elements (strong or weak) are created with a distinguishing name: when a strong software element is discovered on a host containing a weak software element with the same name, the weak software element is merged with the strong one.

Part III

Integrations

11

Embedding UCMDB Applets Using Direct Links

This chapter describes how to embed UCMDB applets into external applications using direct links.

Note: This chapter is relevant to integration developers only.

Using Direct Links to Embed UCMDB Applets

You can directly embed UCMDB applets into external applications using a direct link. A direct link is a URL that you create using the Direct Links wizard. For details, see "Direct Links to an Application Page" in *Solutions and Integrations*.

You can embed UCMDB applets in the following ways:

- Using the URL itself to display CMDB content in an external application. For example, you can create a URL in the Direct Links wizard to display the properties of a certain CI. The login page opens if you are not already logged into HP Universal CMDB.

Note: Using the URL to enable the integration requires opening the link in an IFrame or a new window. This does not allow you to manage the resulting UCMDB applet using JavaScript due to cross-site scripting issues. For example, if you use a URL that displays the properties of a specific CI using the object ID, you cannot use the same URL to display the properties of another CI.

- Using the UCMDB applet tag to embed the UCMDB applet in an external application. This enables integration developers to display a part of the HP Universal CMDB user interface as part of their own application. In addition, this enables interaction with the embedded HP Universal CMDB user interface using JavaScript.

UCMDB Applet Tag Overview

The UCMDB Applet tag creates the environment needed for the UCMDB applet to run (HTML, JavaScript, and server session parameters). The code behind the tag can log into the HP Universal CMDB server in one of the following cases:

- The login parameters are specified and the user is not already logged in
- The **clear session** parameter is true.

Once a user session is established, the tag continues to write the needed environment "in-place", replacing the tag in the JSP with actual HTML / JavaScript. The resulting HTML page includes the required JavaScript code to load the UCMDB applet from the specified server. On error, the specified error string is printed to the page. This string can be customized using the **userErrorMessage** parameter and may include HTML and JavaScript to initiate error handling scenarios.

Example of a UCMDB Applet Tag

```
<ucmdb:ucmdb_applet
serverConnectionString="x.com"
serverType="BAC"
directLinkParameters="cmd=ShowProperties&objectId=2d3b0fec35431ea01625
a4&navigation=false&interfaceVersion=8.0.0"
userName="admin"
userPassword="admin"
clearSessionCookies="false"/>
```

Important: You get the **directLinkParameters** from the URL created in the Direct Links wizard. Delete the part of the parameter that comes before "cmd=", as it is not part of the parameter needed for the UCMDB applet tag. For details on how to create a direct link, see “Direct Links Wizard” on page 439.

The UCMDB applet tag is located at **<HP Universal CMDB root folder>AppServer\webapps\site.war\WEB-INF\tags\CMS\ucmdb_applet.tag**.

You can copy the UCMDB applet tag to an external application server. The UCMDB applet tag requests the login data from the HP Universal CMDB server provided that:

- The external application server can communicate with JSP tags
- The HP Universal CMDB server is accessible to the external application server and to the browser

Direct Link Operation Flow

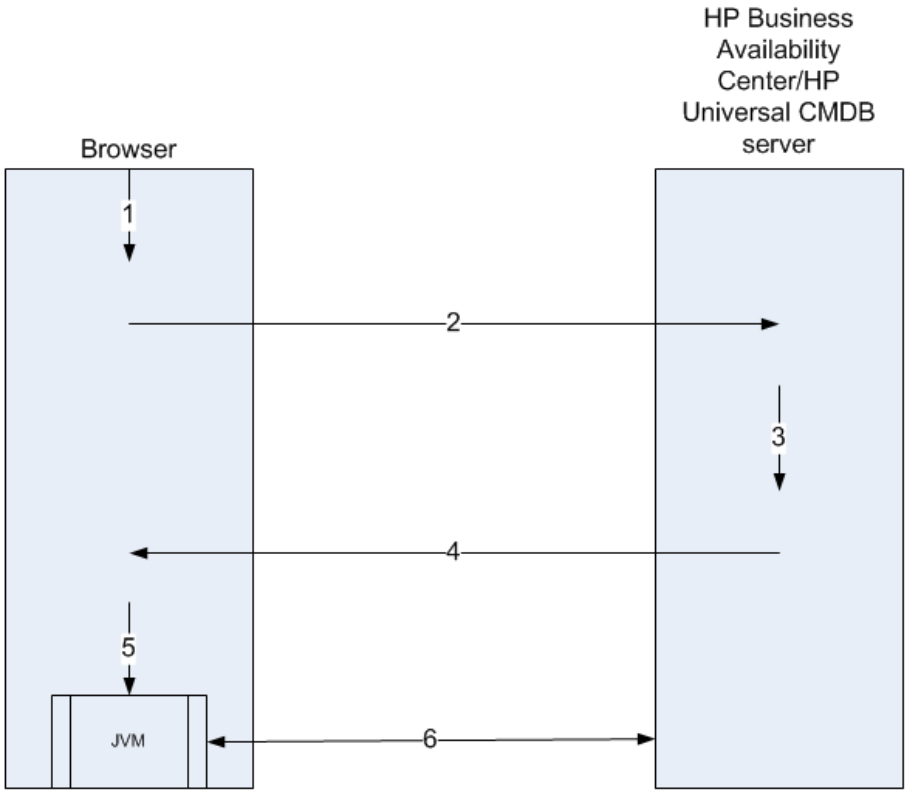
This section describes different scenarios in which direct links are used to embed UCMDB applets into external machines.

This section includes the flow descriptions:

- "Regular Login" on page 284
- "An Embedded UCMDB Applet Using a Direct Link URL" on page 286
- "An Embedded UCMDB Applet Using a UCMDB Applet Tag" on page 288

Regular Login

This flow describes a regular login to the HP Universal CMDB server. It serves as a base reference for the direct link flows.



1 The user enters the direct link URL (into the browser) to log into the HP Universal CMDB server.

2 The browser sends the login request to the HP Universal CMDB server.

3 The server checks the user credentials and creates a user session if required.

4 The server returns the requested first page of HP Universal CMDB.

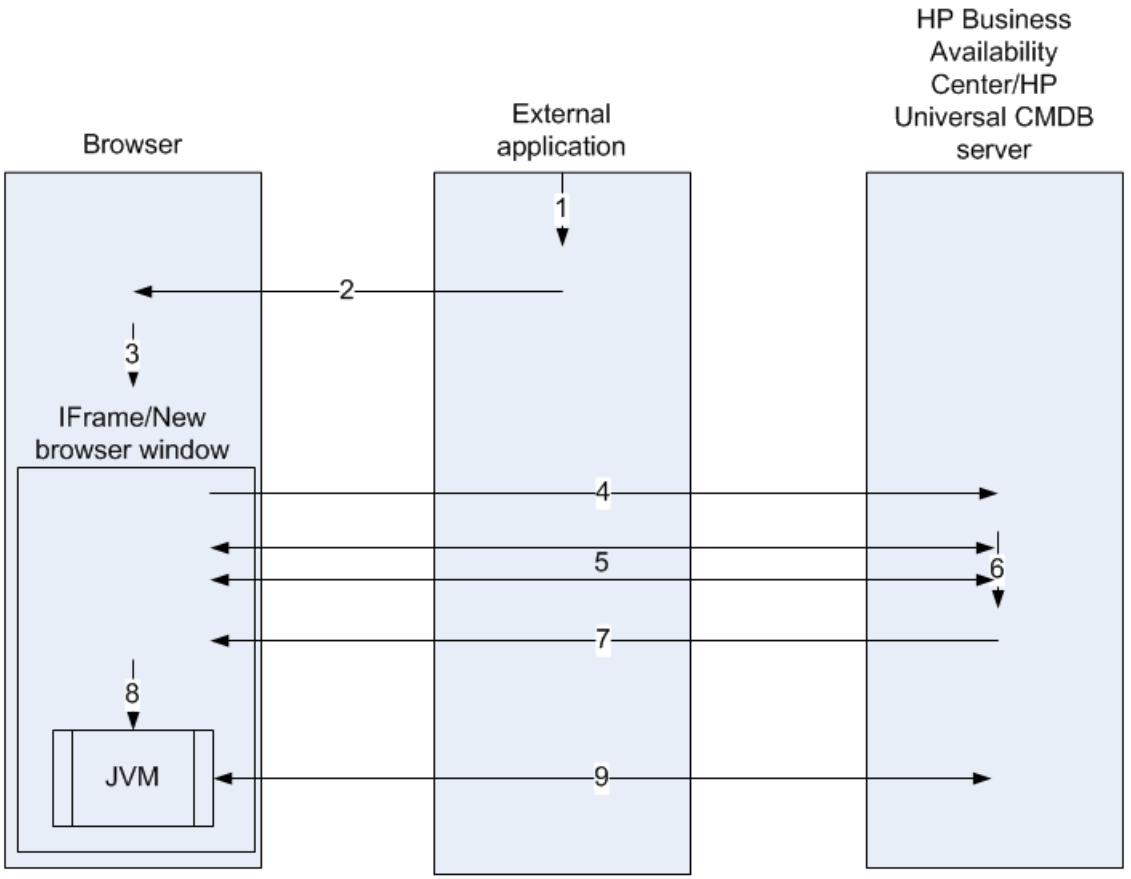
5 The browser loads the Java virtual machine (JVM) with the "code base" location parameter (which instructs the JVM from where to load the UCMDB applet files) as the HP Universal CMDB server.

6 UCMDB applet files (jars) and data are transferred between the HP Universal CMDB server and the JVM loaded in the browser.

At this point that the UCMDB applet is loaded and regards HP Universal CMDB as the server with which it should communicate.

An Embedded UCMDB Applet Using a Direct Link URL

This flow shows how to use the direct link to open a UCMDB applet (opened to a specified context) in a new browser frame.



1 The external application has a page that needs to show the UCMDB applet.

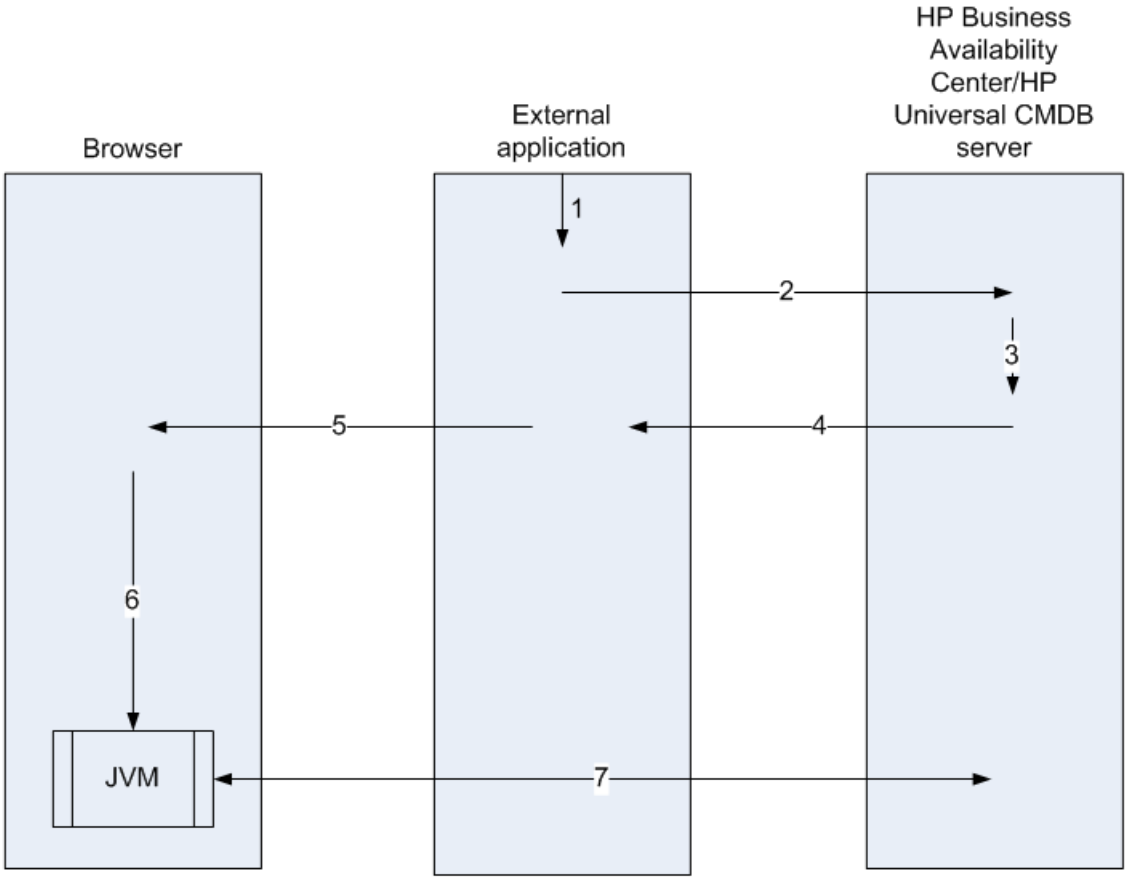
2 The returned page includes a directive to open the direct link in an IFrame or new browser window.

3. The browser opens a new frame (IFrame or new window) with the direct link as the frame URL.
4. The new frame sends the direct link to the HP Universal CMDB server.
5. If the user is not already logged into HP Universal CMDB, HP Universal CMDB displays the login page and waits for the user to enter credentials and click the login button.
6. The user session is created, if required.
7. The HP Universal CMDB server returns a page that includes the directive to load the UCMDB applet.
8. The browser processes the page. When the browser encounters the directive to load the UCMDB applet, it starts the JVM. The browser then passes the needed parameters to the JVM, including the HP Universal CMDB server location as "code base".
9. UCMDB applet files (jars) and data are transferred between the HP Universal CMDB server and the JVM loaded in the browser.

Important: The HP Universal CMDB server must be accessible from the browser machine.

An Embedded UCMDB Applet Using a UCMDB Applet Tag

This flow shows how to use the UCMDB applet tag in conjunction with direct links to embed the UCMDB applet (opened to a specified context) in an external application page.



1 The UCMDB applet tag is included in the processed JSP in the external application.

2 The UCMDB applet tag uses the parameters given to it to create an HTTP/HTTPS connection to the HP Universal CMDB server and requests the applet HTML-let.

3 If the user is not already logged in, the parameters from the UCMDB applet tag are used (see the tag itself for reference information regarding this step). If login fails, the UCMDB applet tag returns an error message, either the preset one or the one specified in the UCMDB applet tag parameters.

4 The HP Universal CMDB server returns the HTML-let (with embedded JavaScript) that loads the UCMDB applet. The external application can now incorporate this HTML-let into the page, which it sends to the browser.

5 The external application sends the complete page which includes the HTML-let to the browser.

6 The browser processes the page. When the browser encounters the HTML-let that loads the UCMDB applet, the JVM is started. The browser then passes the needed parameters to the JVM, including the HP Universal CMDB server location as "code base".

7 UCMDB applet files (jars) and data are transferred between the HP Universal CMDB server and the JVM loaded in the browser.

Important: The HP Universal CMDB server must be accessible from the browser machine as well as the external application machine.

Index

U

W