



Configuration Management System (CMS)

Software Version: Content Pack 28.00 (CP28)

Discovery and Integrations Content Guide - Third Party Integrations

Document Release Date: August 2018
Software Release Date: August 2018



Legal Notices

Disclaimer

Certain versions of software and/or documents ("Material") accessible here may contain branding from Hewlett-Packard Company (now HP Inc.) and Hewlett Packard Enterprise Company. As of September 1, 2017, the Material is now offered by Micro Focus, a separately owned and operated company. Any reference to the HP and Hewlett Packard Enterprise/HPE marks is historical in nature, and the HP and Hewlett Packard Enterprise/HPE marks are the property of their respective owners.

Warranty

The only warranties for products and services of Micro Focus and its affiliates and licensors ("Micro Focus") are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Micro Focus 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. Except as specifically indicated otherwise, a valid license from Micro Focus is required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© 2011 - 2018 Micro Focus or one of its affiliates.

Trademark Notices

MICRO FOCUS and the Micro Focus logo, among others, are trademarks or registered trademarks of Micro Focus (IP) Limited or its subsidiaries in the United Kingdom, United States and other countries. All other marks are the property of their respective owners.

Adobe™ is a trademark of Adobe Systems Incorporated.

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

UNIX® is a registered trademark of The Open Group.

Documentation Updates

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

This site requires that you register for a Software Passport and to sign in. To register for a Software Passport ID, click **Register for Software Passport** on the Micro Focus Support website at <https://softwaresupport.softwaregrp.com>.

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

Support

Visit the Micro Focus Support site at: <https://softwaresupport.softwaregrp.com>.

This website provides contact information and details about the products, services, and support that Micro Focus offers.

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

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up Micro Focus 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 a Software Passport user and to sign in. Many also require a support contract. To register for a Software Passport ID, click **Register for Software Passport** on the Micro Focus Support website at <https://softwaresupport.softwaregrp.com>.

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

Integration Catalog accesses the Micro Focus Integration Catalog website. This site enables you to explore Micro Focus Product Solutions to meet your business needs, includes a full list of Integrations between Micro Focus Products, as well as a listing of ITIL Processes. The URL for this website is <https://softwaresupport.softwaregrp.com/km/KM01702731>.

Contents

Chapter 1: Aperture VISTA Integration	11
Overview	12
Supported Versions	12
Topology	12
How to Use the Aperture VISTA Integration Adapter	13
Aperture VISTA by SQL Adapter	15
Discovery Mechanism	16
Chapter 2: Arxscan Arxview Integration	18
Overview	19
Supported Versions	19
Topology	19
Discovery Mechanism	21
Create an Integration Point between Arxscan Arxview and UCMDB	21
Data Mappings	24
Arxview Integration Adapter	25
Troubleshooting and Limitations – Arxscan Arxview Integration	26
Chapter 3: Atrium Integration	27
Overview	28
Supported Versions	28
How to Work with the Data Push to Atrium Adapter	29
How to Work with the Population from Atrium Adapter	34
Atrium Push Job	38
Data Push into Atrium Adapter	38
Import Data from Atrium Job	40
Population from Atrium Adapter	40
Mapping Files	43
Mapping Files Overview	43
Mapping File Structure	43
Mapping File Elements	44
Troubleshooting and Limitations – Atrium Integration	48
Chapter 4: BMC Remedyforce Integration	49
Overview	50

How to Create Connected Apps in Salesforce	50
How to Populate UCMDB with Data from BMC Remedyforce	51
How to Push Data from UCMDB to BMC Remedyforce	53
Troubleshooting and Limitations – BMC Remedyforce Integration	57
Chapter 5: CA CMDB Integration	58
Overview	59
Supported Versions	59
Integration Mechanism	59
How to Work with the CA CMDB Push Adapter	61
Integration Query	64
Troubleshooting and Limitations – CA CMDB Integration	64
Chapter 6: CiscoWorks LAN Management Solution Integration	66
Overview	67
Supported Versions	67
Topology	67
How to Discover CiscoWorks LMS	68
CiscoWorks LMS Database Ports Job	70
Adapter	70
Trigger Query	70
Parameters	70
Network Devices from CiscoWorks LMS Job	71
Adapter	71
Trigger Query	71
Layer2 Topology from CiscoWorks LMS Job	71
Adapter	71
Trigger Query	72
Discovery Flow	72
CiscoWorks NetDevices Adapter	73
CiscoWorks Layer 2 Adapter	74
Discovery Mechanism	77
Troubleshooting and Limitations – CiscoWorks LAN Management Solution Integration	79
Chapter 7: CyberArk Integration	80
Chapter 8: EMC Control Center (ECC) Integration	81
Overview	82
Supported Versions	82

Topology	82
How to Run the ECC/UCMDB Integration Job	83
ECC Integration Job	85
Views	89
Impact Analysis Rules	94
Reports	96
Chapter 9: HPE OneView Pull Integration	100
Overview	100
Discovery Mechanism	100
Supported Versions	100
Create an Integration Point between HPE OneView and UCMDB	101
Data Mappings	103
Input CITs	104
Used Scripts	104
Discovered CITs	104
Global Configuration File	105
Topology Map	105
How to Integrate with OneView 1.x	106
Troubleshooting and Limitations – OneView Pull Integration	108
Chapter 10: HPE Systems Insight Manager (SIM) Integration	109
Overview	110
Supported Versions	110
SIM Integration Mechanism	110
How to Discover HPE SIM Data Center Infrastructure	113
SIM WebService Ports Job	116
SIM Integration by WebServices Job	117
Instance Views	118
Troubleshooting and Limitations – SIM Integration	120
Chapter 11: IDS Scheer ARIS Integration	121
Overview	122
Supported Versions	122
Topology	122
How to Run the ARIS Integration Job	123
Import CIs from ARIS Job	129
Chapter 12: Importing Data from External Sources	131

Overview	132
Comma Separated Value (CSV) Files	132
CSV Files with Column Titles in First Row	133
Databases	133
Properties Files	133
How to Import CSV Data from an External Source – Scenario	135
How to Convert Strings to Numbers	140
Custom Converters	141
The External_source_import Package	141
Import from CSV File Job	143
Import from Database Job	146
Import from Properties File Job	153
External Source Mapping Files	155
Troubleshooting and Limitations – Importing Data from External Sources	155
Chapter 13: Import from Excel Workbook Discovery	157
Overview	158
Supported Versions	158
Topology	158
How to Import Data from Excel Workbook	158
How to Set Up an Import File in Excel	161
Import from Excel Workbook Job	169
Troubleshooting and Limitations – Import From Excel Workbook Discovery	173
Chapter 14: Microsoft IAMUI Integration	175
Overview	176
Supported Versions	176
How to Push Data from UCMDB to IAMUI	177
Sample Integration Push Query	180
IAMUI Push Adapter	181
Troubleshooting - Microsoft IAMUI Integration	183
Chapter 15: Microsoft SCCM/SMS Integration	185
Overview	186
Supported Versions	186
SMS Adapter	186
How to Populate the CMDB with Data from SCCM/SMS	188
How to Federate Data with SCCM/SMS	190

How to Customize the Integration Data Model in UCMDB	190
Predefined Query for Population Jobs	192
SCCM/SMS Integration Package	192
SMS Adapter Configuration Files	194
Troubleshooting and Limitations – Microsoft SCCM/SMS Integration	195
Chapter 16: NetApp SANscreen/OnCommand Insight Integration	197
Overview	198
Supported Versions	198
Topology	198
How to Discover NetApp SANscreen (NetApp OnCommand Insight)	200
SANscreen Integration by WebServices Job	202
Adapter	202
Parameters	202
Integration Flow	202
SANscreen Adapter	203
Troubleshooting and Limitations – NetApp SANscreen Integration	204
Chapter 17: NetApp OnCommand Insight (OCI) Pull Integration	206
Overview	207
Supported Versions	207
How to Discover NetApp OnCommand Insight (OCI)	207
NetApp OCI Integration Pull Adapter	208
Chapter 18: ServiceNow Integration	211
Overview	212
Supported Versions	212
Push Integration with ServiceNow	212
How to Push Data from UCMDB to ServiceNow	213
Push Integration Mechanism	215
Sample Integration Push Query	216
Supported CITs	217
Pushing Additional CITs	217
Population from ServiceNow	219
How to Populate UCMDB with Data from ServiceNow	219
Population Flow	221
Jython Scripts for Population	222
Mapping Files for Population Flow	223
Supported CITs	225

Troubleshooting and Limitations – ServiceNow Integration	226
Troubleshooting – ServiceNow Integration	226
Limitations – ServiceNow Integration	236
Chapter 19: ServiceNow Integration Using Enhanced Generic Adapter	237
Overview	239
Supported Versions	239
ServiceNow Prerequisites	239
Out of the Box Connector Update Set	240
Setting up ServiceNow for the UCMDB Integration	243
Enable the Multiple Insert Plug-in in ServiceNow	243
Create Inbound Services for Class Models	243
Create Transform Maps between Staging Tables and Target Tables ...	244
Create a Role with Minimal Rights for the Integration	245
Push Integration with ServiceNow	247
How to Push Data from UCMDB to ServiceNow	247
Push Integration Mechanism	251
How to Enable Web Services Security (WS-Security, WSS) for the Push Integration	252
CI Reconciliation	255
Sample Integration Push Query	256
Supported CITs	257
Pushing Additional CITs	257
Population from ServiceNow	258
How to Populate UCMDB with Data from ServiceNow	258
Population Flow	260
Sample Integration Population Query	261
Supported CITs	263
External Class Model Flow	263
Incremental Update	264
Connectors	264
Performance	265
Configuration	265
adapter.properties Configuration	265
ServiceNowConfig.xml Configuration	268
Create a Custom coalesce Field instead of Using the Correlation ID Field	270

ServiceNow Changes	270
Adapter Changes	272
ServiceNow Generic Adapter Enhancements in CP27	272
Changes in ServiceNowConfig.xml	275
Topology Population Flow	281
Standard (Referenced) Population	282
Referencing topology Population	283
Relationship Topology Population	284
Troubleshooting and Limitations – ServiceNow Integration Using Enhanced Generic Adapter	285
Troubleshooting – ServiceNow Integration Using Enhanced Generic Adapter	286
Limitations – ServiceNow Integration Using Enhanced Generic Adapter	289
Chapter 20: Troux Integration	290
Overview	291
Integration Overview	291
Supported Versions	292
Use Cases	292
How to Work with the Troux Push Adapter	293
How to Run a Troux Population Job	298
Chapter 21: UCMDB to XML Adapter	301
Overview	302
Integration Mechanism	302
How to Export UCMDB to XML	302
Adapter	303
Troubleshooting and Limitations – UCMDB to XML Adapter	304
Send documentation feedback	306

Chapter 1: Aperture VISTA Integration

This chapter includes:

Overview	12
Supported Versions	12
Topology	12
How to Use the Aperture VISTA Integration Adapter	13
Aperture VISTA by SQL Adapter	15
Discovery Mechanism	16

Overview

Aperture VISTA is used to model datacenter information, including the exact location of a physical server in a rack, row, space, and floor of a datacenter. VISTA also contains detailed information about the power supply to racks and individual servers. This enables impact analysis from a power supply point of view, and with integration it becomes possible to analyze the impact of power failure on applications, business services and lines of business in UCMDB.

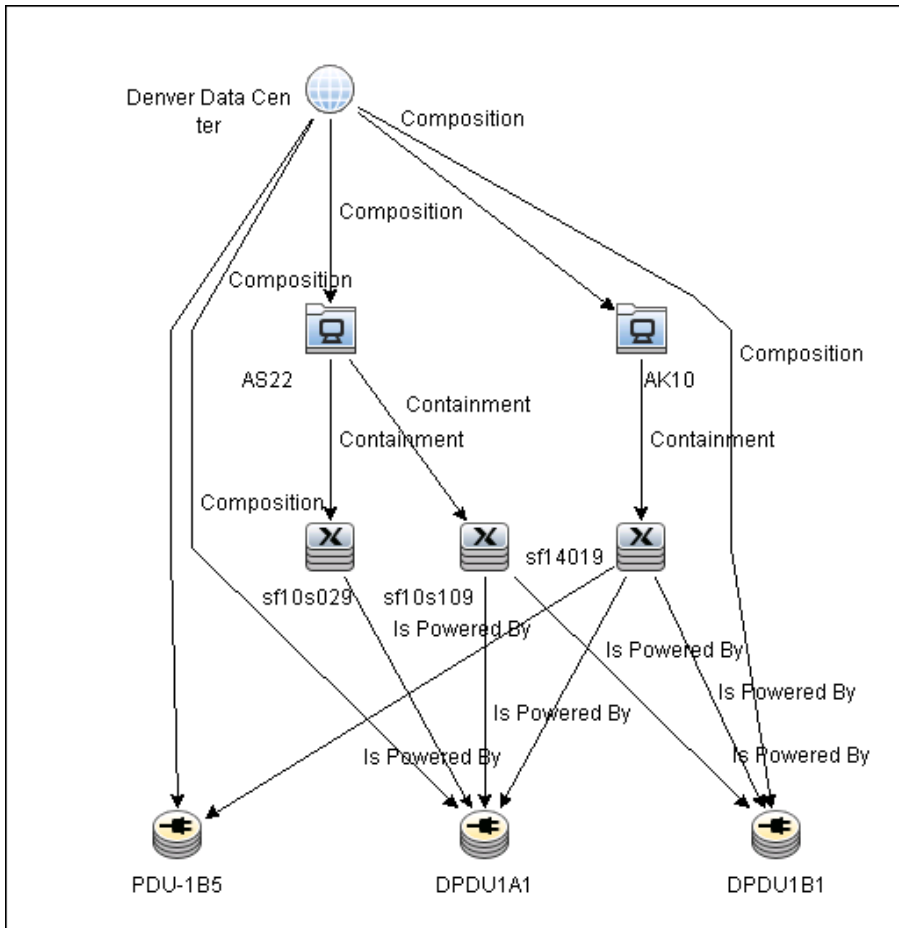
Integration is accomplished by running SQL queries on the Aperture VISTA SQL database.

Supported Versions

UCMDB supports integration with Aperture VISTA version 600. Aperture Integration Management Software Package version 2.0 or later is required.

Topology

The following image displays datacenter topology from VISTA.



Note: For a list of discovered CITs, see ["Discovered CITs" on page 16](#).

How to Use the Aperture VISTA Integration Adapter

1. Prerequisite - Scripts

The following scripts should be run on the VISTA database:

- **v600_VIP_DAL_DV_Devices.sql**
- **v600_Device_to_PDU.sql**

The scripts are located in the discovery probe at **<DataFlowProbe_Home>\runtime\probeManager\discoveryResources\VistaScripts**

2. Prerequisite - Credentials

Population is accomplished using SQL queries over JDBC. The following credentials should be defined:

- Generic DB Protocol (SQL)

For credential information, see "Supported Protocols" in the *UCMDB Discovery and Integrations Content Guide - Supported Content* document.

3. Run the job

- Run **Range IPs by ICMP** to discover the IP address of the SQL Server used by Aperture VISTA.
- Run **Database TCP Ports** to discover SQL Server ports on the IP addresses discovered above.
- Run **MSSQL Connection by SQL** to discover the SQL Server instance used by Aperture VISTA.
- Run **MSSQL Topology by SQL** to discover database instances in the SQL Server instance discovered above.
- Create a new integration point, and use the **Aperture VISTA by SQL** adapter to discover datacenter and power infrastructure from VISTA.

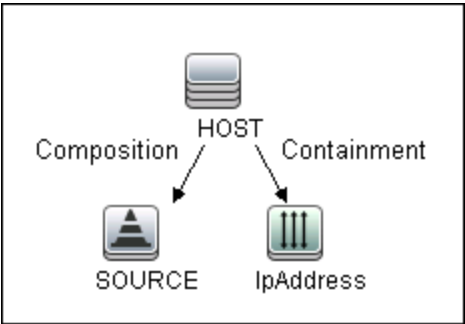
Aperture VISTA by SQL Adapter

This section contains details about the adapter.

Input CIT

Microsoft SQL Server

Input Query



Triggered CI Data

Name	Value
credentialsId	\${SOURCE.credentials_id}
ip_address	\${SOURCE.application_ip}
port	\${PROCESS.application_port}

Used Script

Aperture_Vista_by_SQL.py

Discovered CITs

- Chassis
- Composition
- Containment
- Datacenter
- DatacenterResource
- Node
- PowerDistributionUnit
- Rack
- RemotePowerPanel
- Usage

Discovery Mechanism

Aperture VISTA uses a Microsoft SQL Server database as its data repository. The Aperture Integration Management software package (v2.0 or greater) adds some views to the VISTA database, and this integration adapter collects information by running SQL Queries against these views.

The integration adapter in this package is triggered by SQL Server instances in UCMDB that have a database instance with **vista** in their name. Out-of-the-box discovery jobs may be used to discover these SQL Server database instances.

The adapter works as follows:

1. Datacenter and Power Infrastructure Details

It runs the following SQL query to get details on Datacenter and Power infrastructure from the Aperture VISTA data repository:

```
SELECT device_name, device_serial_number, device_asset_class, device_
manufacturer, device_model, device_model_info, rack_name, rack_asset_number,
rack_serial_number, row_name, grid_location, space_name, floor, building_name,
parent_name, parent_serial_number
```



```
FROM vista.dbo.vip_dal_dv_devices

WHERE device_asset_class='SERVER' OR device_asset_class='PDU' OR device_asset_
class='RPP' OR device_asset_class='CHASSIS' OR device_asset_class='RACK'

GROUP BY device_name, device_serial_number, device_asset_class, device_
manufacturer, device_model, device_model_info, rack_name, rack_asset_number,
rack_serial_number, row_name, grid_location, space_name, floor, building_name,
parent_name, parent_serial_number

ORDER BY device_asset_class, device_name
```

2. Identification of Power Supply Routing

After all the infrastructure, power, and server CIs are created, the adapter uses the following query to identify power supply routing to servers:

```
SELECT downstream_device_name, downstream_device_serial_number, upstream_
device_name, upstream_building_name

FROM vista.dbo.vip_dal_pwr_device_power_sources

WHERE (downstream_device_name IS NOT NULL OR downstream_device_serial_number IS
NOT NULL) AND upstream_device_name IS NOT NULL AND upstream_building_name IS
NOT NULL AND upstream_node_type='PDU'

GROUP BY downstream_device_name, downstream_device_serial_number, upstream_
device_name, upstream_building_name

ORDER BY downstream_device_name
```

Chapter 2: Arxscan Arxview Integration

This chapter includes:

Overview	19
Supported Versions	19
Topology	19
Discovery Mechanism	21
Create an Integration Point between Arxscan Arxview and UCMDB	21
Data Mappings	24
Arxview Integration Adapter	25
Troubleshooting and Limitations – Arxscan Arxview Integration	26

Overview

Arxscan Arxview is a management and monitor tool for enterprise Network-attached storage (NAS) and Storage Area Network (SAN).

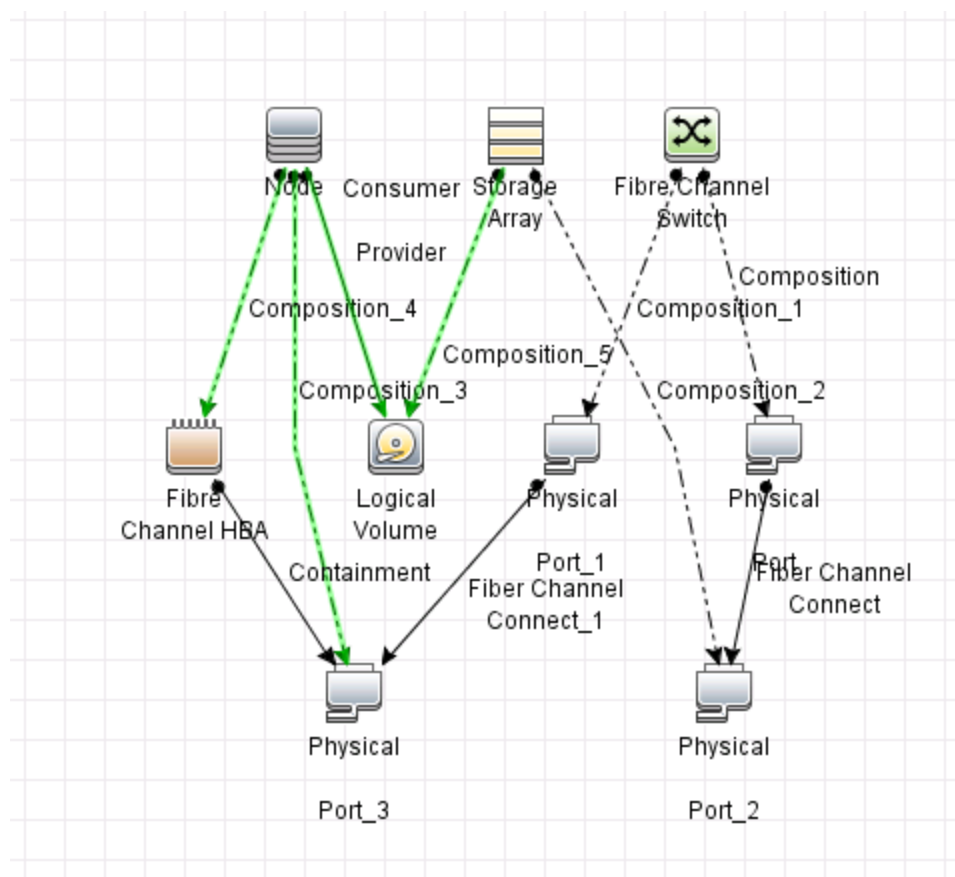
This integration only supports population from Arxscan Arxview. The population adapter pulls storage data from Arxscan Arxview, and then the data is sent to UCMDB.

Supported Versions

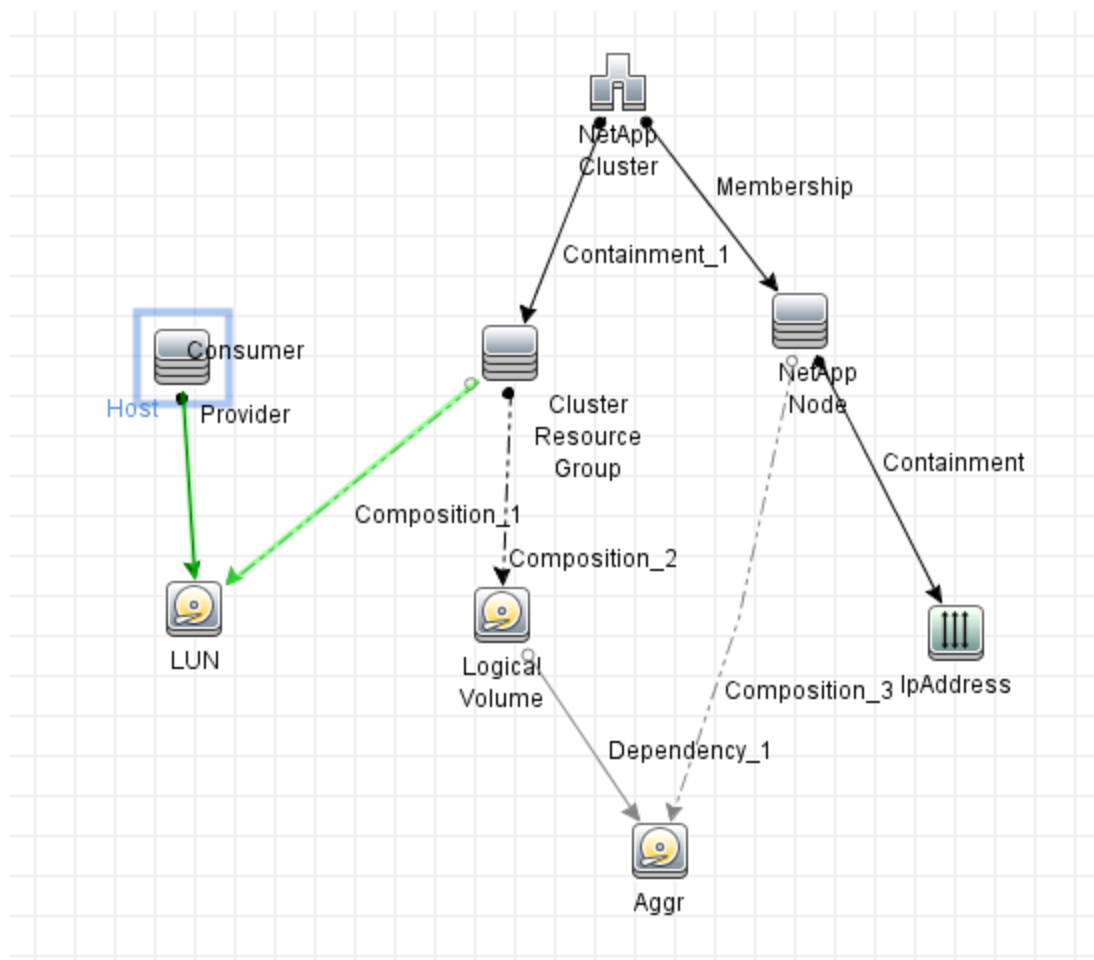
This integration adapter supports Arxscan Arxview version 2.x.

Topology

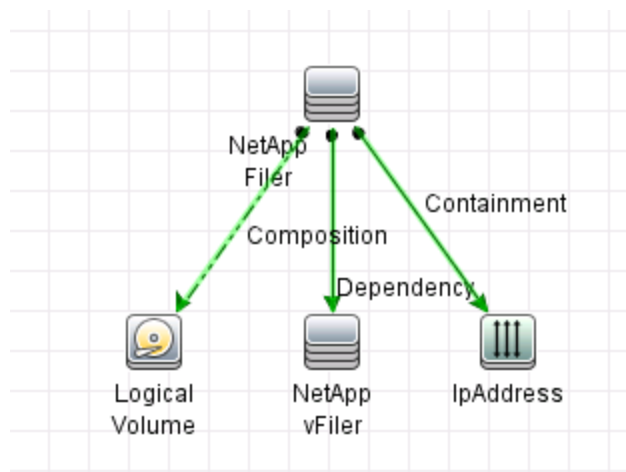
The following image displays the topology of SAN.



The following image displays the topology of NetApp Cluster.



The following image displays the topology of NetApp 7-mode.




Note: For a list of discovered CITs, see ["Discovered CITs" on page 25](#).



Discovery Mechanism



A REST client on the Data Flow Probe connects to the Arxscan Arxview REST API to fetch data.


Create an Integration Point between Arxscan Arxview and UCMDB

To create an integration point between Arxscan Arxview and UCMDB, do the following:

1. Log in to UCMDB as an administrator.
2. Navigate to **Data Flow Management > Integration Studio**.
3. Click .
4. Complete the **Integration Properties** and **Adapter Properties** fields as shown in the following table:

Field (*Required)	Description
Integration Properties section	
*Integration Name	Type the name (unique key) of the integration point.
Integration Description	Type a description of the current integration point.
*Adapter	Click  and then select Third Party Products > Arxview Integration .
*Is Integration Activated?	Select this option to indicate that the integration point is active.
Adapter Properties section	
*Arxview URL	The URL of Arxscan Arxview server.
*Credentials ID	Click  . The Generic Protocol is already selected in the Protocol pane and in the Credentials list.


Field (*Required)	Description
	<p>Note:</p> <ul style="list-style-type: none"> Other credentials are not supported. Ensure that the account has Read access to Arxscan Arxview. <p>To configure the credential for this integration point,</p> <ol style="list-style-type: none"> Ensure the Generic Protocol is selected, and then click . Enter values for the following fields and then click OK: <ul style="list-style-type: none"> Network Scope: Use the default value ALL. User Label: Type a label for the credential. User Name: Provide the user name for the Arxscan Arxview account. Password: Click  and enter the password for the Arxscan Arxview account. Click OK.
Remote JVM Arguments	<p>The JVM parameters that should be passed to the remote process.</p> <p>Default: -Xms1024m -Xmx2560m</p>
Remote JVM Class Path	<p>The external JVM class path.</p> <p>Default: %minimal_classpath%</p>
Run In Separate Process	<p>Specifies whether to run the adapter in the external JVM.</p> <p>Default: true</p>
Trust All SSL Certificates	<p>Specifies whether to trust all SSL certificates when the Arxscan Arxview server does not have a valid certificate.</p> <p>Type one of the following values:</p> <p>true. Trust all SSL certificates.</p> <p>false. Do not trust all SSL certificates.</p> <p>Default: true</p>
*Data Flow Probe	<p>The name of the Data Flow Probe/Integration service used to execute the synchronization from.</p> <p>Select IntegrationService for this integration.</p>


Field (*Required)	Description
	<p>Note: If the IntegrationService option does not exist, consult with your UCMDB administrator for the best selection for your requirements.</p>
Trigger CI Instance	<p>Click  and select one of the following options from the drop-down menu:</p> <p>Select Existing CI. Select this option only if you want to select a valid, existing CI. The Select Existing CI pane appears.</p> <p>Create New CI. Select this option if you want to create a new CI. The Topology CI Creation Wizard appears. Complete the creation of the CI using the Wizard.</p> <p>Note: For details on the Topology CI Creation Wizard, see "Topology CI Creation Wizard" in the <i>Data Flow Management section of the UCMDB Help</i>.</p>

5. Click **OK**.

The integration point is created and its details are displayed.

Note:

- Your settings are not saved to the server until you click **OK**.
- You can edit this integration point by selecting the Integration Point name in the Integration Point pane and then clicking .

6. (Optional) To start the job immediately, click  to perform a full synchronization.

Note:

- The first Full Synchronization may take a while to complete.
- If you do not want the job to begin immediately, the job will be scheduled to run.

To see results of the job, do the following:

1. In Integration Studio, in the Integration Job pane, right-click the **Pull data from Arxview** job.
2. In the drop-down menu, click **Show Results of Job**.

Note:

- The results of the previous job run that are stored in the Probe's database are displayed.
- If the job has not yet run, the following message is displayed: "no results found on selected job".
- Results are only displayed when jobs are run successfully.

Data Mappings

The following entities are mapped from Arxview to UCMDB:

Arxscan Arxview	UCMDB
SAN Array	Storage Array
NAS Filer	NetApp Filer
NAS Virtual Filer	NetApp Filer
NAS Cluster	NetApp Cluster
NAS Node	NetApp Node
NAS VServer	ClusterResourceGroup
FC Switch	Fibre Channel Switch
FC Switch Port	Fibre Channel Port
HostHBAAadapterPort	Fibre Channel Port
HostHBAAadapter	Fibre Channel HBA
NetApp Volume	Logical Volume
NetAppC Volume	Logical Volume
NetAppC Aggregate	Logical Volume
LUN	Logical Volume
Host	Node

Arxview Integration Adapter

This section contains details about the adapter.

Input CIT

Discovery Probe Gateway

Used Script

- arxview_decorators.py
- arxview_validators.py
- arxview_mapping_interfaces.py
- arxview_mapping_implementation.py
- arxview_mapping_file_manager.py
- arxview_rest.py
- arxview_client.py
- arxview_connection_data_manager.py
- arxview_pull.py

Discovered CITs

- Storage Array
- NetApp Filer
- NetApp Cluster
- NetApp Node
- ClusterResourceGroup
- Fibre Channel Switch
- Fibre Channel Port
- Fibre Channel HBA

- Logical Volume
- Node

Global Configuration File

Arxview/arxview.xml

Adapter Parameters

Parameter	Default Value	Description
Arxview URL		The URL of Arxscan Arxview server.
Trust All SSL Certificates	true	Specifies whether to trust all SSL certificates when the Arxscan Arxview server does not have a valid certificate.
credentialsId		The credential used for the Arxscan Arxview integration.
remoteJVMArgs	-Xms1024m -Xmx2560m	The JVM parameters that should be passed to the remote process.
remoteJVMClasspath	%minimal_ classpath%	The external JVM class path.
runInSeparateProcess	true	Specifies whether to run the adapter in the external JVM.

Troubleshooting and Limitations – Arxscan Arxview Integration

This section describes troubleshooting and limitations for the Arxscan Arxview integration.

Problem: You may receive the following error message: “Failed to connect to remote process: details: ***** java heap size“.

Solution: Increase the maximum Java heap size by editing the **remoteJVMArgs** parameter, for example, change the parameter value to **-Xms1024m -Xmx4096m**.

Chapter 3: Atrium Integration

This chapter includes:

Overview	28
Supported Versions	28
How to Work with the Data Push to Atrium Adapter	29
How to Work with the Population from Atrium Adapter	34
Atrium Push Job	38
Data Push into Atrium Adapter	38
Import Data from Atrium Job	40
Population from Atrium Adapter	40
Mapping Files	43
Mapping Files Overview	43
Mapping File Structure	43
Mapping File Elements	44
Troubleshooting and Limitations – Atrium Integration	48

Overview

UCMDB-Atrium integration consists of two independent, bi-directional parts: the **Data Push into Atrium** and the **Population from Atrium**.

- The **Data Push into Atrium** in UCMDB replicates CIs and relationships to Atrium and Remedy.

The out-of-the-box integration does not transfer a specific list of CIs and relationships, but does enable you to replicate any CI or relationship from UCMDB to Remedy or Atrium.

For examples of enabling the integration with commonly used CIs and relationships, see ["Configure synchronization queries" on page 32](#).

- The **Population from Atrium** in UCMDB pulls CIs and relationships from Atrium to UCMDB.

Supported Versions

Universal CMDB integrates with the following BMC products:

- BMC Remedy Service Desk (Remedy) versions 7.0, 7.1, 7.5, and 7.6
- BMC Atrium CMDB (Atrium) versions 2.0, 2.1, 7.5.x, 7.6.x and earlier, 8.1.x, and 9.x.

Note: Support for Atrium 9.x requires UCMDB 10.22 or later. This is because Atrium 9.x drivers run only on Java Runtime Environment (JRE) 8 that is bundled with UCMDB 10.22 or later.

How to Work with the Data Push to Atrium Adapter

This task includes the following steps:

- ["Prerequisite - Set up protocol credentials" below](#)
- ["Configure the Properties file" below](#)
- ["Configure the Data Flow Probe" on the next page](#)
- ["Configure synchronization queries" on page 32](#)
- ["Create XML mapping files" on page 32](#)
- ["Create an integration point" on page 33](#)
- ["Define a Job" on page 33](#)
- ["Invoke a full run of the job" on page 33](#)

1. Prerequisite - Set up protocol credentials

Make sure that you have set up the Remedy protocol. For credential information, see "Supported Protocols" in the *UCMDB Discovery and Integrations Content Guide - Supported Content* document.

2. Configure the Properties file

Configure the **push.properties** file: **Data Flow Management > Adapter Management > Resources > Packages > AtriumPushAdapter > Configuration Files > push.properties**.

Property	Description
jythonScript.name	The name of the Jython script that is invoked by this push adapter.
mappingFile.default	The default XML mapping file used by mapping if a specific XML mapping file is not defined for an integration query. At least one default mapping file must be present in every adapter.
DebugMode	If this value is set to true , the CI and relationships being pushed to Remedy/Atrium are also saved to XML files on the Data Flow Probe, under the following folder:

Property	Description
	/discoveryResource/AtriumPushAdapter/work.
deleteRelations	<p>By default, UCMDB deletes CIs and their connected relationships.</p> <p>If this value is set to true, UCMDB supports the following two scenarios:</p> <ul style="list-style-type: none"> Deletes CIs and their connected relationships Deletes relationships only
smartUpdateIgnoreFields	A comma separated list of attributes (transferred from UCMDB to Atrium) that should not be used to check whether a CI has changed in Atrium. For example, as updateTime always changes, you would not want to update a CI in Atrium just because this attribute has changed.
sortCSVFields	Parameter that includes the TQL results of CSV aggregated fields that must always be sorted. When child attribute values are mapped and aggregated as CSV, the results are not sorted. This can trigger an update, even though nothing has changed in Atrium. To prevent an update, add here the CSV aggregated fields that must always be sorted.
testConnNameSpace	Must be set to the BMC NameSpace being used for test connection purposes (for example, BMC.CORE).
testConnClass	Must be set to the name of a BMC class, to query for connection test purposes (for example, BMC_ComputerSystem).

3. Configure the Data Flow Probe

- a. Create the **AtriumPushAdapter** folder in the following directory on the Data Flow Probe server: **<DataFlowProbe_Home>\runtime\probeManager\discoveryResources**.
 - **For Atrium 7.6.04 and earlier versions:** Copy the JAR and DLL files listed in the table below from the BMC server to the following directory on the Data Flow Probe server:
<DataFlowProbe_Home>\runtime\probeManager\discoveryResources\AtriumPushAdapter.
 - **For Atrium 8.1:** Copy the files **arapi81_build001.jar** and **cmdbapi.jar** from the BMC server to the following directory on the Data Flow Probe server:
<DataFlowProbe_Home>\runtime\probeManager\discoveryResources\AtriumPushAdapter.

- **For Atrium 9:** Copy the files **arapi90_build002.jar** and **cmdbapi90.jar** from the BMC server to the following directory on the Data Flow Probe server:

**<DataFlowProbe_Home>\runtime\probeManager\discoveryResources\
AtriumPushAdapter.**

For details on deploying packages, see "Package Manager" in the *Administer* section of the *UCMDB Help*.

JAR Files	DLL Files
arapi75.jar	arapi75.dll
arutil75.jar	arencrypt75.dll
cmdbapi75.jar	arjni75.dll
commons-beanutils.jar	arrpc75.dll
commons-codec-1.3.jar	arutiljni75.dll
commons-collections-3.2.jar	arutil75.dll
commons-configuration-1.3.jar	arxmlutil75.dll
commons-digester-1.7.jar	cmdbapi75.dll
commons-lang-2.2.jar	cmdbjni75.dll
log4j-1.2.14.jar	icudt32.dll
oncrpc.jar	icuinbmc32.dll
spring.jar	icuucbmc32.dll
	Xalan-Cbmc_1_9.dll
	XalanMessagesbmc_1_9.DLL
	xerces-cbmc_2_6.dll
	xerces-depdombmc_2_6.dll

Note:

- The AR System Java API is forward and backward compatible with other versions of the AR System, except Atrium version 7.6.0.4 where you must use the SDK of the same version. For a complete compatibility matrix, refer to the "API Compatibility" section in the *BMC Remedy/Atrium Developer Reference Guide*.
- The arencrypt*.dll files are only required if encryption is enabled on the Remedy server.

- b. Edit the **WrapperGateway.conf** file (or **WrapperManager.conf** if the Probe Manager and

Gateway are running in separate mode) in the following directory: **<DataFlowProbe_Home>\bin.**

Add the following line after the **wrapper.java.library.path.2=%content_dll%** line:

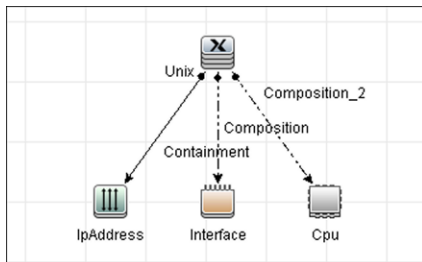
```
wrapper.java.library.path.3=%runtime%/probeManager
/discoveryResources/AtriumPushAdapter
```

- c. **For Atrium 7.6.04 and earlier versions only:** Add the complete path to the Atrium DLL files (for example, **C:\UCMDB\DataFlowProbe\runtime\probeManager\discoveryResources\AtriumPushAdapter**) to the Windows system path on the Data Flow Probe machine.
- d. Restart the Data Flow Probe service.

4. Configure synchronization queries

The CIs and relationships to be pushed to Remedy/Atrium must be queried from UCMDB. Create queries (of type **Integration**) to query the CIs and relationships that have to be pushed to Remedy/Atrium.

An example of such a query (**atrium_push_sample_query**) is included with the Atrium package. To access the query, navigate to **Modeling > Modeling Studio > Root > Integration > Atrium**.



5. Create XML mapping files

For every query created in the step above, create an XML mapping file with the same name as the integration query (the name must have the same case) in the following directory:

<UCMDB Server installation folder>\runtime\fcmdb\CodeBase\AtriumPushAdapter\mappings

A sample mapping file (**atrium_push_sample_query.xml**) is provided out-of-the-box with the Atrium package.

For more details, see ["Mapping Files" on page 43](#).

6. Create an integration point

For details about creating an integration point, see "Integration Point Pane" in the *Data Flow Management section of the UCMDB Help*.

- a. In the Integration Studio, create an integration point, selecting the **Data Push into Atrium** adapter. Enter the following information:

Name	Description
Credentials	<ul style="list-style-type: none"> • Select Remedy Protocol. • Select the credentials to be used with this integration point. <p>For credential information, see "Supported Protocols" in the <i>UCMDB Discovery and Integrations Content Guide - Supported Content</i> document.</p>
Hostname/IP	The host name or IP address of the BMC Remedy server.
Integration Name	The name you give to the integration point.
Is Integration Activated	Select this check box to create an active integration point. You clear the check box if you want to deactivate an integration, for instance, to set up an integration point without actually connecting to a remote machine.
Port	The port number of the BMC Remedy server.
Data Flow Probe	Select the Data Flow Probe that should run this integration.


- b. Test the connection. If a connection is not successfully created, check the integration point parameters and try again.
- c. Save the integration point.

7. Define a Job

For details, see "New Integration Job/Edit Integration Job Dialog Box" in the *Data Flow Management section of the UCMDB Help*.

Select the queries that will synchronize data between UCMDB and Remedy/Atrium. Save the job definition and the integration point.

8. Invoke a full run of the job

In the Integration Studio, on the Job Definition tool bar, click  to run a full discovery job. For details, see "Integration Jobs Pane" in the *Data Flow Management section of the UCMDB Help*.

How to Work with the Population from Atrium Adapter

This task includes the following steps:

- ["Prerequisites - File preparation" below](#)
- ["Prerequisites - Set up protocol credentials" on page 36](#)
- ["Prerequisites - Create XML mapping files" on page 36](#)
- ["Run the job" on page 36](#)

1. Prerequisites - File preparation

- **For Atrium 7.6.04 and earlier versions:** Locate the files listed in the table below on the Remedy ARS and Atrium system, and copy them to:

**<DataFlowProbe_
Home>\runtime\probeManager\discoveryResources\AtriumImportAdapter**

Note: All the files listed in the table below are required.

- **For Atrium 8.1:** Locate the files **arapi81_build001.jar** and **cmdbapi.jar** on the Remedy ARS and Atrium system and copy them to:

**<DataFlowProbe_
Home>\runtime\probeManager\discoveryResources\AtriumImportAdapter**

- **For Atrium 9:** Locate the files **arapi90_build002.jar** and **cmdbapi90.jar** on the Remedy ARS and Atrium system and copy them to:

**<DataFlowProbe_
Home>\runtime\probeManager\discoveryResources\AtriumImportAdapter**

JAR Files	DLL Files
arapi75.jar	arapi75.dll
arutil75.jar	arencrypt75.dll
cmdbapi75.jar	arjni75.dll
commons-beanutils.jar	arrpc75.dll
commons-codec-1.3.jar	arutiljni75.dll

JAR Files	DLL Files
commons-collections-3.2.jar	arutil75.dll
commons-configuration-1.3.jar	arxmlutil75.dll
commons-digester-1.7.jar	cmdbapi75.dll
commons-lang-2.2.jar	cmdbjni75.dll
log4j-1.2.14.jar	icudt32.dll
oncrpc.jar	icuinbmc32.dll
spring.jar	icuucbmc32.dll
	Xalan-Cbmc_1_9.dll
	XalanMessagesbmc_1_9.DLL
	xerces-cbmc_2_6.dll
	xerces-depdombmc_2_6.dll

Note:

- The AR System Java API is forward and backward compatible with other versions of the AR System, except Atrium version 7.6.0.4 where you must use the SDK of the same version. For a complete compatibility matrix, refer to the "API Compatibility" section in the *BMC Remedy/Atrium Developer Reference Guide*.
- The arencrypt*.dll files are only required if encryption is enabled on the Remedy server.

- b. Edit the **WrapperGateway.conf** file (or **WrapperManager.conf** if the Probe Manager and Gateway are running in separate mode) in the following directory: **<DataFlowProbe_Home>\bin**.

Add the following line after the **wrapper.java.library.path.2=%content_dll%** line:

```
wrapper.java.library.path.3=%runtime%/probeManager
/discoveryResources/AtriumPushAdapter
```

- c. **For Atrium 7.6.04 and earlier versions only:** Add the complete path to the Atrium DLL files (for example, **C:\UCMDB\DataFlowProbe\runtime\probeManager\discoveryResources\AtriumImportAdapter**) to the Windows system path on the Data Flow Probe machine.
- d. Restart the Data Flow Probe service.

2. Prerequisites - Set up protocol credentials

Configure a generic protocol with the ARS server's username and password.

Note: While creating the generic protocol, set the protocol description to **atrium**.

3. Prerequisites - Create XML mapping files

This step involves creating XML mapping files (in the **<DataFlowProbe_Home>\runtime\probeManager\discoveryResources\TQLEExport\Atrium\data** directory).

These files map the BMC Atrium classes, attributes and relationships to their UCMDB equivalents. To create the XML mapping files for the topology requires identification of the topology to be imported from Atrium, and ensuring an equivalent topology exists in UCMDB. For more details, see ["Mapping Files" on page 43](#).

4. Run the job

In the Integration Studio, create a new integration point.

- a. Provide a name and description for the integration point.
- b. Under **Integration Properties > Adapter**, select the **Population from Atrium** adapter.
- c. Configure the following adapter properties:
 - **ARS_Server**
 - **ARS_Port**
 - **BMC_NameSpace**
- d. Under **Adapter Properties > Data Flow Probe**, select the Data Flow Probe to be used for the integration.
- e. Under **Adapter Properties > Trigger CI instance** select:
 - i. **Select Existing CI** (if you have a valid, existing CI). The **Select Existing CI** pane appears. Select the CI, or
 - ii. **Create New CI** (if you need to create a new CI). The **Topology CI Creation Wizard** appears. Complete the creation of the CI using the Wizard.

Note: For details on the Topology CI Creation Wizard, see "Topology CI Creation Wizard" in the *Data Flow Management section of the UCMDB Help*.
- f. Save the integration point.

- g. Run the job.

Note: For details on running an integration job, see "Integration Studio" in the *Data Flow Management section of the UCMDB Help*.

Atrium Push Job

Adapter

This discovery uses the adapter called **Data Push into Atrium**.

Integration Flow

Integration includes the following activities:

1. **Querying the UCMDB for CIs and relationships.** When an ad-hoc integration job is run in the Integration Studio, the integration process:
 - a. Receives the names of the integration queries that are defined in the job definition for that integration point.
 - b. Queries UCMDB for the results (new, updated, or deleted CIs and relationships) of these defined queries.
 - c. Applies the mapping transformation according to the pre-defined XML mapping files for every query.
 - d. Pushes the data to the Data Flow Probe.
2. **Sending the data to BMC Remedy/Atrium.** On the Data Flow Probe, the integration process:
 - a. Receives the CI and relationship data sent from the UCMDB Server.
 - b. Connects to the BMC Remedy/Atrium server using the Java API.
 - c. Transfers the CIs and relationships.

Data Push into Atrium Adapter

This section contains details about the adapter.

Used Script

pushToAtrium.py

Parameters

Parameter	Description
credentialsId	The credentials ID to use for Atrium connection.
host	The host name or IP address of the remote Atrium server.
port	The Atrium server's connection port (if not using portmapper).
probeName	An internal setting which UCMDB automatically replaces.

Import Data from Atrium Job

Adapter

This job uses the adapter called **Population from Atrium**.

Integration Flow

The integration flow has the following steps:

1. **Querying the Atrium server**

In this step, the integration adapter connects to the Atrium server and queries it for classes, attributes and relationships, described in the XML mapping files. The result of this step is the creation of intermediate XML files (in the `<probe>\runtime\probeManager\discoveryResources\TQLEExport\Atrium\inter` directory).

2. **Mapping the data**

In this step, the data collected from the previous step and stored in the intermediate XML file, is converted into the UCMDB data format based on the mappings defined in the XML mapping files.

3. **Pushing the data to the UCMDB server**

In this final step, after being mapped into the UCMDB object state holder vector format, the data is sent to the UCMDB server.

Population from Atrium Adapter

This section contains details about the adapter.

Input CIT

The input CIT for this adapter is **discoveryprobegateway**. The job uses an instance of the Discovery Probe Gateway which has access to connect to the remote BMC Atrium server.

Used Scripts

The adapter uses the following scripts:

Script	Description
atrium_query.py	Used to query BMC Atrium for data.
atrium_map.py	Used to map the queried data into data UCMDB can use.
atrium_to_ucmdb.py	Used to push imported data into UCMDB.

Discovered CITs

This integration can discover any CIT or relationship which is (a) mapped in the integration and (b) can be queried and converted to its UCMDB equivalent.

Parameters

Parameter	Detail
ARS_Port	The port for connecting to the ARS server. If portmapper is being used, this should be left as 0. Otherwise, specify the TCP port.
ARS_Server	The hostname or IP address of the BMC ARS server.
BMC_NameSpace	The BMC NameSpace to use. (For example: BMC.CORE.)
ChunkSize	The chunk size in which data should be retrieved from the remote server.
DateParsePattern	Set the date pattern to parse Atrium date strings.
DebugMode	Set to true to run the integration in the debug mode; this does not send data to UCMDB. Default: false
DryRunMode	Set to true to run the integration in the dryrun mode; this does not query any data from Atrium. Default: false
MaxCIs	The maximum number of CI synchronized from a remote server.

Parameter	Detail
	Default: 100000
credentialsId	The credentials ID to use for Atrium integration.
remoteJVMArgs	JVM parameters that should be passed to the remote process.
remoteJVMClasspath	The external JVM class path.
runInSeparateProcess	Whether to run the adapter in an external JVM. Default: true.

Mapping Files

This section includes:

Mapping Files Overview	43
Mapping File Structure	43
Mapping File Elements	44

Mapping Files Overview

A mapping file is an XML file that defines which CIT or relationship in UCMDB is mapped to which CIT or relationship in the target data repository.

Mapping files:

- Control which CITs and relationships are to be pushed.
- Control the attributes for the CITs and relationships that are to be mapped.
- Map attribute values from multiple CIs to one target CI.
- Map attributes of children CIs (those having a **containment** or **composition** relationship) to the parent CI in the target data repository. For example:
 - Set a **Number of CPUs** value for a target **node** CI.
 - Set a **Total Memory** value for a target **node** CI.
- Map attributes of parent CIs (those having a **containment** or **composition** relationship) in the target data repository CI. For example, in the Atrium target data repository, set the value of a **Container Server** attribute on the **Installed Software** CIT by retrieving the value of the UCMDB **Installed Software** CI container node.

Mapping File Structure

Every mapping file has the following skeletal structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<integration>
  <info>
    <source ... .. />
    <target ... .. />
  </info>
  <source_ci_type name="...">
    <target_ci_type name="...">
      <targetprimarykey>
        <pkey>...</pkey>
      </targetprimarykey>
      <target_attribute name="..." datatype="..." >
        <map type="..." />
      </target_attribute>
    </target_ci_type>
  </source_ci_type>
</integration>
```

Note: An ellipsis (...) signifies a configurable section.

Mapping File Elements

This section includes:

Main Parent Elements

- **<integration>**. The root element of the XML file. This element has no attributes.
- **<info>**. The source and target data repositories being used, for example:

```
<info>
  <source name="Atrium" versions="7.6" vendor="BMC" />
  <target name="UCMDB" versions="9.0" vendor="HP" />
</info>
```

- **<targetcisis>**. The element that encapsulates the mapping for all CI types.
- **<targetrelations>**. The element that encapsulates the mapping for all relationship types.

CI Type Mapping Elements

- **<source_ci_type>**. The element that defines a CI type of the source data repository, for example:

```
<source_ci_type name="BMC_ComputerSystem" namespace="BMC.CORE" query=""
mode="update_else_insert">
```

- **Attribute: name.** Defines the name of the source CI type.
- **Attribute: namespace.** Defines the namespace in the Atrium system that the CI type is associated with.
- **Attribute: query.** Allows applying an additional filter to the query result.

For example, query=""DatasetId'="BMC.ASSET" AND 'AssetLifecycleStatus'!=8".

- **Attribute: mode.** Defines the mode of the update in the target data repository. The possible values are: **update**, **insert**, and **update_else_insert**.
- **<target_ci_type>**. The element that defines the target CIT, for example:

```
<target_ci_type name="unix">
```

- **Attribute: name.** Defines the name of the target CIT.
- **<targetprimarykey>**. The element that defines a list of all primary keys of the target CIT, for example:

```
<targetprimarykey>
  <pkey>host_key</pkey>
</targetprimarykey>
```

- **<target_attribute>**. The element that defines an attribute mapping from the source CI type to the target CI type attribute. Attribute mapping can be of the following types:

- **Constant.** This type enables setting a constant value on the target attribute:

```
<target_attribute name="data_note" datatype="string" length="127">
  <map type="constant" value="ATRIUM DATA" />
</target_attribute>
```

- **Direct.** This type enables setting a direct value of a source data repository attribute on the target data repository:

```
<target_attribute name="name" datatype="string">
```

```
<map type="direct" source_attribute="Name" />
</target_attribute>
```

- **Compound String.** This type enables the use of the above mapping types together to form more complex values for the target attribute, for example:

```
<target_attribute name="Bunch_O_Data" datatype="char" length="510"
option="uppercase">
  <map type="compoundstring">
    <source_attribute name="name"/>
    <constant value="_UNIX_Server, IP="/>
    <childattr name="ip_address" source_attribute="ip_address"
aggregation="csv"/>
    <constant value=", CPU="/>
    <childattr name="cpu" source_attribute="display_label"
aggregation="csv"/>
  </map>
</target_attribute>
```

Relationship Type Mapping Elements

- **<link>.** The element that defines a relationship mapping from the source data repository to a target data repository, for example:

```
<link source_link_type="composition"
  target_link_type="BMC_HostedSystemComponents"
  source_ci_type_end1="unix"
  source_ci_type_end2="cpu"
  role1="Source"
  role2="Destination"
  mode="update_else_insert">
  <target_ci_type_end1 name="BMC_ComputerSystem"
    superclass="BMC_System" />
  <target_ci_type_end2 name="BMC_Processor"
    superclass="BMC_SystemComponent" />
  ... Relationship attribute mapping elements similar to the CI type attribute
  mapping elements ...
</link>
```

- **Attribute: source_ci_type_end1.** The **End1** CI type of the source link.
- **Attribute: source_ci_type_end2.** The **End2** CI type of the source link.
- **Attribute: source_link_type.** Defines the name of the source link. To specify namespace, use

the following format: "NAMESPACE:CLASSNAME". For example: BMC.J2EE:BMC_J2EEApplicationServer. The same format applies to <antecedent> and <dependent> elements.

- **Attribute: target_link_type.** Defines the name of the target link. To specify namespace, use the following format: "NAMESPACE:CLASSNAME". For example: BMC.J2EE:BMC_J2EEApplicationServer. The same format applies to <antecedent> and <dependent> elements.
- **<target_ci_type_end1>.** Used to specific the value of the target links end1 CI type.
- **<target_ci_type_end2>.** Used to specific the value of the target links end2 CI type.

Troubleshooting and Limitations – Atrium Integration

The integration mapping file enables the mapping only of concrete CI types and relationships to the CI types and relationships in BMC Remedy/Atrium. That is, a parent CIT cannot be used to map children CIs. For example, if **UCMDB Node** is mapped to **BMC_ComputerSystem**, any Node CIT of type **Unix** is not transferred. A mapping must be separately created for **Unix** to **BMC_ComputerSystem**.

Chapter 4: BMC Remedyforce Integration

This chapter includes:

- Overview 50
- How to Create Connected Apps in Salesforce 50
- How to Populate UCMDB with Data from BMC Remedyforce 51
- How to Push Data from UCMDB to BMC Remedyforce 53
- Troubleshooting and Limitations – BMC Remedyforce Integration 57

Overview

BMC Remedyforce-UCMDB integration consists of two independent, bi-directional parts: **Data Push into BMC Remedyforce** and **Population from BMC Remedyforce**.

- **Data Push into BMC Remedyforce** provides the ability to push CIs and relationships from UCMDB to BMC Remedyforce.
- **Population from BMC Remedyforce** provides the ability to import CIs and relationships from BMC Remedyforce into UCMDB.

How to Create Connected Apps in Salesforce

To create Connected Apps in Salesforce, do the following:

1. Log in to Salesforce as an administrator.
2. In the drop-down list of the account (in the upper-right corner), select **Setup**.
3. In the left-hand pane, go to **App Setup > Create > Apps**.
4. In the **Connected Apps** pane, click the **New** button.
5. On the **New Connected App** page, fill the following required fields under **Basic Information**:
 - **Connected App Name**. For example, UCMDB Integration.
 - **API name**. For example, UCMDB Integration.
 - **Contact Email**.
6. Go to **API (Enable OAuth Settings)**, and select **Enable OAuth Settings**.
 - In the **Callback URL** field, enter **https://login.salesforce.com/**.
 - In the **Selected OAuth Scopes** field, select **Access and manage your data (api)**, and then click **Add**.
7. Click the **Save** button to save the new Connected App.
8. In the **Connected Apps** list, find the App that you just created, and then click **Manage**.

- a. On the page that opens, click the **Edit** button.
 - b. Under **OAuth policies**, select **All users may self-authorize** in the **Permitted Users** list, and then click the **Save** button.
9. Go back to the **Connected Apps** list, and click the App that you just created.
10. Go to **API (Enable OAuth Settings)**, and note down the **Consumer Key** and **Consumer Secret**, which will be used for the configuration of Credential in UCMDB.

How to Populate UCMDB with Data from BMC Remedyforce

1. Prerequisite - Set up protocol credentials

Make sure that you have set up the Salesforce Rest Protocol. For credential information, see "Supported Protocols" in the *UCMDB Discovery and Integrations Content Guide - Supported Content* document.

2. Create XML Mapping Files.

The XML mapping file describes the topology that should be pulled from the BMC Remedyforce instance and how it should be mapped to the UCMDB class model.

To create your own mapping file, do the following:

- a. Go to **Data Flow Management > Adapter Management > Resources > Packages > Remedyforce_Pull > Configuration Files > RemedyforcePull/sample_mapping.xml**.

The **sample_mapping.xml** file is a sample mapping file for your reference, and all mappings are commented out by default.



- b. Right-click this file and select **Save As**.
 - c. Enter a name that you want and click **OK**.
 - d. Customize your own mapping file. The main mapping file elements are described below:

Element	Attribute
<source_ci_type>	name: The CMDB class name of BMC Remedyforce, such as BMC_ComputerSystem . needContainer: When set to true , the CI cannot exist without a root

Element	Attribute
	container. This is required by the reconciliation rule of UCMDB, for example, a running software must have a node as its root container.
<target_ci_type>	name: The CI type in UCMDB, such as node .
<target_attribute>	name: The attribute of the CI type in UCMDB.
<map>	type: The type of the attribute mapping, including direct and constant . <ul style="list-style-type: none"> direct. This type enables taking the attribute value of a BMC Remedyforce CI and setting it to a UCMDB CI attribute unchanged. constant. This type enables setting a constant value on a UCMDB attribute.
<map>	source_attribute: The field name of BMC Remedyforce CMDB class, such as BMCServiceDesk__Description__c .
<link>	source_link_type: The name of relationship in BMC Remedyforce.
<link>	target_link_type: The name of relationship in UCMDB.
<link>	source_ci_type_end1 and source_ci_type_end2: The class names of BMC Remedyforce.
<link>	target_ci_type_end1 and target_ci_type_end2: The CI types in UCMDB.

3. Create the Integration Point.

Define the integration point as follows:

- In UCMDB, go to **Data Flow Management > Integration Studio**.
- Click the **New Integration Point**  button.
- In the New Integration Point dialog box, enter a name and description for the integration point.
- In the Adapter field, click the **Select Adapter**  button.
- In the Select Adapter list, select **Remedyforce Pull Integration** and click **OK**.
- In the Adapter Properties section, enter the following required properties:

Field	Value
Credentials	Select the credential entries you created in step 1 for the Salesforce Rest

Field	Value
ID	Protocol.
Mapping File	The name of the XML mapping file you created in step 2. Leave blank if you want all the mapping files in this folder to be processed.
Remote JVM Arguments	The JVM parameters that should be passed to the remote process when the integration is running in a separate process.
Run in Separate Process	Indicated whether to run in a separate process. Default: False .
Data Flow Probe	Select the name of the Data Flow Probe on which the integration will run.
Trigger CI Instance	Select the DataFlowProbe CI, which refers to the same Probe as Data Flow Probe property above.

- g. Click **OK** to save the integration point. At this point, a default integration job definition is automatically created.

4. Define a synchronization schedule for the integration job, if desired.

Jobs can also be run manually, without a schedule.

5. Run full data synchronization.

Click  to synchronize all data.

How to Push Data from UCMDB to BMC Remedyforce

1. Prerequisite - Set up protocol credentials

Make sure that you have set up the Salesforce Rest Protocol. For credential information, see "Supported Protocols" in the *UCMDB Discovery and Integrations Content Guide - Supported Content* document.

2. Configure queries

The CIs and relationships to be pushed to BMC Remedyforce have to be queried from UCMDB using TQL queries. Create integration type queries to query the CIs and relationships that have to be pushed to BMC Remedyforce.

An example of such a query, **RemedyforcePushSampleQuery**, is included with this integration package, and can be viewed in the Modeling Studio.

For details on viewing queries in the Modeling Studio, see the *Modeling section of the UCMDB Help*. For details about the push query, see **UCMDB Help > Developer Reference > Creating Discovery and Integration Adapters > Developing Push Adapters > Build an Adapter Package**.

Note: An attribute mapping is required for each CI that maps **global_id** of UCMDB to BMC Remedyforce as its unique ID. Below is an example:

```
<target_attribute name="__ID__" datatype="String">
  <map type="direct" source_attribute="global_id"/>
</target_attribute>
```

3. Create XML mapping files

For every query created in the step above, create an XML mapping file with exactly the same name (case-sensitive) as the integration query.

You can find a sample mapping file **RemedyforcePushSampleMapping.xml** through **Data Flow Management > Adapter Management > Resources > Packages > Remedyforce_Push > Configuration Files > RemedyforcePushAdapter/mappings /RemedyforcePushSampleMapping.xml**.

For details on how to create your own mapping file, see **UCMDB Help > Developer Reference > Creating Discovery and Integration Adapters > Developing Push Adapters > Create Mappings**.

4. Advanced usage of the mapping

In order to support more complex requirements of the mapping, you can embed dynamic business logic in the mapping file. Two new datatypes are defined to achieve the goal:

- Evaluate an expression

The datatype is **eval**. This mapping can evaluate a Python expression as a value of the target attribute. Below is an example:

```
<target_attribute name="BMCSERVICEdesk__Name__c" datatype="eval">
```

```
<map type="constant" value=" 'Intel' if ('Intel' in
CI['TEL_COL_Vendor__c']) else 'AMD' "/>
</target_attribute>
```

In this example:

CI is a build-in container. If the target CI attribute TEL_COL_Vendor__c contains string Intel, set the BMCServiceDesk__Name__c attribute value to Intel. Otherwise, set it to AMD.


- Run a function

The datatype is function. This mapping can run a customized function that contains the custom business logic. The return value of the function is the value of the target attribute. The value of the function is composed of a python module name and a function name with a dot.

Below is an example:

```
<target_attribute name="BMCServiceDesk__Name__c" datatype="function">
    <map type="constant" value="my_module.set_name" />
</target_attribute>
```

To create your own module, for example, **my_module**, do the following:

- i. Go to **Data Flow Management > Adapter Management > Resources > Packages > Remedyforce_Push > Scripts**.
- ii. Click the  button and select **New Jython Script**.
- iii. Type **my_module** in the **Name** field, and then click **OK**.
- iv. Put the following code into **my_module.py**:

```
#coding=utf-8
import string
import re

import logger
import modeling

from appilog.common.system.types import ObjectStateHolder
from appilog.common.system.types.vectors import
ObjectStateHolderVector


def set_name(CI):
    name = "AMD"
    if "Intel" in CI['TEL_COL_Vendor__c']:
        name = "Intel++"
    return name
```

5. Create the integration point

Define the integration point as follows:

- a. In UCMDB, go to **Data Flow Management > Integration Studio**.
- b. In the New Integration Point dialog box, enter a name and description for the integration point.


Ensure the **Is Integration Activated** option is enabled.

- c. In the Adapter field, click the **Select Adapter**  button.
- d. In the Select Adapter list, select **Remedyforce Push Integration** and click **OK**.
- e. In the Adapter Properties section, enter the following required properties:

Field	Value
Credentials ID	Select the credential entries you created in step 1 for the Salesforce Rest Protocol. For credential information, see Supported Protocols in the <i>UCMDB Discovery and Integrations Content Guide</i> .
Data Flow Probe	Select the name of the Data Flow Probe on which the integration will run.
Additional Probes	This field is unavailable for this integration.

- f. Click **Test connection** to verify the connectivity, and click **OK**. If the connection fails, verify that the information provided is correct.
- g. Click **OK** to save the integration point.
- h. Add a new job definition to the integration point. Provide a name for the job definition and select the queries to use to synchronize data from UCMDB to BMC Remedyforce. Define a synchronization schedule if required.

Note: Jobs can also be run without a schedule.

- i. Save the job definition, and then the integration point.
- j. Click  to run a full synchronization for each job at least once.

Troubleshooting and Limitations – BMC Remedyforce Integration

If something goes wrong, check the debug log file **probeMgr-adaptersDebug.log** on Data Flow Probe. All the jobs' running information is recorded there.

Chapter 5: CA CMDB Integration

This chapter includes:

- Overview 59
- Supported Versions 59
- Integration Mechanism 59
- How to Work with the CA CMDB Push Adapter 61
- Integration Query 64
- Troubleshooting and Limitations – CA CMDB Integration 64

Overview

The UCMDB - CA CMDB integration adapter allows pushing CIs and relationships from UCMDB into CA CMDB.

This is achieved by querying the UCMDB for CIs and Relationships based on queries defined in the push integration adapter. The output of the queried CIs and Relationships are saved in an XML file.

GRLoader, a utility provided with CA CMDB, transfers the CIs and Relationship data stored in the XML file into CA CMDB. An XML mapping file is used to define how the CIs and Relationships in UCMDB are related to the CIs and Relationships in CA CMDB.

The CA CMDB integration package is bundled in **CACmdbPushAdapter.zip**.

Supported Versions

UCMDB supports integration with CA CMDB R12.0 and R12.5.

Integration Mechanism

This section describes the UCMDB - CA CMDB integration mechanism:

1. UCMDB is queried for CIs and Relationships
 - a. When an ad-hoc job is run from the defined integration point, the integration receives the names of the integration queries that have been defined in the job definition for that integration point.
 - b. The integration process queries UCMDB for the results of these queries (new/updated/deleted CIs and Relationships), and applies the mapping transformation according to the pre-defined XML mapping files for every query.
 - c. It then pushes the data to the Data Flow Probes.
2. Queried data is converted into temporary XML files on the Data Flow Probe system

On the Data Flow Probe side, the integration process receives the CI and Relationship data sent

from the UCMDB server, and converts it into a format which can be used as input XML for the GRLoader, a utility provided with CA CMDB used to transfer the CI and Relationship data into CA CMDB.

3. CA CMDB GRLoader utility is invoked on the Data Flow Probe system

The integration process programmatically invokes the CA CMDB GRLoader utility on the Data Flow Probe system with the necessary parameters (for example, CA CMDB server, port, username, and password), using the input XML file created in the previous step to transfers the CIs and Relationship data into CA CMDB.

How to Work with the CA CMDB Push Adapter

The CA CMDB push adapter allows replication of CIs and Relationships from UCMDB to CA CMDB.

This task includes:

- ["Prerequisite - Other" below](#)
- ["Prerequisite - Set up the CA CMDB protocol" below](#)
- ["Configure integration queries" on the next page](#)
- ["Create the XML mapping files" on the next page](#)
- ["Create an integration point" on the next page](#)

1. Prerequisite - Set up the CA CMDB protocol

This integration uses the **CA CMDB protocol**. For credential information, see "Supported Protocols" in the *UCMDB Discovery and Integrations Content Guide - Supported Content* document.

2. Prerequisite - Other

Data Flow Probe System:

- a. Copy all of the files in the CA CMDB system's **%NX_ROOT%\javallib** directory to the **CaCmdbPushAdapter** directory on the data flow probe system:

```
<UCMDB Installation>\DataFlowProbe\runtime\probeManager\  
discoveryResources\CaCmdbPushAdapter
```

- b. Locate the file, **NX.ENV**, in the **CaCmdbPushAdapter** directory. If the file does not exist, create it in the **CaCmdbPushAdapter** directory and add the following text to it:

```
@NX_LOG=C:/CA/java/lib/log
```

- c. Open **<UCMDB Installation>\DataFlowProbe\runtime\probeManager\discoveryConfigFiles\globalSettings.xml**, locate the following line, and add **",CaCmdbPushAdapter/*.*" as illustrated in bold:**

```
db/oracle/*.*;db/mssqlserver/*.*;db/db2/*.*;db/sybase/*.*;nnm/*.*;AtriumPu  
shAdapter/*.*;CaCmdbPushAdapter/*.*
```

- d. Restart the Data Flow Probe service.

3. Configure integration queries

Create integration queries to query the CIs and Relationships that must be pushed from UCMDB to CA CMDB.

For an example of such an integration query, see ["Integration Query" on page 64](#).

4. Create the XML mapping files

For every integration query that you create, create an XML mapping file with the exact same name as the integration query (case-sensitive). Create the XML files in the following directory:

```
<UCMDB Installation>\UCMDBServer\runtime\fcmdb\CodeBase\  
CaCmdbPushAdapter\mappings
```

For more information about mapping files, see "Prepare the Mapping Files" in the *Developer Reference section of the UCMDB Help*.

Note: A sample mapping file, **Unix_SW_to_CACMDB.xml**, is provided out-of-the-box with the integration package.

5. Create an integration point

In UCMDB create an integration point. (For details, see "Integration Studio" in the *Data Flow Management section of the UCMDB Help*.)

Include the following details:

- a. Provide a name and description for the integration point.
- b. Provide the following details for the **CaCmdbPushAdapter** adapter:

Attribute	Description
Hostname/IP	The host name or IP address of the CA CMDB server.
Port	The port number of the CA CMDB server.
Credentials	The CA CMDB credential that was created in the prerequisites section above
Data Flow Probe	The name of the Data Flow Probe on which the integration runs.

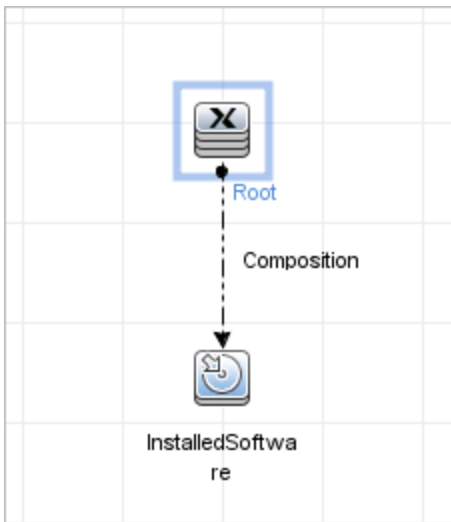
- c. Test the connection to the target CMDB server.
- d. Add a job definition to the integration point, selecting the queries to use to synchronize data

between UCMDB and CA CMDB. Define a synchronization schedule, if required.

- e. Invoke the ad hoc job, **Full Topology Sync**, for a full synchronization of the data.

Integration Query

The integration query, **Unix_SW_to_CACMDB**, is included with the CA CMDB integration package. This is an example of a query that can be used to query the CIs and relationships that must be pushed from UCMDB to CA CMDB. This query is accessible from UCMDB's Modeling Studio, among the query resources. For details, see "Modeling Studio Page" in the Modeling section of the UCMDB Help.



Troubleshooting and Limitations – CA CMDB Integration

This section describes troubleshooting and limitations related to UCMDB - CA CMDB integration.

- **Debug Mode**

To create an XML dump of the CIs and links being sent to the CA CMDB server for debug purposes, in **<UCMDB installation>\DataFlowProbe\runtime\probeManager\discoveryConfigFiles\CaCmdbPushAdapter\push.properties**, set the value of the **debugMode** property to **true** and restart the Data Flow Probe service.

This ensures that every time the integration is invoked, a set of XML files is created in the **<UCMDB installation>\DataFlowProbe\runtime\probeManager\discoveryResources\CaCmdbPushAdapter\work** directory. These files are

time-stamped and contain the CIs and links that UCMDB is trying to push to CA CMDB. This information can be helpful in debugging a problem with the integration:

- If data is not being sent from UCMDB, there is a problem on the UCMDB side.
- If data is not being processed by CA CMDB's GRLoader utility, there might be a reconciliation issue or some other issue on the CA CMDB side.
- During the export of more than 4,000 CIs (this is default UCMDB configuration) the export TQL result is divided into chunks. To make sure that chunks are created properly, you must configure the export TQL as follows: In the container node's properties, set the **Element Name** field to **Root**.

Chapter 6: CiscoWorks LAN Management Solution Integration

This chapter includes:

Overview	67
Supported Versions	67
Topology	67
How to Discover CiscoWorks LMS	68
CiscoWorks LMS Database Ports Job	70
Adapter	70
Trigger Query	70
Parameters	70
Network Devices from CiscoWorks LMS Job	71
Adapter	71
Trigger Query	71
Layer2 Topology from CiscoWorks LMS Job	71
Adapter	71
Trigger Query	72
Discovery Flow	72
CiscoWorks NetDevices Adapter	73
CiscoWorks Layer 2 Adapter	74
Discovery Mechanism	77
Troubleshooting and Limitations – CiscoWorks LAN Management Solution Integration	79

Overview

CiscoWorks LAN Management Solution (LMS) is a suite of management tools that simplify the configuration, administration, monitoring, and troubleshooting of Cisco networks.

This integration involves synchronizing devices, topology, and hierarchy of network infrastructure in UCMDB, and also synchronizes relationships between various hardware and logical network entities to enable end-to-end mapping of the data network infrastructure. The integration enables change management and impact analysis across all business services mapped in UCMDB, from a data network point of view.

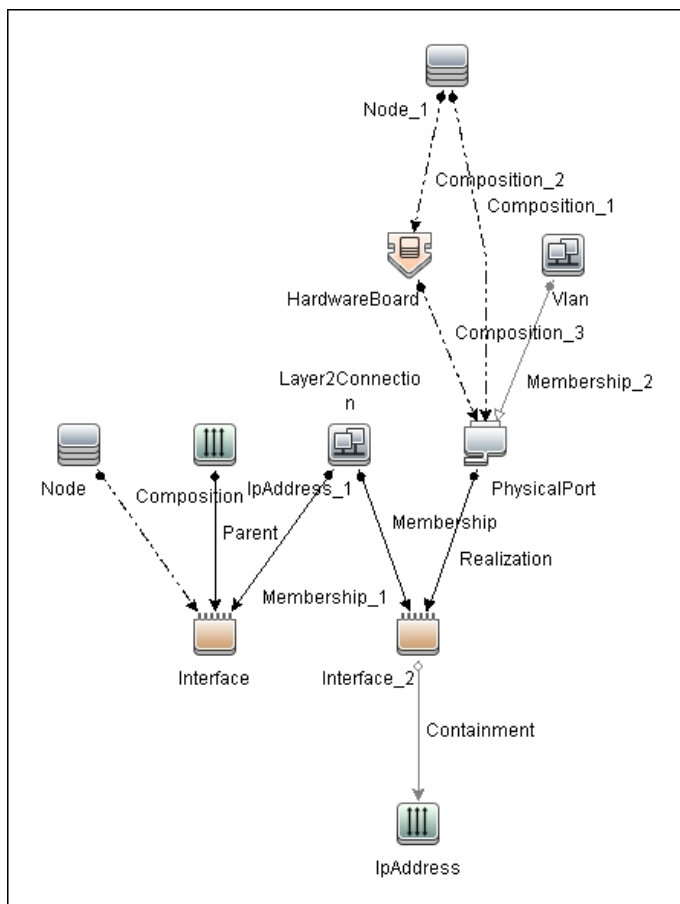
Supported Versions

This integration supports CiscoWorks LAN Management Solution Version 3.x.

Topology

The following image displays the CiscoWorks LAN Management Solution topology.

Note: For a list of discovered CITs, see ["Discovered CITs" on page 73](#).



How to Discover CiscoWorks LMS

1. Prerequisites - Set up protocol credentials

Add credentials for the Sybase database instances used by CiscoWorks LMS (**RMENGDB** and **ANIDB**) to the Generic DB (SQL) protocol.

For credential information, see "Supported Protocols" in the *UCMDB Discovery and Integrations Content Guide - Supported Content* document.

2. Run the discovery

- Discover IP addresses of the Sybase databases **RMENGDB** and **ANIDB** used by CiscoWorks LMS.
- Run the **CiscoWorks LMS Database Ports** job to discover the TCP ports at which the Sybase databases used by CiscoWorks LMS are listening.

- c. Create a new integration point, and use the **CiscoWorks NetDevices** adapter to discover network device information from CiscoWorks.
- d. Create a new integration point, and use the **CiscoWorks Layer 2** adapter to discover node (server) information from CiscoWorks.

Steps 2a and 2b are optional (although highly recommended - see note below) since CiscoWorks adapters are available in the Integration Studio, allowing manual creation of the necessary **IpAddress**, **Node** and **IpServiceEndpoint** CIs.

Note: The CiscoWorks Layer 2 job requires additional data about CIs created by the **CiscoWorks NetDevices** adapter and already in UCMDB. This information is provided by the Input Query, which contains CI types (**NetDevice** and **PhysicalPort**) that provide this data in addition to CI types required to identify the integration target (**IpServiceEndpoint**).

For this reason, it is highly recommended to execute steps 2a and 2b. If steps 2a and 2b are not executed, creating the integration target CIs (while creating an integration point using the **CiscoWorks Layer 2** adapter) requires the creation of **Node** and **PhysicalPort** CIs.

CiscoWorks LMS Database Ports Job

Adapter

This job uses the TCP Ports Discovery adapter

Trigger Query

- **Trigger CI:** IpAddress
- **Trigger Query:**



- **CI attribute conditions:**

CI	Attribute Value
IpAddress	NOT IP Probe Name is null

Parameters

Ports: 43443, 43455

Network Devices from CiscoWorks LMS Job

Adapter

This job uses the CiscoWorks_NetDevices adapter

Trigger Query

- **Trigger CI:** IpServiceEndpoint
- **Trigger Query:** CiscoWorks RME DB Port



- **CI attribute conditions:**

CI	Attribute Value
IpServiceEndPoint	NetworkPortNumber Equal 43455

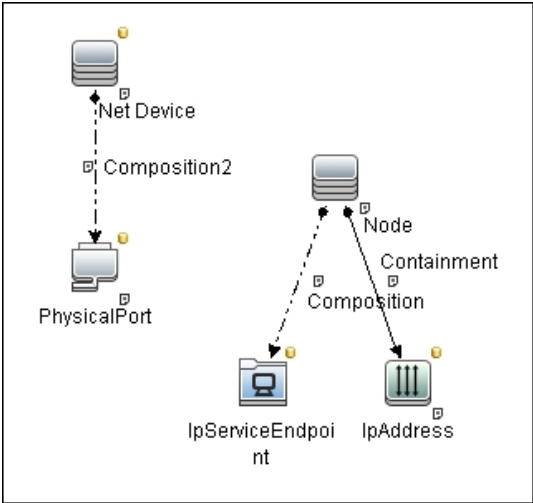
Layer2 Topology from CiscoWorks LMS Job

Adapter

This job uses the CiscoWorks_Layer2 adapter

Trigger Query

- **Trigger CI:** IpServiceEndpoint
- **Trigger Query:** CiscoWorks Campus DB Port



- **CI attribute conditions:**

CI	Attribute Value
IpServiceEndPoint	NetworkPortNumber Equal 43443
IpAddress	NOT IP Probe Name Is null
NetDevice	NOT Name Is null
PhysicalPort	NOT Name Is null AND NOT Port VLAN Is null AND NOT PortIndex Is null AND NOT Container Is null

Discovery Flow

Add IP addresses of the Sybase databases **RMENGDB** and **ANIDB** used by CiscoWorks LMS to a discovery probe range:

1. Range IPs by ICMP
2. CiscoWorks LMS Database Ports

3. CiscoWorks NetDevices
4. CiscoWorks Layer 2

CiscoWorks NetDevices Adapter

This section contains details about the adapter.

Input CIT

IpServiceEndpoint: the TCP port at which the **RMENGDB** Sybase instance is listening. (The default is 43455.)

Used Scripts

- ciscoworks_utils.py
- CiscoWorks_NetDevices.py

Discovered CITs

- Composition
- Containment
- HardwareBoard
- Interface
- IpAddress
- IpSubnet
- Layer2Connection
- Membership
- Node

- PhysicalPort
- Realization
- Vlan

Parameters

Parameter	Description
allowDnsLookup	If an IP address is not available for a node, setting this to true enables DNS lookup using the node name. Default: false.
ignoreNodesWithoutIP	If set to false , CIs for nodes without IPs are created with the storage system's internal ID as host_key . Note: this may result in duplicate nodes. Default: true.
rmeDbName	The name of the CiscoWorks Resource Manager Essentials database in Sybase. Default: rmengdb.
queryChunkSize	The number of network devices to query at a time. Default: 250.

CiscoWorks Layer 2 Adapter

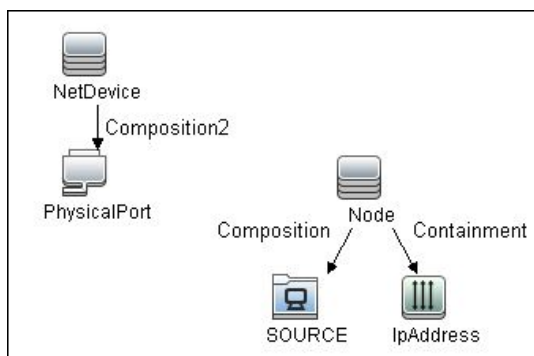
This section contains details about the adapter.

Input CIT

IpServiceEndpoint: the TCP port at which the **ANIDB** Sybase instance is listening. (The default is 43443.)

Input Query

CiscoWorks LMS Campus DB with PhysicalPorts



Node Conditions

Node Name	Condition
SOURCE (IpServiceEndPoint)	NetworkPortNumber Equal 43443
IpAddress	NOT IP Probe Name Is null
NetDevice	NOT Name Is null
PhysicalPort	NOT Name Is null AND NOT Port VLAN Is null AND NOT PortIndex Is null AND NOT Container Is null

Triggered CI Data

Name	Value
db_port	\${SOURCE.network_port_number}
ip_address	\${IpAddress.ip_address}
netdevice_cmdbid	\${NetDevice.global_id}
netdevice_name	\${NetDevice.name}
port_cmdbid	\${PhysicalPort.global_id}
port_container_cmdbid	\${PhysicalPort.root_container}

Name	Value
port_index	\${PhysicalPort.port_index}
port_name	\${PhysicalPort.name}
port_vlan	\${PhysicalPort.port_vlan}

Used Scripts

- ciscoworks_utils.py
- CiscoWorks_Layer2.py

Discovered CITs

- Composition
- Containment
- HardwareBoard
- Interface
- IpAddress
- IpSubnet
- Layer2Connection
- Membership
- Node
- PhysicalPort
- Realization
- Vlan

Parameters

Parameter	Description
allowDnsLookup	If an IP address is not available for a node, setting this to true enables DNS lookup using the node name. Default: false.
ignoreNodesWithoutIP	If set to false , CIs for nodes without IPs are created with the storage system's internal ID as host_key . Note: this may result in duplicate nodes. Default: true.
campusDbName	The name of the CiscoWorks Campus database in Sybase. Default: anidb.
queryChunkSize	The number of nodes to query at a time. Default: 1,000.

Discovery Mechanism

The adapters in this package connect to the Sybase databases used by CiscoWorks LMS using JDBC, and run SQL queries to retrieve information. The Sybase database instances are used as part of the trigger for jobs in this package. This allows the jobs to be included in UCMDB's spiral discovery schedule.

The package includes the following two adapters:

- **CiscoWorks NetDevices**
- **CiscoWorks Layer 2**

CiscoWorks NetDevices triggers off the CiscoWorks Resource Manager Essentials database, and retrieves network devices, VLAN and layer two infrastructure from it.

CiscoWorks Layer 2 triggers off the CiscoWorks Campus Manager database, and retrieves nodes (servers). It associates them with VLANs and layer two infrastructure retrieved by **CiscoWorks NetDevices**.

Database queries executed by this package on the CiscoWorks databases are as follows:

Note: The following query is used by the **CiscoWorks NetDevices** and **CiscoWorks Layer 2** adapters on the **RMENGDB** and **ANIDB** database instances

Get the database name to verify that queries are run on the correct database:

```
SELECT db_name()
```

Note: The following queries are used by the **CiscoWorks NetDevices** adapter on the **RMENGDB** database instance

Get a count of the number of network devices in the database (This is required to determine the number of chunks to query. For details on chunking, see ["Parameters" on page 74.](#))

```
SELECT COUNT(1) FROM lmsdatagrpf.NETWORK_DEVICES
```

Get information on network devices managed by CiscoWorks LMS

```
SELECT netdevices.Device_Id,  
deviceState.NetworkElementID, netdevices.Device_Display_Name,  
netdevices.Host_Name, netdevices.Device_Category,  
netdevices.Device_Model, netdevices.Management_IPAddress,  
deviceState.Global_State  
FROM lmsdatagrpf.NETWORK_DEVICES netdevices JOIN dba.DM_Dev_State  
deviceState ON netdevices.Device_Id=deviceState.DCR_ID
```

Get additional details on each network device.

```
SELECT * FROM dba.PhysicalTypeEnum  
  
SELECT ne.ElementName, ne.ReportedHostName, ne.DNSDomainName, ne.Description,  
ne.PrimaryOwnerContact, ne.ElementLocation, os.OSName, os.Version, os.ROMVersion,  
pe.Manufacturer, pe.SerialNumber  
FROM dba.OperatingSystem os, dba.PhysicalElement pe, dba.networkelement ne  
WHERE os.NetworkElementID=<networkDeviceID> AND  
ne.NetworkElementID=<networkDeviceID> AND pe.NetworkElementID=<networkDeviceID> AND  
LOWER(pe.PhysicalType)=<physicalType> AND pe.PhysicalElementId IN (1, 2)
```

Get port and VLAN information for each network device.

```
SELECT phyPort.PhysicalPortID, phyPort.SNMPPhysicalIndex, phyPort.ParentRelPos,  
port.PORT_NAME, port.PORT_DESC, port.PORT_DUPLEX_MODE, port.PORT_TYPE, port.PORT_  
SPEED, port.VLAN_NAME, port.VLANID, interface.EndpointID, interface.Description,  
interface.Alias, interface.MediaAccessAddress  
FROM lmsdatagrpf.PORT_INVENTORY port JOIN dba.PhysicalPort phyPort ON port.PORT_  
NAME=phyPort.PortName JOIN dba.IFEntryEndpoint interface ON port.PORT_  
NAME=interface.EndpointName  
WHERE phyPort.NetworkElementID=<networkDeviceID> AND  
interface.NetworkElementID=<networkDeviceID> AND port.DEVICE_ID=<networkDeviceID>
```

AND phyPort.PortName=port.PORT_NAME

Get IP Address details for each network device.

```
SELECT IPAddress, SubnetMask FROM dba.IPProTOCOLEndPoint WHERE  
NetworkElementId=<networkDeviceID>
```

Get information on modules in each network device.

```
SELECT MODULE_NAME, SW_VERSION, FW_VERSION, SLOT_NUMBER FROM lmsdatagrpf.MODULE_  
INVENTORY WHERE DEVICE_ID=<networkDeviceID>
```

Note: The following queries are used by the **CiscoWorks Layer 2** adapter on the **ANIDB** database instance.

Get a count of the number of nodes (servers) in the database (This is required to determine if chunking is required. See ["Parameters" on page 77.](#))

```
SELECT COUNT(1) FROM lmsdatagrpf.End_Hosts
```

Get information on nodes managed by or known to CiscoWorks LMS.

```
SELECT HostName, DeviceName, Device, MACAddress, IPAddress,  
SubnetMask, Port, PortName, VLAN, VlanId, associatedRouters  
FROM lmsdatagrpf.End_Hosts  
WHERE HostName IS NOT NULL AND NOT HostName='' AND IPAddress IS NOT  
NULL AND NOT IPAddress=''
```

Troubleshooting and Limitations – CiscoWorks LAN Management Solution Integration

If there is a database connection failure, copy the Sybase JDBC driver (**jconnectnn.jar** or similar JAR file) from the Sybase system to the **<DataFlowProbe_**

Home>\runtime\probeManager\discoveryResources\db\sybase directory on the DFM probe file system.

If the database connection failure occurs after the driver is copied, it may be necessary to change the driver classes in **globalSettings.xml** from:

```
<Sybase>com.sybase.jdbc.SybDriver</Sybase>
```

to

```
<Sybase>com.sybase.jdbc3.SybDriver</Sybase>
```

Chapter 7: CyberArk Integration

CyberArk is a product that implements an external password vault. CyberArk Enterprise Password Vault, part of the CyberArk Privileged Account Security Solution, enables organizations to secure, manage and track the use of privileged credentials whether on-premise or in the cloud, across operating systems, databases, applications, hypervisors, network devices and more.

The integration between UCMDB and CyberArk Enterprise Password Vault allows Universal Discovery administrators to configure credentials for supported Universal Discovery protocols, which enables administrators to manage the credentials in a secure and easy way.

Instead of storing the passwords themselves in UCMDB/UD, this integration involves storing only references (in the CyberArk Enterprise Password Vault) to the passwords, and retrieving the passwords when they are needed from the vault using the stored references.

Note: As the CyberArk integration enables the discovery of content but does not actually perform data collection, no MDR integration license is required for the use of this capability.

How the CyberArk Integration Works

The CyberArk integration enables UCMDB/UD to retrieve usernames and passwords from the CyberArk Enterprise Password Vault as follows:

1. Administrators to create a Safe, Application, and Account on the CyberArk Server, including username, password, and unique reference ID.
2. Universal Discovery administrators to create a credential on UCMDB Server, using the same CyberArk Safe, Application, and Account values created in step 1 as reference ID in the following format: <Safe_Name>\<Folder_Path>\<Reference_ID>
3. The CyberArk integration synchronizes the CyberArk references to Data Flow Probes. No password information contained.
4. Universal Discovery administrators to run discovery jobs using the unique referenceID to retrieve username and password from CyberArk.

For more details about the integration, see *Integrating UCMDB with CyberArk Enterprise Password Vault* in the *Data Flow Management section of the UCMDB Help* for version 10.22 or later.

Chapter 8: EMC Control Center (ECC) Integration

This chapter includes:

Overview	82
Supported Versions	82
Topology	82
How to Run the ECC/UCMDB Integration Job	83
ECC Integration Job	85
Views	89
Impact Analysis Rules	94
Reports	96

Overview

Integration between ECC and DFM involves synchronizing devices, topology, and hierarchy of storage infrastructure in the UCMDB database (CMDDB). This enables Change Management and Impact Analysis across all business services mapped in UCMDB from a storage point of view.

DFM initiates discovery on the ECC database. Synchronized Configuration Items (CIs) include Storage Arrays, Fibre Channel Switches, Hosts (Servers), Storage Fabrics, Storage Zones, Logical Volumes, Host Bus Adapters, Storage Controllers, and Fibre Channel Ports. The integration also synchronizes physical relationships between hardware, and logical relationships between Logical Volumes and hardware devices, to enable end-to-end mapping of the storage infrastructure.

You integrate ECC with UCMDB using Data Flow Management.

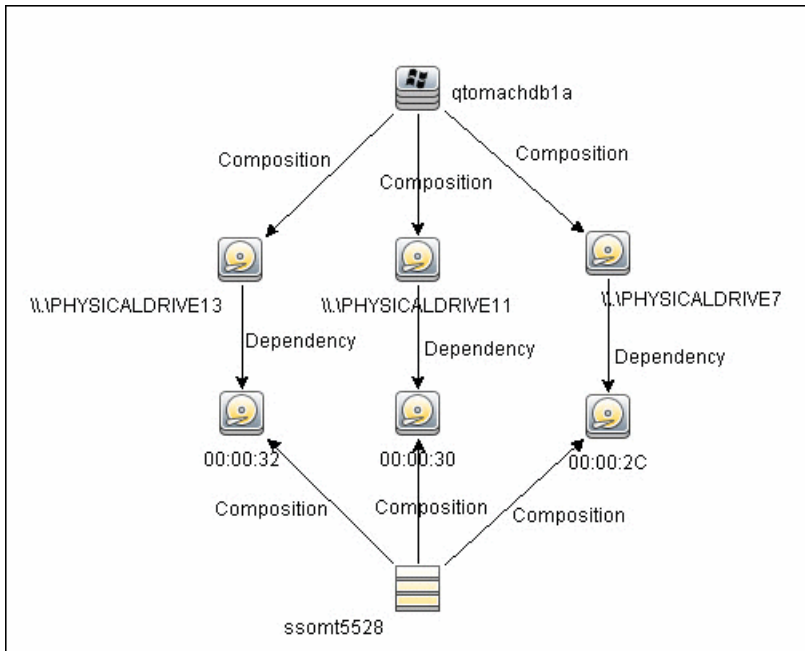
The integration includes the **ECC_Integration.zip** package, which contains the trigger TQL, DFM script, adapter, and job for ECC integration.

Supported Versions

Target Platform	OS Platform	DFM Protocol	ECC Version
EMC Control Center	All	Generic DB (SQL) over JDBC, SSL optional	6.0 and 6.1

Topology

The following diagram illustrates the storage topology and shows the relationships between logical volumes on a storage array and those on servers:



How to Run the ECC/UCMDB Integration Job

This task includes the steps to run the ECC/UCMDB integration job.

Note: For details on running an integration job, see "Integration Studio" in the *Data Flow Management section of the UCMDB Help*.

In the Integration Studio, create a new integration point.

1. Provide a name and description for the integration point.
2. Under **Integration Properties > Adapter**, select the **EMC Control Center** adapter.
3. Under **Adapter Properties > Data Flow Probe**, select the **Data Flow Probe**.
4. Under **Adapter Properties > Trigger CI instance** select:
 - a. **Select Existing CI** (if you have a valid, existing CI). The **Select Existing CI** pane appears. Select the CI or
 - b. **Create New CI** (if you need to create a new CI). The **Topology CI Creation Wizard** appears. Complete the creation of the CI using the Wizard.

Note: For details on the Topology CI Creation Wizard, see "Topology CI Creation Wizard" in the *Data Flow Management section of the UCMDB Help*.

5. Verify the credentials for the chosen CI instance. Right-click on **Trigger CI instance** and select **Actions > Edit Credentials Information**.
6. Save the integration point.
7. Run the job.

Tip: You can include the ECC job in the DFM schedule. For details, see "New Integration Job/Edit Integration Job Dialog Box" in the *Data Flow Management section of the UCMDB Help*.

ECC Integration Job

This section includes:

- ["ECC Integration Mechanism" below](#)
- ["Trigger Query" on page 87](#)
- ["Adapter" on page 88](#)
- ["Discovered CITs and Relationships" on page 89](#)

ECC Integration Mechanism

The following workflow explains how the **ECC Integration by SQL** job discovers the storage topology of ECC. The job:

1. Connects to the ECC Oracle database instance using credentials from the Generic DB Protocol (SQL). For details, see ["How to Run the ECC/UCMDB Integration Job" on page 83](#).
2. Queries for fibre channel switches and ports on each switch and creates **Fibre Channel Switch** CIs:

```
SELECT switch.st_id, switch.st_sn, switch.st_alias, switch.st_model, switch.st_
version, switch.st_vendor, switch.sw_managementurl, switch.sw_domain,
switch.sw_portcount, switch.sw_portcount_free FROM stssys.sts_switch_list
switch WHERE LOWER(switch.sw_principal) = 'true'
```

3. Queries for fibre channel adapters and ports on each Fibre Channel Switch and creates **Fibre Channel HBA** and **Fibre Channel Port** CIs:

```
SELECT port.port_id, port.port_number, port.port_type, port.adport_alias,
port.port_wwn, port.port_status, port.conn_port_wwn FROM stssys.sts_switch_port
port WHERE port.st_id = switch.st_id from above query
```

4. Queries for storage arrays and creates **Storage Array** CIs:

```
SELECT array.st_id, array.st_sn, array.st_alias, array.st_type, array.st_model,
array.st_vendor, array.st_microcode, array.sy_microcode_patch, array.sy_
microcode_patchdate FROM stssys.sts_array_list array
```

5. Queries for Fibre Channel ports, Fibre Channel host bus adapters (HBA), and logical volumes on each storage array, and creates **Fibre Channel Port**, **Fibre Channel Port HBA**, and **Logical Volume** CIs:

```

SELECT port.port_id, port.port_number, port.port_type, port.adport_alias,
port.port_wwn, port.port_status FROM stssys.sts_array_port port WHERE port.st_
id = array.st_id from above query
SELECT hba.port_id, hba.ad_id, hba.ad_name FROM stssys.sts_array_port hba WHERE
hba.st_id = array.st_id from above query
SELECT logicalVolume.sd_id, logicalVolume.sd_name, logicalVolume.sd_alias,
logicalVolume.sd_size, logicalVolume.sd_type FROM stssys.sts_array_device
logicalVolume WHERE logicalVolume.st_id = array.st_id from above query

```

6. Queries for hosts/servers and creates appropriate **Computer**, **Windows**, or **Unix** CIs. Results of this query are used to create host resource CIs, such as **CPU**, if this information is available:

```

SELECT host.host_id, host.host_name, host.host_alias, host.host_domain,
host.host_model, host.host_ip, host.host_vendorname, host.host_cpucount,
host.host_installedmemory, host.host_os, host.host_osversion, host.host_
oslevel, host.host_osclass FROM stssys.sts_host_list host

```

7. Queries for Fibre Channel ports, Fibre Channel host bus adapters (HBA), and logical volumes on each host/server and creates **Fibre Channel Port**, **Fibre Channel Port HBA**, and **Logical Volume** CIs:

```

SELECT port.port_id, port.port_number, port.adport_alias, port.port_wwn FROM
stssys.sts_host_hba port WHERE port.host_id = host.host_id from above query
SELECT hba.ad_id, hba.ad_name, hba.fibread_nodewwn, hba.ad_vendor, hba.ad_
revision, hba.ad_model, hba.port_id, hba.ad_driver_rev FROM stssys.sts_host_hba
hba WHERE hba.host_id = host.host_id from above query
SELECT logicalVolume.hd_id, logicalVolume.hd_name, logicalVolume.hd_type,
logicalVolume.hd_total FROM stssys.sts_host_device logicalVolume WHERE
logicalVolume.hd_id IS NOT NULL AND logicalvolume.arrayjbod_type = 'Array' AND
logicalVolume.host_id = host.host_id from above query

```

8. Queries for logical volume mapping between logical volumes on hosts/servers and logical volumes on storage arrays, and adds **Dependency** relationships between hosts/servers and storage arrays:

```

SELECT sd_id FROM stssys.sts_host_shareddevice WHERE hd_id = logicalvolume.hd_
id from above query

```

9. Queries for paths between hosts/servers and storage arrays and adds **Fibre Channel Connect** relationships between respective hosts/servers, switches, and storage arrays:

```

SELECT port.port_wwn, port.conn_port_wwn FROM stssys.sts_array_port_connection
port WHERE port.port_wwn IS NOT NULL AND port.conn_port_wwn IS NOT NULL
SELECT port.port_wwn, port.conn_port_wwn FROM stssys.sts_switch_port port WHERE
port.port_wwn IS NOT NULL AND port.conn_port_wwn IS NOT NULL

```

Trigger Query

Trigger CI: ECC Oracle database

Adapter

The following table gives details of the adapter parameters.

Parameter	Description
allowDNSLookup	<p>If a node in the ECC database does not have an IP address but has a DNS name, it is possible to resolve the IP address by the DNS name.</p> <ul style="list-style-type: none">• True: If a node does not have an IP address, an attempt is made to resolve the IP address by DNS name (if a DNS name is available). <p>Default: False</p>
ignoreNodesWithoutIP	<p>Defines whether or not nodes in ECC without IP addresses should be pulled into UCMDB.</p> <ul style="list-style-type: none">• True. Nodes without IPs are ignored.• False. A Node CI is created with an ECC ID as the node key attribute. <p>Note: Setting this parameter to False may result in duplicate CIs in the CMDB.</p> <p>Default: True</p>

Discovered CITs and Relationships

- CPU
- Containment
- Composition (link)
- Dependency (link)
- Fibre Channel Connect (link)
- Fibre Channel HBA
- Fibre Channel Port
- Fibre Channel Switch
- Node
- IpAddress
- Logical Volume
- Membership (link)
- Storage Array
- Storage Processor
- Unix
- Windows

Views

The **Storage_Basic** package contains views that display common storage topologies. These are basic views that can be customized to suit the integrated ECC applications.

To access the Storage_Basic package, go to **Administration > Package Manager**. For details, see "Package Manager" in the *Administer section of the UCMDB Help*.

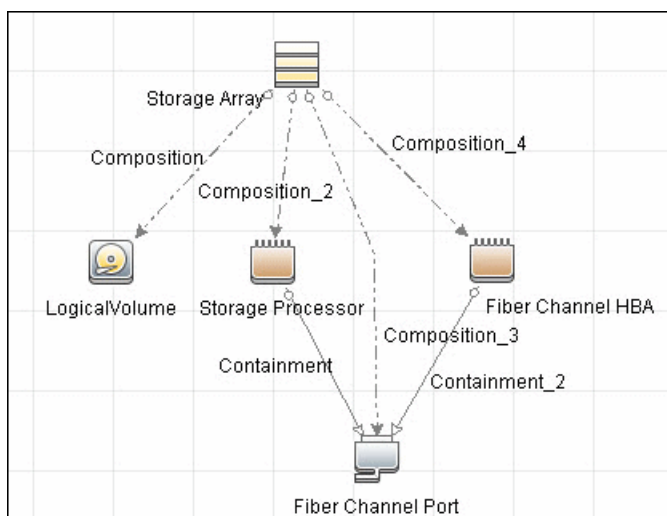
This section includes:

- ["Storage Array Details" below](#)
- ["FC Switch Details" on the next page](#)
- ["Host Storage Details" on page 92](#)
- ["SAN Topology" on page 93](#)
- ["Storage Topology" on page 93](#)

Storage Array Details

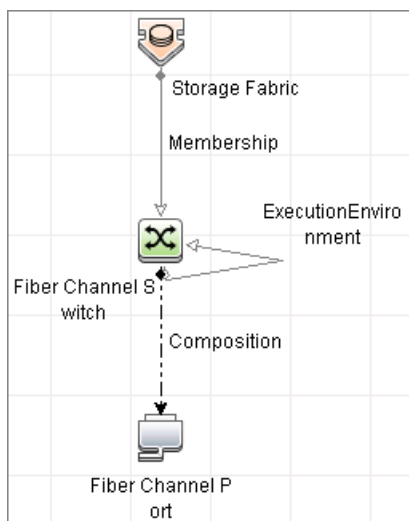
This view shows a Storage Array and its components including Logical Volumes, HBAs, Storage Processors, and Fibre Channel Ports. The view shows each component under its container Storage Array and groups Logical Volumes by CI Type.

Storage Array does not require all components in this view to be functional. Composition links stemming from the Storage Array have a cardinality of zero-to-many. The view may show Storage Arrays even when there are no Logical Volumes or Storage Processors.



FC Switch Details

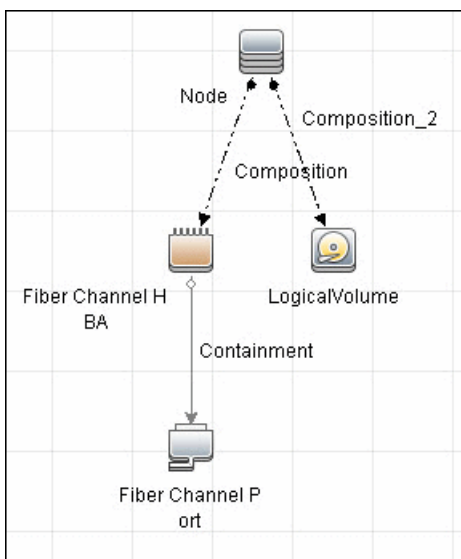
This view shows a Fibre Channel Switch and all connected Fibre Channel Ports.



Note: Although shown in the preceding graphic, the ECC job does not discover Storage Fabrics. The view represented by this query is populated without Storage Fabrics.

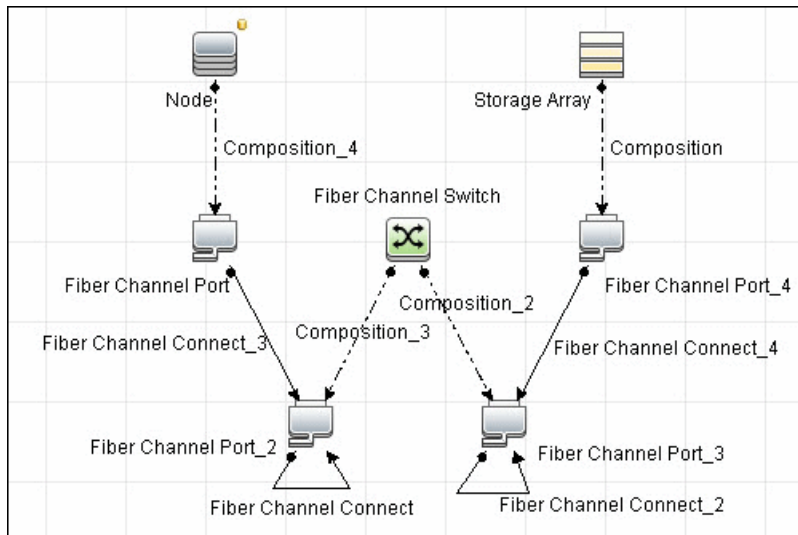
Host Storage Details

This view shows only Hosts that contain a Fibre Channel HBA or a Logical Volume. This keeps the view storage-specific and prevents hosts discovered by other DFM jobs from being included in the view.



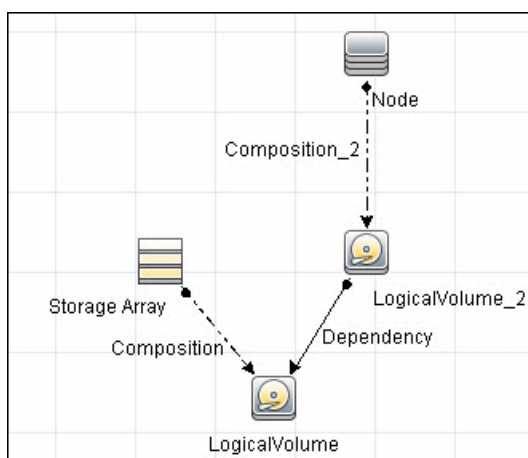
SAN Topology

This view maps physical connections between Storage Arrays, Fibre Channel Switches, and Hosts. The view shows Fibre Channel Ports below their containers. The view groups the Fibre Channel Connect relationship CIT to prevent multiple relationships between the same nodes from appearing in the top layer.



Storage Topology

This view maps logical dependencies between Logical Volumes on Hosts and Logical Volumes on Storage Arrays. There is no folding in this view.



Impact Analysis Rules

The **Storage_Basic** package contains basic impact analysis rules to enable impact analysis and root cause analysis in UCMDB. These impact analysis rules are templates for more complex rules that you can define based on business needs.

All impact analysis rules fully propagate both Change and Operation events. For details on impact analysis, see "Impact Analysis Manager Page" and "Impact Analysis Manager Overview" in the *Modeling section of the UCMDB Help*.

To access the Storage_Basic package: **Administration > Package Manager**. For details, see "Package Manager" in the *Administer section of the UCMDB Help*.

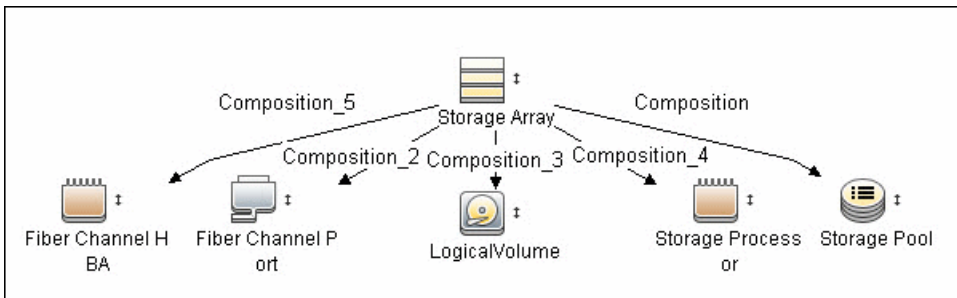
Note: Impact analysis events are not propagated to Fibre Channel Ports for performance reasons.

This section includes:

- ["Storage Array Devices to Storage Array" below](#)
- ["Host Devices to Host" on the next page](#)
- ["Logical Volume to Logical Volume" on the next page](#)
- ["FC Switch Devices to FC Switch" on the next page](#)
- ["FC Port to FC Port" on page 96](#)

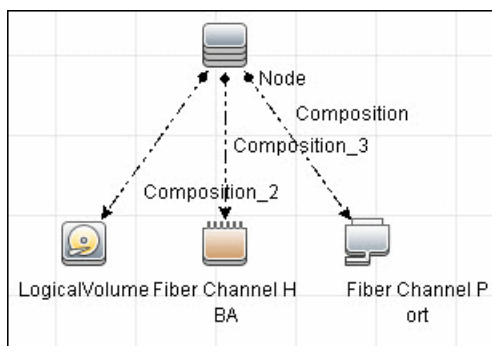
Storage Array Devices to Storage Array

This impact analysis rule propagates events between Logical Volumes, Storage Processors, Fibre Channel HBAs, and Storage Arrays.



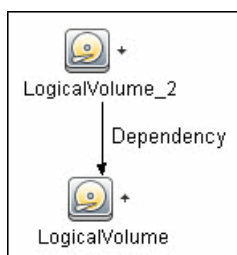
Host Devices to Host

This impact analysis rule propagates events between Fibre Channel HBAs and Hosts, and Logical Volumes on the Host.



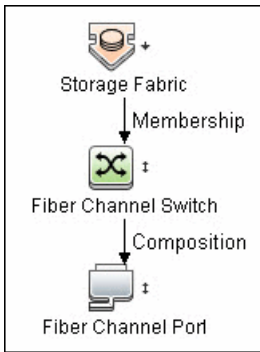
Logical Volume to Logical Volume

This impact analysis rule propagates events on a Logical Volume contained in a Storage Array to the dependent Logical Volume on the Host.



FC Switch Devices to FC Switch

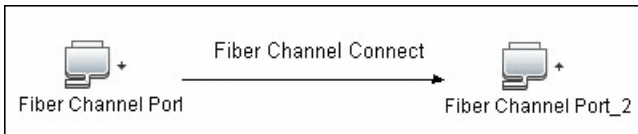
This impact analysis rule propagates events from a Fibre Channel Port to and from a Switch. The event is also propagated to the associated Storage Fabric.



Note: Although shown in the preceding graphic, the ECC job does not discover Storage Fabrics. The rule represented by this query is used without Storage Fabrics.

FC Port to FC Port

This rule propagates events on a Fibre Channel Port to another connected Channel Port.



Example Scenario of HBA Crashing on a Storage Array

- The event propagates from the HBA to the Storage Array and the Logical Volumes on the Array because of the Storage Devices to Storage Array rule.
- The impact analysis event on the Logical Volume then propagates to other dependent Logical Volumes through the Logical Volume to Logical Volume rule.
- Hosts using those dependent Logical volumes see the event next because of the Host Devices to Host rule.
- Depending on business needs, you define impact analysis rules to propagate events from these hosts to applications, business services, lines of business, and so on. This enables end-to-end mapping and impact analysis using UCMDB.

Reports

The **Storage_Basic** package contains basic reports that can be customized to suit the integrated ECC applications.

In addition to the system reports, Change Monitoring and Asset Data parameters are set on each CIT in this package, to enable Change and Asset Reports in UCMDB.

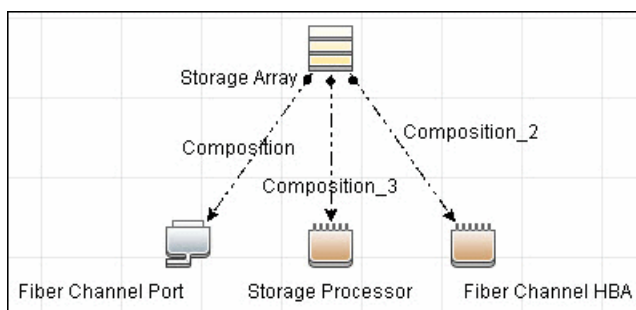
To access the Storage_Basic package: **Administration > Package Manager**. For details, see "Package Manager" in the *Administer section of the UCMDB Help*.

This section includes:

- ["Storage Array Configuration" below](#)
- ["Host Configuration" on the next page](#)
- ["Storage Array Dependency" on the next page](#)
- ["Host Storage Dependency" on page 99](#)

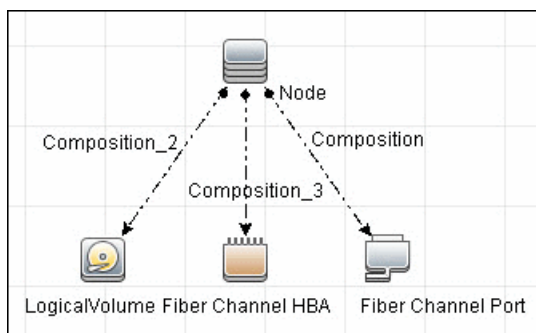
Storage Array Configuration

This report shows detailed information on Storage Arrays and its sub-components including Fibre Channel Ports, Fibre Channel Arrays, and Storage Processors. The report lists Storage Arrays with sub-components as children of the Array.



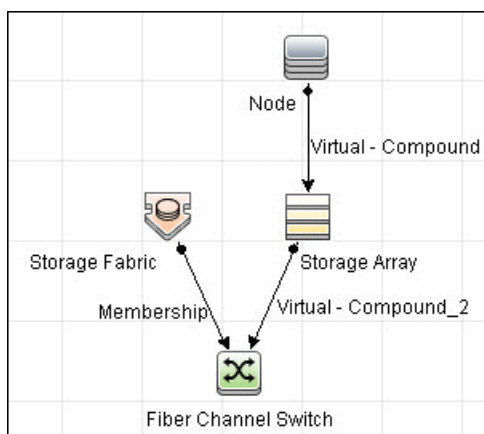
Host Configuration

This report shows detailed information on hosts that contain one or more Fibre Channel HBAs, Fibre Channel Ports, or Logical volumes. The report lists hosts with sub-components as children of the host.



Storage Array Dependency

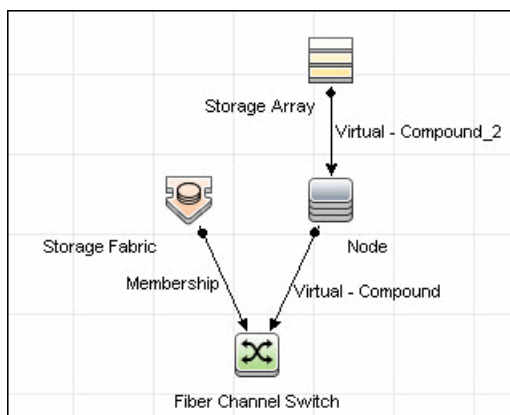
This report maps dependencies on a Storage Array. The report also displays information on switches connected to it.



Note: Although shown in the preceding graphic, the ECC job does not discover Storage Fabrics. The report represented by this query is populated without Storage Fabrics.

Host Storage Dependency

This report shows detailed information on storage infrastructure dependencies of a Host. The report lists hosts and dependent components.



Note: Although shown in the preceding graphic, the ECC job does not discover Storage Fabrics. The report represented by this query is populated without Storage Fabrics.

Chapter 9: HPE OneView Pull Integration

This chapter includes:

Overview	100
Discovery Mechanism	100
Supported Versions	100
Create an Integration Point between HPE OneView and UCMDB	101
Data Mappings	103
Input CITs	104
Used Scripts	104
Discovered CITs	104
Global Configuration File	105
Topology Map	105
How to Integrate with OneView 1.x	106
Troubleshooting and Limitations – OneView Pull Integration	108

Overview

HPE OneView software provides management tools for converged infrastructure and are used by system administrators to provision, control, and manage software-defined data center components.

This integration adapter pulls data from HPE OneView, and then the data is sent to UCMDB.


Discovery Mechanism




A REST client on the Data Flow Probe connects to the HPE OneView REST API to fetch data.



Supported Versions

This integration adapter supports HPE OneView versions 2.x, 3.0, and the REST API version 200. To integrate with HPE OneView 1.x, see ["How to Integrate with OneView 1.x" on page 106](#).

Create an Integration Point between HPE OneView and UCMDB

1. Log on to UCMDB as an administrator.
2. Navigate to **Data Flow Management > Integration Studio**. A list of existing integration points is displayed.
3. Click **New Integration Point** . The New Integration Point dialog box opens.
4. Complete the **Integration Properties** and **Adapter Properties** fields as shown in the following table:


Field (*Required)	Description
Integration Properties section	
*Integration Name	Type the name (unique key) of the integration point.
Integration Description	Type a description of the current integration point.
*Adapter	Click  and then select HP Software Products / Micro Focus Products > OneView Integration .
*Is Integration Activated?	Select this option to indicate the integration point is active.
Adapter Properties section	
*Credentials ID	<p>Click the Select Credential Id  button. The Generic Protocol is already selected in the Protocol pane and in the Credentials list.</p> <p>Note:</p> <ul style="list-style-type: none"> Other credentials are not supported. Ensure that the account has Read access to OneView. <p>To configure the credential for this integration point,</p> <ol style="list-style-type: none"> Ensure the Generic Protocol is selected, and then click . The Generic Protocols Parameters dialog box opens. Provide values for the following fields and click OK: <ul style="list-style-type: none"> Network Scope: Use the default value ALL. User Label: Type a label for the credential.


Field (*Required)	Description
	<ul style="list-style-type: none"> • User Name: Provide the user name for the OneView account. • Password: Click  and provide the password for the OneView account. <p>c. Click OK twice.</p>
*OneView URL	The URL of OneView.
Trust All SSL Certificates	<p>Specifies whether to trust all SSL certificates when the OneView server does not have a valid certificate.</p> <p>Type one of the following values:</p> <p>True. Trust all SSL certificates.</p> <p>False. Do not trust all SSL certificates.</p> <p>Default value: True</p>
*Data Flow Probe	<p>The name of the Data Flow Probe/Integration service used to execute the synchronization from.</p> <p>Select IntegrationService for this integration.</p> <p>Note: If the IntegrationService option does not exist, consult with your UCMDB administrator for the best selection for your requirements.</p>
Trigger CI Instance	<p>Click  and select one of the following options from the drop-down menu:</p> <p>Select Existing CI. Select this option only if you want to select a valid, existing CI. The Select Existing CI pane appears.</p> <p>Create New CI. Select this option if you want to create a new CI. The Topology CI Creation Wizard appears. Complete the creation of the CI using the Wizard.</p> <p>Note: For details on the Topology CI Creation Wizard, see "Topology CI Creation Wizard" in the <i>Data Flow Management section of the UCMDB Help</i>.</p>

5. Click **OK**.

The integration point is created and its details are displayed.

Note:

- Your settings are not saved to the server until you click **OK**.
- You can edit this integration point by selecting the Integration Point name in the Integration Point pane and then clicking .

6. (Optional) To start the job immediately, click  to perform a full synchronization.

Note:

- The first Full Synchronization may take a while to complete.
- If you do not want the job to begin immediately, the job will be scheduled to run.

To see results of the job, perform the following:

1. In Integration Studio, in the Integration Job pane, right-click the OneView job.
2. In the drop-down menu, click **Show Results of Job**.

Note:

- The results of the previous job run that are stored in the Probe's database are displayed.
- If the job has not yet run, the following message is displayed: "no results found on selected job".
- Results are only displayed when jobs are run successfully.

Data Mappings

The following entities are mapped from HPE OneView to UCMDB:

HPE OneView	UCMDB
Datacenter	Datacenter
Rack	Rack
Enclosure	Enclosure
Server-hardware	Node
Interconnect	Fiber Channel Switch
PDU	Power Distribution Unit
Slot	Hardware Board

HPE OneView	UCMDB
Port	Physical Port

Input CITs

Discovery Probe Gateway

Used Scripts

oneview_pull.py

oneview_connection_data_manager.py

oneview_client.py

oneview_rest.py

oneview_mapping_file_manager.py

oneview_mapping_implementation.py

oneview_mapping_interfaces.py

oneview_validators.py

oneview_decorators.py

Discovered CITs

DataCenter

Rack

Enclosure

Node

Fibre Channel Switch

Hardware Board

Physical Port

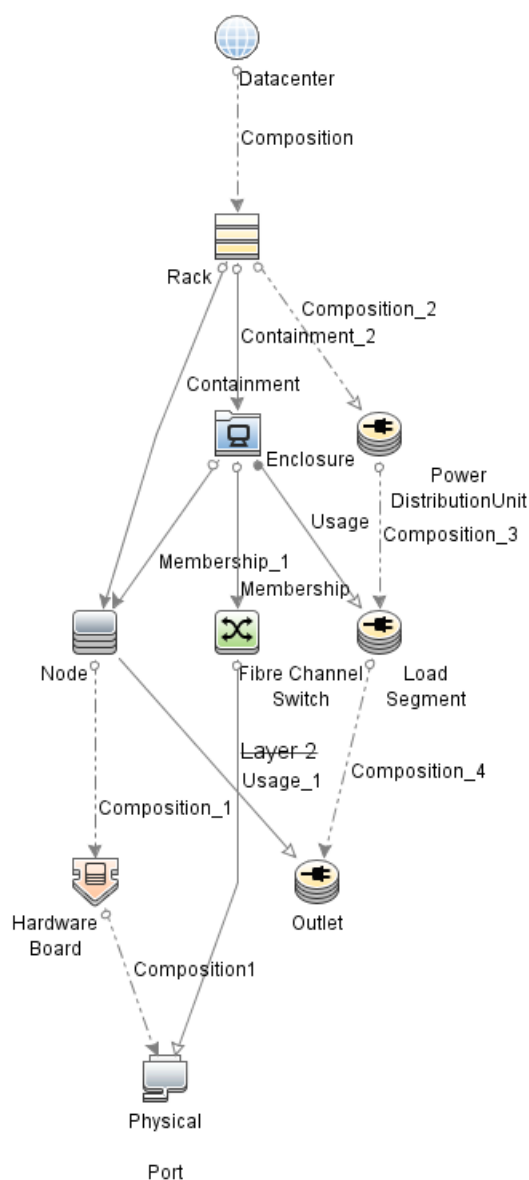
Power Distribution Unit

Global Configuration File

HPOneView/oneview.xml

Topology Map

Note: For a list of Discovered CITs, see ["Discovered CITs" on the previous page](#)



How to Integrate with OneView 1.x

To integrate with OneView 1.x, do the following:

- Change the source attribute powerState of the CI type interconnect to powerStatus in the configuration file **oneview.xml** as follows:

```
<source_ci_type name="interconnect" query="/rest/interconnects"
```

```

base="members">
  <target_ci_type name="fcswitch">
  ...
    <target_attribute name="fcswitch_status">
      <map type="direct" source_attribute="powerStatus"/>
    </target_attribute>
  ...
  </target_ci_type>
</source_ci_type>

```

- Replace `self._apiVersion = 200` with `self._apiVersion = 4` in **oneview_client.py** as follows:

```

class OneViewClient(object):
    def __init__(self, url, trustAllCerts=True):
        self.url = url

        self._apiVersion = 4

        self._headers = {'X-API-Version': self._apiVersion,
                        'Accept': 'application/json',
                        'Content-Type': 'application/json'}
    }

```

Note: For OneView versions 2.x and 3.0, the default settings are as follows:

- `source_attribute = PowerState` in **oneview.xml**
- `self._apiVersion = 200` in **oneview_client.py**

The table lists the supported attributes for the fibre channel switches obtained from the **/rest/interconnects** API for different API and OneView versions:

API Version	Source Attribute Name	Version 1 Support	Version 2 Support	Version 3 Support
4	powerStatus	Yes	No	No
4	powerState	No	Yes	No
200	powerStatus	No	No	No
200	PowerState	No	Yes	Yes

Troubleshooting and Limitations – OneView Pull Integration

- **Problem:** You receive the following error message: "Failed to pull data from OneView".

Solution: View the communication log.

Chapter 10: HPE Systems Insight Manager (SIM) Integration

This chapter includes:

Overview	110
Supported Versions	110
SIM Integration Mechanism	110
How to Discover HPE SIM Data Center Infrastructure	113
SIM WebService Ports Job	116
SIM Integration by WebServices Job	117
Instance Views	118
Troubleshooting and Limitations – SIM Integration	120

Overview

Universal CMDB (UCMDB) can discover data center infrastructure information stored in an HPE Systems Insight Manager (SIM) system. Integration involves synchronizing devices, topology, and the hierarchy of a data center infrastructure in the UCMDB database (CMDB). This enables change management and impact analysis across all business services mapped in UCMDB, from an infrastructure point of view.

UCMDB initiates discovery on the HPE SIM server through Web service calls. Synchronized configuration items (CIs) include nodes such as Windows, and UNIX servers, network devices, printers, clusters, cellular/partitioned systems, blade enclosures, and racks. Some server components, for example, CPU, are also synchronized. The integration also synchronizes relationships between blade servers and blade enclosures, virtual machines, physical servers, and so on. The synchronization uses an XML-based mapping that dynamically changes synchronized CIs and attributes without requiring a code change.

For details on nodes and attributes in HPE SIM, refer to the Database tables section of the *HPE SIM Technical Reference* guide.

Supported Versions

This integration solution supports HPE SIM versions 5.1, 5.2, 5.3, 6.0, 6.1, 6.2, 6.3, 7.0, 7.1, 7.2, 7.3, 7.4, 7.5, and 7.6.

SIM Integration Mechanism

DFM uses the HPE SIM Web service API to retrieve node information from the HPE SIM database. DFM also enables you to specify extended attributes that should be retrieved for each node.

HPE SIM represents hosts (blade enclosures, racks, servers, and so on) as Nodes; UCMDB has separate CITs for each such host. To represent hosts correctly in UCMDB, a two-level mapping is used, to enable integration customization without code changes. This makes the integration completely customizable and dynamic.

For details on jobs, see "Module/Job-Based Discovery" in the *Data Flow Management section of the UCMDB Help*.

The following sections describe the levels of mapping:

HPE SIM Node to UCMDB Node Mapping

All IP-enabled systems are represented as **Nodes** in HPE SIM and each node has attributes (for example, operating device type and operating system name) that can be used to classify nodes as specific CITs in UCMDB. The first level of mapping involves setting parameters on the **SIM Integration** job. This job includes **HostCitIdentifierAttributes** and **HostCitIdentifierMap** parameters that are used for the mapping:

- **HostCitIdentifierAttributes**. This attribute specifies the names of HPE SIM Node attributes that are used for the mapping. This parameter uses the **DeviceType** and **OSName** out-of-the-box Node attributes. The parameter accepts comma-separated node attribute names, is case sensitive, and expects each node attribute name to be enclosed in single quotes.
- **HostCitIdentifierMap**. This attribute specifies the mapping between values of the above HPE SIM Node attributes and corresponding UCMDB CITs. This parameter accepts a comma-separated list of value pairs, where each value pair takes the following format:

'node attribute value': 'UCMDB CI Type'

Both attributes are case-sensitive and must be enclosed in single quotes. Each Node-attribute value is one possible value of one or more Node attribute names specified in the **HostCitIdentifierAttributes** parameter. Each UCMDB CIT is the name (not the display name) of the UCMDB CIT to which this value maps.

This parameter has out-of-the-box mappings as follows:

HPE SIM Node Attribute	UCMDB CIT
'AIX'	'unix'
'Complex'	'complex'
'Embedded'	'management_processor'
'Enclosure'	'enclosure'
'HPUX'	'unix'
'Hypervisor'	'unix'
'LINUX'	'unix'
'MgmtProc'	'management_processor'
'Printer'	'netprinter'

HPE SIM Node Attribute	UCMDB CIT
'Rack'	'rack'
'Server'	'node'
'Solaris'	'unix'
'Switch'	'switch'
'WINNT'	'nt'
'Workstation'	'node'

Example mapping based on the above settings:

- If the **DeviceType** attribute of a node has the value **Switch**, in UCMDB the node is represented as a **Switch** CIT.
- If the **OSName** attribute of a node has the value **WINNT**, in UCMDB the node is represented as an **NT** CIT (Display name: **Windows**).

The DFM script parses these mapping parameters from left to right and does not stop on success, so the rightmost match is considered final. This means that if a node has **DeviceName = Server** and **OSName = HPUNIX**, the rightmost match is **OSName** with value **HPUNIX**. The resulting CIT for this node in UCMDB is **unix** because **HPUNIX** maps to **unix**.

Node Attribute to CI Type and CI Attribute Mapping

Once the nodes are mapped to CITs using DFM job parameters as described in ["HPE SIM Node to UCMDB Node Mapping" on the previous page](#), individual node attributes (including extended node attributes) are mapped to corresponding attributes (or CITs, as appropriate) using a generic UCMDB integration framework. The framework uses an XML file in which source and target CIT and attribute names are specified.

A sample XML mapping file (**SIM_To_UCMDB_Sample_MappingFile.xml**) that includes all node CITs mapped in the ["HPE SIM Node to UCMDB Node Mapping" on the previous page](#) section is included in the **SIM_Integration** package. The sample file includes host resources (for example, CPU, Disk) and relationship mapping information, to build relationships between various nodes (for example, Blade Enclosure to server, virtual machine host to guest, and so on).

Using this framework, you can map additional CITs without any code changes. For example, to map HBAs, add a new section to the XML file. Define the node attributes that identify an HBA and its attributes. Relationships between HBAs and HOSTs are also required.

How to Discover HPE SIM Data Center Infrastructure

This task describes how to discover data center infrastructure information stored in an HPE Systems Insight Manager (SIM) system.

This task includes the following steps:

- ["Prerequisites" below](#)
- ["Perform setup on the Probe machine" on the next page](#)
- ["Enable chunking - optional" on the next page](#)
- ["Run the job" on page 115](#)

1. Prerequisites

Important: If you set up an HTTPS connection to connect to the SIM Webservice API (that is, an SSL enabled HTTP connection), the **SIM Integration** job performs **no validation** of any certificates presented by the SIM server. The job trusts any certificate issued by the HPE SIM server and uses it for SSL enabled communication.

The following additional requirements must be satisfied for the mapping file to be valid for SIM (for details on the mapping files, see ["SIM Integration Mechanism" on page 110](#)):

- Verify that source and target are **SIM** and **UCMDB** respectively.
- Verify that attribute names specified in the **HostCitIdentifierAttributes** parameter are included as attributes of each host CIT in the XML file.

That is, the **OSName** and **DeviceType** attributes must be included for each **host_node** (Computer), **chassis** (Chassis), **netprinter** (Net Printer), **switch** (Switch), **nt** (Windows), **unix** (UNIX), **hp_complex** (Complex), and **management_processor** (Management Processor) CIT.

- Verify that default attributes (that is, non-extended attributes) of a node have a **Node.** prefix in the mapping file.

That is, you should specify attributes such as **OSName**, **DeviceType**, and **IPAddress** as **Node.OSName**, **Node.DeviceType**, and **Node.IPAddress**.

- Verify that each Node CIT has the following attribute mapping to enable the generation of the

host_key attribute:

```
<target_attribute name="host_key" datatype="StrProp" >  
  <map type="direct" source_attribute="host_key" />  
</target_attribute>
```

Note: The **host_key** attribute is the primary key attribute on Node and derived CITs. Since SIM uses a different type of key attribute, the XML definition for the **host_key** attribute is included in the mapping file, to enable generation of the **host_key** primary key attribute.

- Verify that the IP Address mapping section has the following attribute to enable automatic population of the IP domain attribute:

```
<target_attribute name="ip_domain" datatype="StrProp">  
  <map type="direct" source_attribute="ip_domain" />  
</target_attribute>
```

Note: For details on the list of SIM nodes and attributes, refer to the SIM documentation.

2. Perform setup on the Probe machine

- a. Copy **mxpartnerlib.jar** from this directory:

<DataFlowProbe_Home>\runtime\probeManager\discoveryResources\hpsim

to this directory:

<DataFlowProbe_Home>\content\lib

- b. Open **<DataFlowProbe_Home>\bin\WrapperEnv.conf** for editing.
- c. Comment out line ~51 with a hash sign (#) at the beginning so that it looks as follows:

```
#set.SYSTINET_CLASSES=%lib%/webservice;.....
```
- d. Save and close the file.
- e. Restart the Probe.

3. Enable chunking - optional

If the SIM server being discovered contains or manages a large number of nodes (more than 1,000), you should consider enabling chunking (**Data Flow Management > Adapter Management > select an adapter > Adapter Management tab > Adapter Parameters pane**):

Adapter Parameters	
Name	Value
ChunkSize	500
DebugMode	false
HostCidIdentifierAttributes	'DeviceType', 'OSName'
HostCidIdentifierMap	'Server':host_node', 'Workstation':host_node', 'Rack':rac...
dbIP	

- To reduce load on the SIM server, if necessary, you can set the **ChunkSize** parameter in the **SIM Integration** adapter to a lower value than the default **500**.
- Populate the **dbIP** parameter in the **SIM Integration** adapter with the IP address of the SIM CMS database.
- Populate the **SIM Database ...** fields in the SIM protocol with connection details for the SIM CMS database.

Note: SIM CMS database details (except for the password) are located in the **Systems Insight Manager\config\database.props** file on the SIM server.

4. Run the job

Note: For details on running an integration job, see "Integration Studio" in the *Data Flow Management* section of the *UCMDB Help*.

In the Integration Studio, create a new integration point.

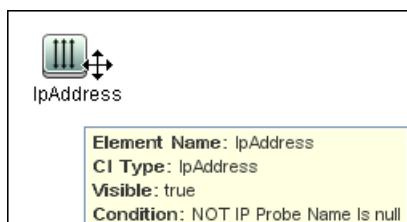
- Provide a name and description for the integration point.
- Under **Integration Properties > Adapter**, select the **Systems Insight Manager** adapter.
- Complete the **dbIP** field with the IP address of the SIM CMS database.
- Under **Adapter Properties > Data Flow Probe**, select the **Data Flow Probe**.
- Under **Adapter Properties > Trigger CI instance** select:
 - Select Existing CI** (if you have a valid, existing CI). The **Select Existing CI** pane appears. Select the CI or
 - Create New CI** (if you need to create a new CI). The **Topology CI Creation Wizard** appears. Complete the creation of the CI using the Wizard.

Note: For details on the Topology CI Creation Wizard, see "Topology CI Creation Wizard" in the *Data Flow Management* section of the *UCMDB Help*.

- f. Save the integration point.
- g. Run the job.

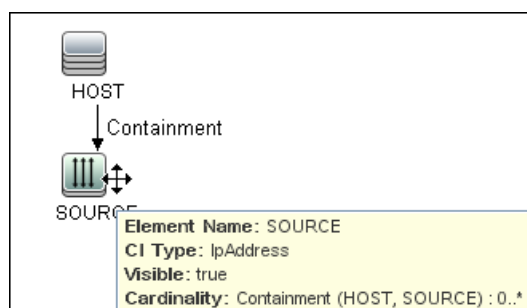
SIM WebService Ports Job

Trigger Query



Adapter

- Input query:



Discovered CITs

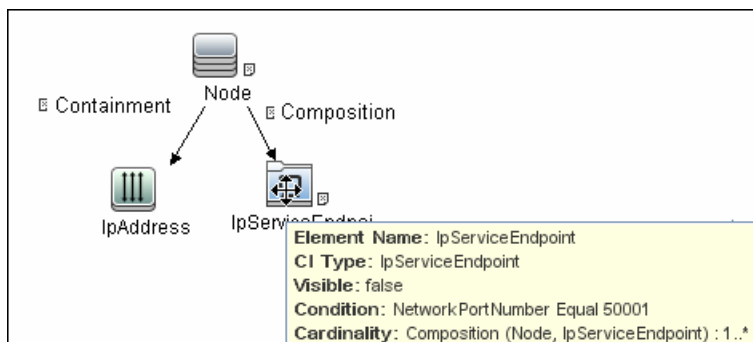
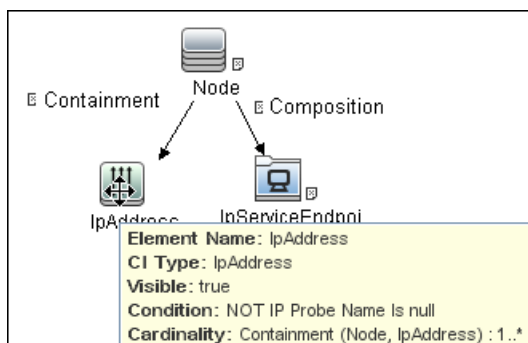
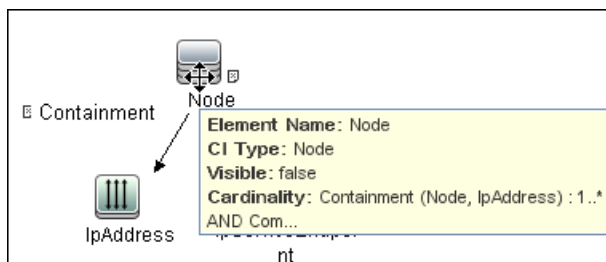
- Composition
- Containment
- IpAddress

- IpServiceEndpoint
- Node
- Usage

SIM Integration by WebServices Job

This section includes details about the job.

Trigger Query



Discovered CITs

- Chassis
- Composition
- Computer
- Containment
- Cpu
- Dependency
- Enclosure
- HP Complex
- Interface
- IpAddress
- LogicalVolume
- Management Processor
- Membership
- Net Printer
- Node
- Process
- Rack
- Switch

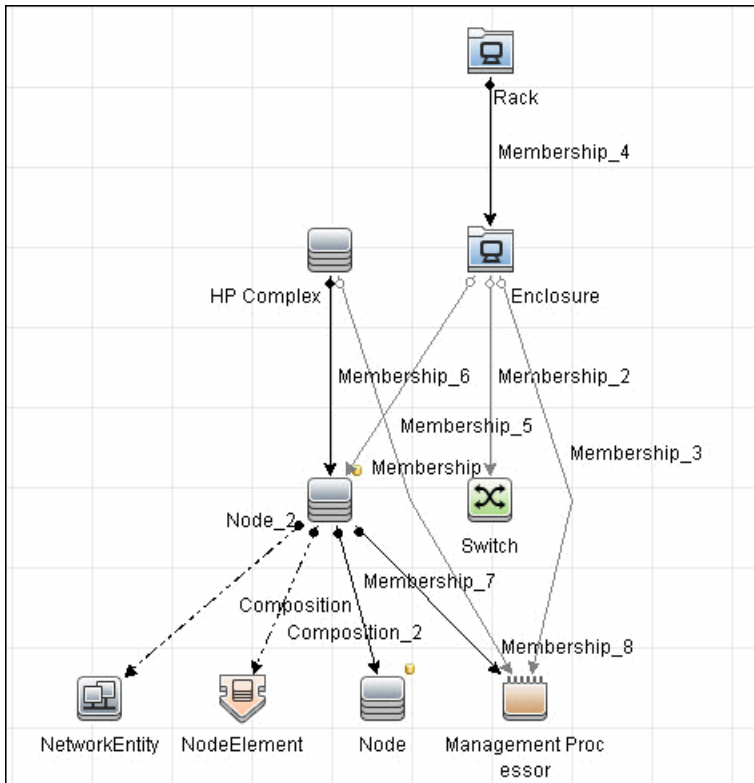
Instance Views

The package includes two adapter views that show all nodes and resources retrieved from SIM, as well as relationships between these nodes.

This section includes the following topics:

- ["Host Infrastructure View" on the next page](#)
- ["Hosts and Resources from HPE SIM" on page 120](#)

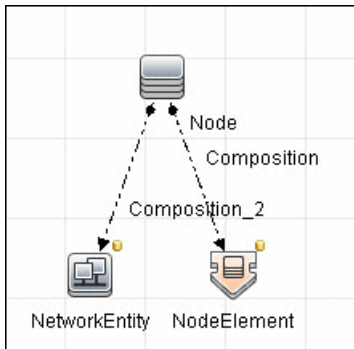
Host Infrastructure View



This view shows relationships between Chassis, Blade Enclosures, Servers, Workstations, Virtual Machine hosts to guests, and so on. This view also shows the interdependence between various nodes in an environment, to enable change management and correlation.

You can use this view, for example, to identify all the servers housed within a specific blade enclosure and all virtual machines running on servers within this blade enclosure. This enables analysis of the impact of shutting down a blade enclosure (say, for a firmware upgrade) on virtual machines. If UCMDB knows of services provided by these virtual machines and which business service these services are part of, it becomes possible to analyze the impact of a blade enclosure outage all the way to a business service.

Hosts and Resources from HPE SIM



This view shows Node CIs retrieved from HPE SIM with associated HostResource and NetworkResource CIs also retrieved from HPE SIM.

Troubleshooting and Limitations – SIM Integration

This section describes troubleshooting and limitations for SIM integration.

Note: The following limitation only applies when (a) using UCMDB Version 9.00-9.03, **and** (b) the user manually increased the out-of-the-box value in the pattern execution option **maximum threads** to greater than 1.

Limitation: If there are multiple SIM servers in the environment and this integration is used to integrate with all of them, you should create a new integration job for each SIM server and schedule them to run separately. This is because the integration uses XML files to process results from SIM, and running the integration against multiple SIM servers simultaneously causes the XML files to be overwritten (because the file name is static).

Chapter 11: IDS Scheer ARIS Integration

This chapter includes:

- Overview 122
- Supported Versions 122
- Topology 122
- How to Run the ARIS Integration Job 123
- Import CIs from ARIS Job 129

Overview

UCMDB integration with IDS Scheer ARIS IT Architect (ARIS) involves synchronizing business services/processes and Enterprise Architecture (EA) information from ARIS to the UCMDB database. This enables end-to-end Change Management and Impact Analysis from the IT infrastructure (at the data center level) to the business service/process level.

The integration involves a UCMDB initiated pull of information from an XML export generated by ARIS. Synchronized configuration items (CIs) include Business Service, Business Process, Business Process Step, Ownership information and Business Application (software). The integration requires manual reconciliation of business application instances in UCMDB.

ARIS_Integration.zip, contains the views, scripts, adapters, and jobs for the IDS Scheer ARIS Integration.

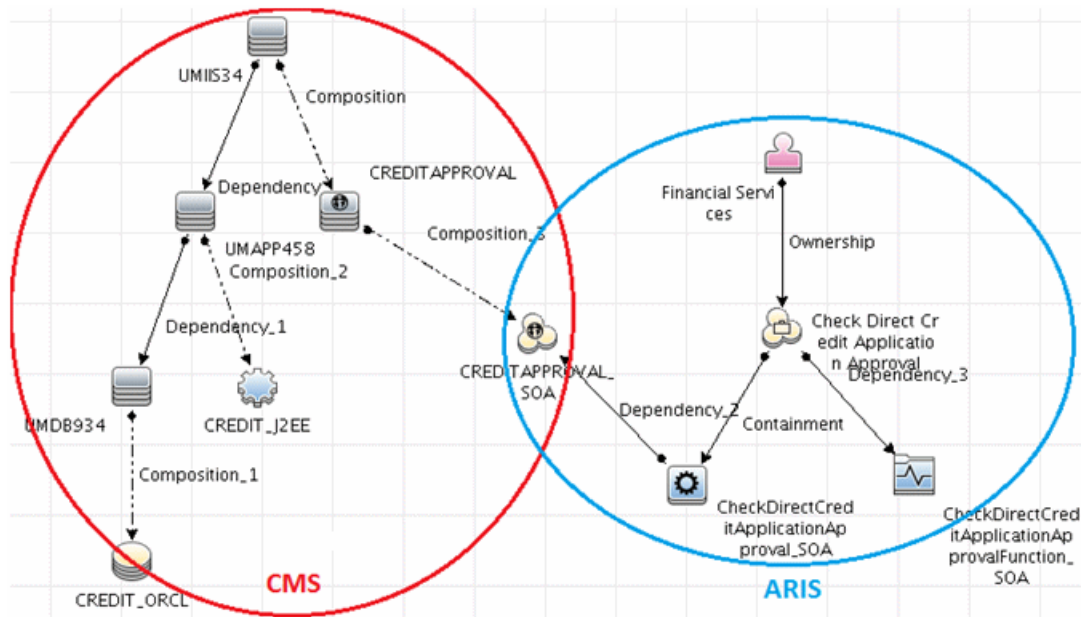
Supported Versions

This integration supports ARIS IT Architect version 7.1.

Topology

The following image is a sample topology showing relationships between the IT infrastructure (data center layer) and Business Processes/Services.

Note: For a list of discovered CITs, see ["Discovered CITs" on page 129](#).



How to Run the ARIS Integration Job

This task includes the steps to run the ARIS integration job, to integrate the IDS Scheer ARIS IT Architect CIs into UCMDB.

This task includes the following steps:

- ["Export the ARIS model to an XML file" below](#)
- ["Set up the ARIS-UCMDB mapping" on the next page](#)
- ["Run the job" on page 128](#)

1. Export the ARIS model to an XML file

This integration solution uses an XML output file generated by ARIS. It is recommended to export the ARIS model to a minimal XML file for use by the UCMDB integration job.

When exporting the data:

- The output XML file should NOT be compressed.
- The language of the output file must be the same as the language used for UCMDB.
- Configure settings as follows:

- Assignments: No assignments
- Connections: n connections, with a connection level of 1
- Select to perform a minimum export
- Options to export users and groups and group structures should NOT be selected.

Note: Save the exported file to a location accessible to the Data Flow Probe.

For more details on exporting XML files in ARIS, contact your IDS Scheer support representative or ARIS IT Architect documentation.

2. Set up the ARIS-UCMDB mapping

Data flow is initiated by UCMDB reading the XML file generated by ARIS. The integration job reads the data in this file and creates CIs.

A user configurable mapping file (also in XML format) may be used to customize mapping of:

- ARIS Object Types to UCMDB CI types
- ARIS links to UCMDB relationships

This mapping XML file, **ARIS_To_UCMDB.xml**, is located in the following folder:

```
<UCMDB installation>\UCMDB\DataFlowProbe\runtime\  
probeManager\discoveryResources\TQLEXP\ARIS\data
```

To set up the ARIS Object Type - UCMDB CI Type mapping:

Note: These mapping instructions are followed by an illustrated example.

- a. For each ARIS object type that you want to map, in the exported ARIS XML file (the **source** XML) locate the relevant **ObjDef** tag, and note the **TypeNum** and **AttrDef.Type** values.
- b. In the mapping file, **ARIS_To_UCMDB.xml**, locate the **<targetcis>** section and enter these values into the **source_CI_type** **namesource_attribute** attributes respectively.

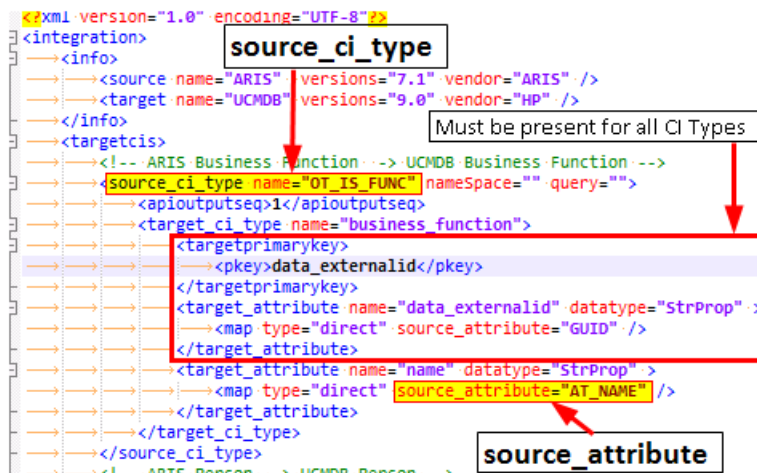
Example:

In the following image of the source XML file, the object, **ObjDef.4hzv--y----p--**, has the following attribute values:

- TypeNum = **OT_IS_FUNC**
- AttrDef.Type = **AT_NAME**



These values are entered in the mapping file's **source_CI_type** name and **source_attribute** attributes, as illustrated below:



Note: The section marked as **Must be present for all CI Types** must exist for ALL CI type mappings defined in the mapping file. This section populates the unique object ID used by ARIS in the "data_externalid" attribute of the UCMDB CI type.

To set up the ARIS Link - UCMDB Relationship mapping:

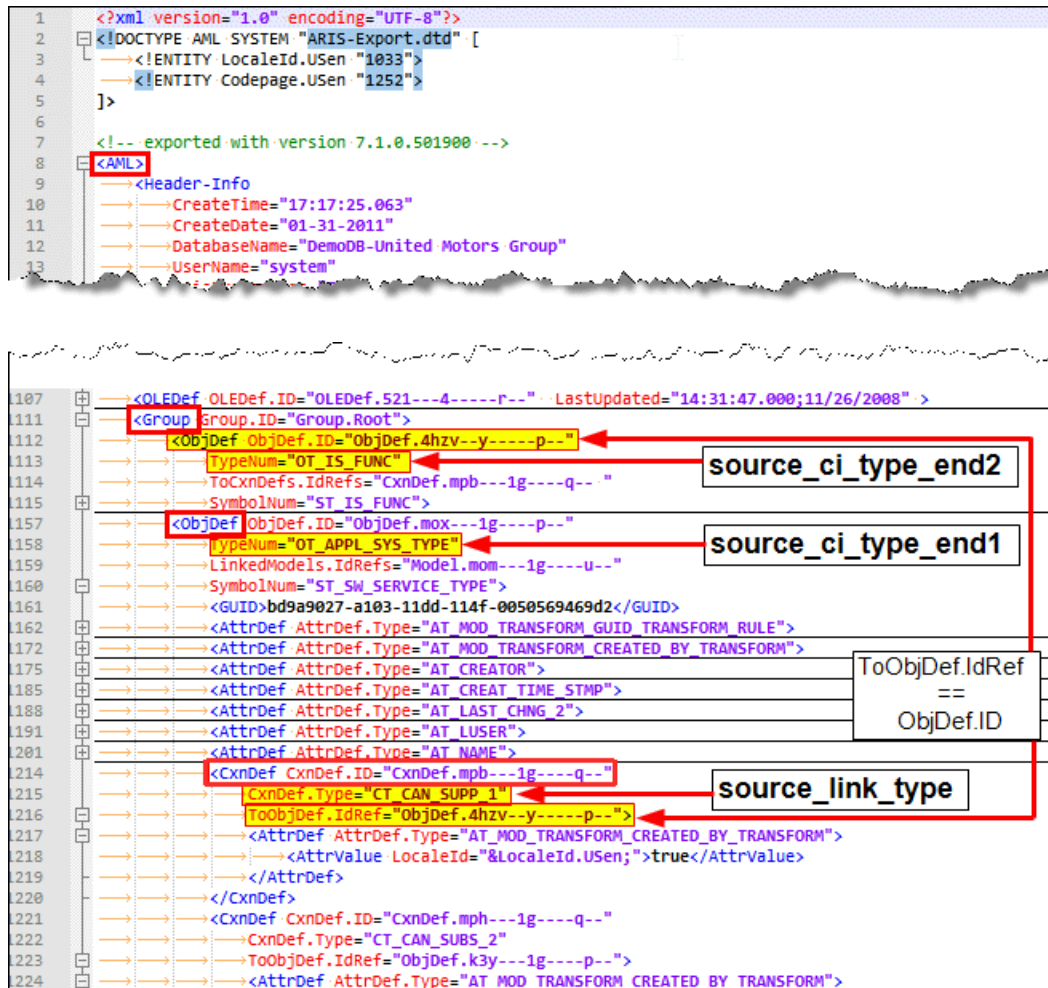
Note: These mapping instructions are followed by an illustrated example.

- c. For each ARIS link that you want to map, note the following values in the source XML file:
 - Locate the relevant **CxnDef** tag and note the **CxnDef.Type** attribute.
 - Locate the CxnDef tag's parent, **ObjDef**. Note the **TypeNum** value under this ObjDef.
 - Under CxnDef, note the **ToObjDef.IDRef** attribute, and search for an ObjDef tag with the identical value. Then, under this ObjDef, note the **TypeNum** attribute.
- d. In the mapping file, **ARIS_To_UCMDB.xml**, locate the **<targetrelations>** section and enter the source link's values as follows:
 - For **source_link_type**, enter the CxnDef.Type attribute
 - For **source_ci_type_end1**, enter the TypeNum value of the CxnDef tag's parent.
 - For **source_ci_type_end2**, enter the TypeNum value of the ObjDef that is equivalent to the ToObjDef.IDRef.

Example:

In the following image of the source XML file, the link, **CxnDefn CxnDef.ID=CxnDef.mpb---1g----q--**, has the following attribute values:

- CxnDef.Type = **CT_CAN_SUPP_1**
- CxnDef's parent's TypeNum attribute = **OT_APPL_SYS_TYPE**
- ToObjDef.IdRef = **ObjDef.4hzv--y----p--**. The equivalent ObjDef, **ObjDef.4hzv--y----p--**, was found in line 1112, and its TypeNum attribute is **OT_IS_FUNC**.



These values are entered in the mapping file's **<link>** tag, in the **source_link_type**, **source_ci_type_end1**, and **source_ci_type_end2** attributes respectively, as illustrated below:

```

34  <target_attribute name="name" datatype="StrProp
35  <map type="direct" source_attribute="AT_NAM
36  </target_attribute>
37  </target_ci_type>
38  </source_ci_type>
39  </targetcis>
40
41  <targetrelations>
42  <!-- ARIS link -> UCMDB relationship -->
43  <link source_link_type="CT_CAN_SUPP_1"
44  target_link_type="usage"
45  namespace=""
46  mode="update_else_insert"
47  source_ci_type_end1="OT_APPL_SYS_TYPE"
48  source_ci_type_end2="OT_IS_FUNC"
49  query="">
50  <apioutputseq>4</apioutputseq>
51  <target_ci_type_end1 name="business_application"/>
52  <target_ci_type_end2 name="business_function"/>
53  <targetprimarykey>
54  <pkey>name</pkey>
55  </targetprimarykey>
56  </link>
57  </targetrelations>
58
59  </integration>

```

3. Run the job

In the Integration Studio, create a new integration point.

- a. Provide a name and description for the integration point.
- b. Under **Integration Properties > Adapter**, select the **Software AG ARIS** adapter.
- c. Fill in the value for **ARIS_XML_file**. Set the value as the path to the XML file containing the exported ARIS data. See ["Export the ARIS model to an XML file" on page 123](#).
- d. Copy the DTD file, **ARIS-Export.dtd** from **<ARIS server>\Program Files\ARIS7.1\aml** to the directory where you saved the exported ARIS XML.
- e. Under **Adapter Properties > Data Flow Probe**, select the Data Flow Probe.
- f. Under **Adapter Properties > Trigger CI instance** select:
 - **Select Existing CI** (if you have a valid, existing CI). The **Select Existing CI** pane appears.
 - **Create New CI** (if you need to create a new CI). The **Topology CI Creation Wizard** appears. Complete the creation of the CI using the Wizard.

Note: For details on the Topology CI Creation Wizard, see "Topology CI Creation Wizard" in the *Data Flow Management section of the UCMDB Help*.

- g. In the Adapter Properties section, make sure that the following property is using its default value **true**:

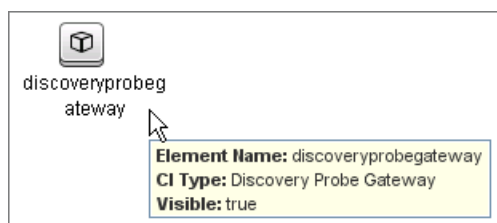
- **runInSeparateProcess.** This parameter enables the job to run in an external JVM process, separate from the main probe process. The default value is **true** and this value should not be changed.
- h. Save the integration point.
- i. Run the job.

Note: For details on running an integration job, see "Integration Studio" in the *Data Flow Management section of the UCMDDB Help*.

Import CIs from ARIS Job

Trigger Query

- Trigger CI: Probe
- Trigger query:



Adapter

- Input query: There is no input query for this job.

Discovered CITs

The UCMDDB-ARIS integration discovers the following CITs:

- BusinessApplication
- BusinessFunction
- BusinessService
- Containment
- Node
- Organization
- Ownership
- Person

Note: To view the topology, see ["Topology" on page 122](#).

Chapter 12: Importing Data from External Sources

This chapter includes:

Overview	132
Comma Separated Value (CSV) Files	132
CSV Files with Column Titles in First Row	133
Databases	133
Properties Files	133
How to Import CSV Data from an External Source – Scenario	135
How to Convert Strings to Numbers	140
Custom Converters	141
The External_source_import Package	141
Import from CSV File Job	143
Import from Database Job	146
Import from Properties File Job	153
External Source Mapping Files	155
Troubleshooting and Limitations – Importing Data from External Sources	155

Overview

Your data is probably stored in several formats, for example, in spreadsheets, databases, XML documents, properties files, and so on. You can import this information into Universal CMDB and use the functionality to model the data and work with it. External data are mapped to CIs in the CMDB.

The following external data sources are currently supported:

- ["Comma Separated Value \(CSV\) Files" below](#)
- ["Databases" on the next page](#)
- ["Properties Files" on the next page](#)

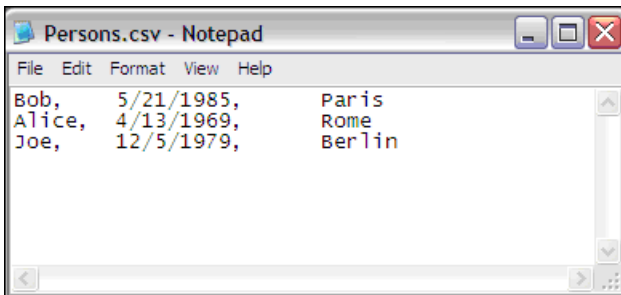
Comma Separated Value (CSV) Files

A *.csv file has a format that stores tabular data. Each row in a CSV file represents a set of values delimited with a particular delimiter. All rows are homogeneous, that is, each row has the same number of values. Values from all rows with the same index create a column. Values in a single column represent the same type of data. Therefore a CSV file represents a table of data (with rows and columns).

The default delimiter for CSV files is the comma, but any symbol can be used as a CSV delimiter, for example, a horizontal tab.

Note: Microsoft Office Excel includes native support for the CSV format: Excel spreadsheets can be saved to a CSV file and their data can then be imported into UCMDB. CSV files can be opened in an Excel spreadsheet.

Example of a CSV file:



CSV Files with Column Titles in First Row

CSV files often include column headings in the first row. When data is imported from these files, the titles are considered data and a CI is created for this row. To prevent a CI being created, you can define which row DFM should start at when importing data from a CSV file:

1. Select **Adapter Management > Resources pane > Packages > External_source_import > Adapters > Import_CSV**.
2. In the **Adapter Definition** tab, locate the **Adapter Parameters** pane.
3. Locate the **rowToStartIndex** parameter.

By default, the value is **1**, that is, DFM retrieves data from the first row.

4. Replace **1** with the number of the row at which to start retrieving data. For example, to skip the first row and start with the second row, replace **1** with **2**.

Databases

A database is a widely used enterprise approach to storing data. Relational databases consist of tables and relations between these tables. Data is retrieved from a database by running queries against it.

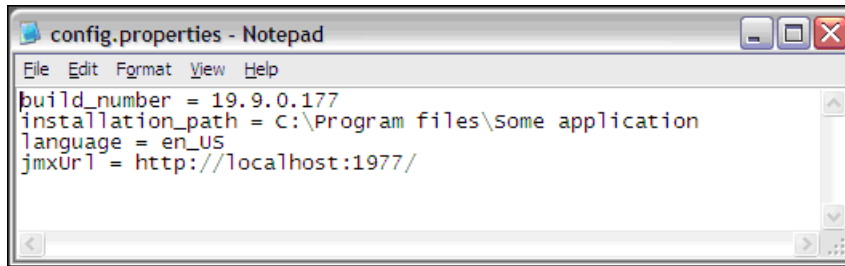
The following databases are supported: Oracle, Microsoft SQL Server, and MySQL.

Properties Files

A properties file is a file that stores data in the **key = value** format. Each row in a properties file contains one key-to-value association. In code terms, a properties file represents an associative array and each element of this array (key) is associated with a value.

A properties file is commonly used by an application to hold its configuration. If your application uses a configuration file, you can model the application in UCMDB.

Example of a properties file:



How to Import CSV Data from an External Source – Scenario

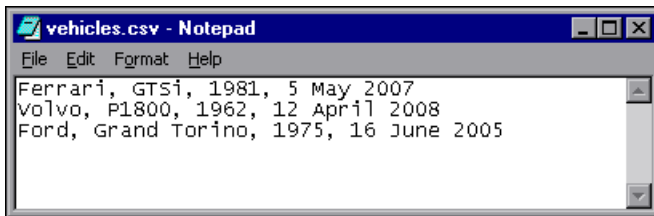
The UCMDB administrator must model a vehicle catalog that is stored in a CSV file.

This task includes the following steps:

- ["Prerequisites" below](#)
- ["Create a CIT" below](#)
- ["Create a mapping file" on the next page](#)
- ["Run the job" on page 138](#)
- ["Add the discovered Shell CI to the job" on page 138](#)
- ["Result" on page 138](#)

1. Prerequisites

The admin opens the CSV file and analyzes the data:



The file includes the name, model, year of manufacture, and the date when the car was purchased, that is, there are four columns of data:

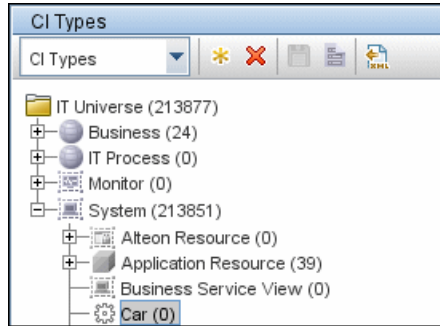
1	Name	string
2	Model	string
3	Year of manufacture	integer
4	Date of purchase	date

There are three rows to the file, which means that the admin expects three CIs to be created in UCMDB.

2. Create a CIT

The admin creates a CIT.

- a. The admin creates a CIT named **Car** to hold the attributes that are to be mapped to the data in the CSV file (name, model, and so on):



For details, see "Create a CI Type" in the *Modeling section of the UCMDB Help*.

- b. During the creation of the CIT, the admin adds these attributes as follows:

Key	Name	Display Name	Type
BODY_ICON	BODY_ICON	BODY_ICON	string
root_candidatefordel...	root_candidatefordel...	Candidate For Deleti...	date
date_of_purchase	date_of_purchase	Car Date of Purchase	date
model	model	Car Model	string
name	name	Car Name	string
year_of_manufacture	year_of_manufacture	Car Year of Manufa...	integer

For details, see "Attributes Page" in the *Modeling section of the UCMDB Help*.

3. Create a mapping file

The admin uses the template, **mapping_template.xml**, to create a mapping file that makes the information available to the **Import_CSV** adapter. The mapping file is located in the following folder: **Adapter Management > Resources pane > External_source_import > Configuration Files**.

- a. For each attribute, the admin adds a **<map>** marker:

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=".\\mapping_schema.xsd"
parserClassName="com.hp.ucmdb.discovery.library.
communication.downloader.cfgfiles.CiMappingConfigFile">
  <ci type="car">
    <map>
      <attribute>name</attribute>
```



```

        <column>0</column>
    </map>
    <map>
        <attribute>model</attribute>
        <column>1</column>
    </map>
    <map>
        <attribute>year_of_manufacture</attribute>
        <column>2</column>
    </map>
    <map>
        <attribute>date_of_purchase</attribute>
        <column>3</column>
    </map>
</ci>
</mappings>

```

- b. The admin then adds information about the attribute type:

```

<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=".\\mapping_schema.xsd"
parserClassName="com.hp.ucmdb.discovery.library.
communication.downloader.cfgfiles.CiMappingConfigFile">
    <ci type="">
        <map>
            <attribute>name</attribute>
            <column>0</column>
        </map>
        <map>
            <attribute>model</attribute>
            <column>1</column>
        </map>
        <map>
            <attribute>year_of_manufacture</attribute>
            <column>2</column>
            <converter module="import_converters">stringToInt</converter>
        </map>
        <map>
            <attribute>date_of_purchase</attribute>
            <column>3</column>
            <converter module="import_converters">stringToDate</converter>
        </map>
    </mappings>

```

All conversions between the values in the CSV file and the CI attributes are done by a converter. Several converter types are included in the package by default. For details, see ["How to Convert Strings to Numbers" on page 140](#).

4. Run the job

In the Integration Studio, create a new integration point.

- Provide a name and description for the integration point.
- Under **Integration Properties > Adapter**, select the **Import from CSV** adapter.
- Under **Adapter Properties > Data Flow Probe**, select the Data Flow Probe.
- Under **Adapter Properties > Trigger CI instance** select:
 - **Select Existing CI** (if you have a valid, existing CI). The **Select Existing CI** pane appears. Select the CI; or
 - **Create New CI** (if you need to create a new CI). The **Topology CI Creation Wizard** appears. Complete the creation of the CI using the Wizard.

Note: For details on the Topology CI Creation Wizard, see "Topology CI Creation Wizard" in the *Data Flow Management section of the UCMDDB Help*.

- Save the Integration Point.
- Run the job.

Note: For details on running an integration job, see "Integration Studio" in the *Data Flow Management section of the UCMDDB Help*.

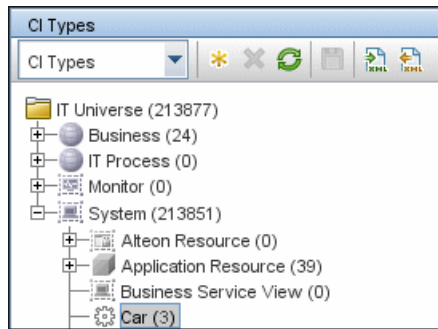
5. Add the discovered Shell CI to the job

Note: This step only applies if using **UCMDDB 9.03** and earlier.

After activation, the admin locates the **Shell** CI (of the machine where the `cars.csv` file is located) and adds it to the job. For details, see "Choose CIs to Add Dialog Box" in the *Data Flow Management section of the UCMDDB Help*.

6. Result

The admin accesses the CIT Manager and searches for instances of the **Car** CIT. UCMDDB finds the three instances of the CIT:



How to Convert Strings to Numbers

Converters enable you to specify the way data should be converted between the external source and a CI's attributes.

A CSV file contains records of type `string`. However, some of the record values need to be handled as numbers. This is done by adding a **converter** element to the **map** element (in `[your mapping file name].xml`):

```
<converter module="import_converters"></converter>
```

The **import_converters.py** file (**Adapter Management > Resources pane > Packages > External_source_import > Scripts**) contains a set of the most commonly needed converters and types:

- `toString`
- `stringToInt`
- `stringToLong`
- `stringToFloat`
- `stringToBoolean`
- `stringToDate`
- `stringToDouble`
- `skipSpaces`
- `binaryIntToBoolean`
- `stringToByteArray`
- `stringToZippedByteArray`
- `stringToList`

Note: The values of **stringToList** are separated by semicolons (;).

Example of a Converter

A CSV file contains the following row:

Usain, 21, Male

This row must be mapped to the **Person** CIT that includes name (`Usain`), age (`21`), and gender (`Male`) attributes. The age attribute should be of type **integer**. Therefore, the string in the CSV file must be

converted to an integer in the CIT to make it compliant with the CIT attribute type, before the Person CIs can retrieve the age values.

This is done by adding a **converter** element to the **map** element:

```
<map>
  <attribute>age</attribute>
  <column>2</column>
  <converter module="import_converters">stringToInt</converter>
</map>
```

module="import_converters". This attribute specifies from which module the converter is to be retrieved. A module is a Jython script file that contains a set of converter methods, in this case, `import_converters.py`.

stringToInt. The name of the converter. In the `import_converters.py` file, the method is written as follows:

```
def stringToInt(value):
    if value is not None:
        return int(value.strip())
    else:
        return 0
```

Custom Converters

You can write your own custom converters: Add a new method to the `import_converters.py` file or create your own script and add a set of converter methods to it. Call the method with the name of the script, for example:

```
<converter module="your_converter_script">[your_converter_method]
</converter>
```

The External_source_import Package

The **External_source_import** package consists of three jobs and three adapters. There is one job and one adapter for each external source (CSV file, properties file, database):

External Source	Job	Adapter
CSV file	Import from CSV file	Import_CSV
Properties file	Import from Properties file	Import_Properties_file

External Source	Job	Adapter
Database	Import from Database	Import_DB

The adapters are located in the **Integration Studio** and are accessed when creating or editing an Integration Point.

Import from CSV File Job

This section includes the following topics:

- ["Job Details" below](#)
- ["Adapter Parameters" below](#)
- ["Delimiters, Quotes, and Escaping Characters" on page 145](#)

Job Details

The job details are as follows:

- Adapter: Import from CSV
- Input CI Type: Shell
- Discovered CIs: ConfigurationItem
- Required Protocols: SSH, NTCMD, Telnet

This job has no Trigger queries associated with it. That is, this job is not triggered automatically (nor are the Import from Properties file and the Import from Database jobs). After you activate the job, you must manually add input CIs to the job so that it runs against a particular destination. For details, see ["Add the discovered Shell CI to the job" on page 138](#).

Adapter Parameters

The following parameters are included by default:

Parameter	Description
bulkSize	This parameter only works if the parameter flushObjects is true , in which case, when sending discovery results, it sets the size of chunks used to that number of CIs. Default: 2,000 (CIs).

Parameter	Description
ciType	The CIT name. This job creates and reports CIs of this type to UCMDB, based on data in the CSV file. For example, if the CSV file contains records for UNIX hosts, you must set the ciType parameter to unix .
csvFile	The full path to the CSV file on the remote machine. The job uses the Shell CI type as input to reach this path on the remote machine.
delimiter	The delimiter used in the CSV file. The comma (,) delimiter is the default but other delimiters are supported. For details, see "Delimiters" on the next page .
flushObjects	<p>This parameter allows customization of the reporting mechanism.</p> <p>If true, the probe divides the discovery result into chunks, and sends each chunk to the UCMDB Server. This helps prevent out-of-memory issues where a large amount of data is sent. The chunk size can be configured with the bulkSize parameter.</p> <p>If false (the default value), the probe sends the discovery result without dividing it into chunks.</p>
mappingFile	For details of the mapping file, see "External Source Mapping Files" on page 155 .
mappingString	<p>The string containing mapping information used to map the CSV column indexes and attributes to import. You define this mapping in the following format:</p> <ul style="list-style-type: none"> mapping elements should be separated by commas each mapping element should be specified in a <column number>:<attribute name> format, for example: <p>The string 0:host_key,1:name defines the mapping of two attributes of a host CI, where the host's host_key attribute is taken from the value in the first column (0) and the name attribute is taken from the value in the second column (1).</p>
processWithBinaryMode	<p>Indicates whether to copy the target CSV file to Data Flow Probe and decode it using the value of the fileEncoding parameter.</p> <p>Default: false.</p> <p>Note: If false, the target CSV file may be in a different encoding method and the fileEncoding parameter in the Import from CSV File job might not be used correctly.</p>

Parameter	Description
quoteSymbol	Quoting symbol used in the CSV file. Default symbol: "
rowToStartIndex	For details on setting the row at which DFM starts collecting data, see "CSV Files with Column Titles in First Row" on page 133 .
skipEmptyValues	This flag determines whether to skip empty values. If true , empty column values are not sent.

For details on overriding an adapter parameter, see "Override Adapter Parameters" in the *Developer Reference section of the UCMDDB Help*.

Mapping Information for the Import from CSV File Job

You can specify mapping information for the **Import from CSV File** job with one of the following methods:

- In an external XML file. You must specify the **mappingFile** parameter. For details, see ["External Source Mapping Files" on page 155](#).
- Directly in a job's **ciType** and **mappingString** parameters, without using an external file.

Note: When using this mapping method, you cannot specify attribute types or converters.

If the **mappingFile** parameter is specified, the job tries to retrieve mapping information from the XML file. If it is not specified, the job uses the mapping information specified in the **ciType** and **mappingString** parameters.

Delimiters, Quotes, and Escaping Characters

- Delimiters

The delimiter divides values in the same row of a CSV file. Supported delimiters are:

- **Single symbol.** Any symbol can be used as a delimiter, for example, the pipe sign (|), the letter O. Delimiters are case sensitive.
- **ASCII code.** If an integer number is used as the value for a delimiter parameter, this value is

treated as ASCII code, and the related symbol is used as the delimiter. For example, **9** is a valid delimiter because **9** is the ASCII code for the horizontal tab.

- **Known character sequence.** A sequence of characters can be used to represent special characters. For example, `\t` represents the horizontal tab.

- Quotation Marks

You can use double or single quotes in values, that is, all values residing between the two quotes are treated as a single value.

- If a delimiter symbol is used in a value, the value must be surrounded with quotation marks. For example, the following row includes a comma inside a value, so the value must be quoted:

Morganfield, "25 Hope Road, Kingston", Jamaica

- If a quote character is used in a value, the character must be escaped by inserting a backslash before it:

McKinley \"Muddy Waters\" Morganfield, \"April 4, 1915\"

This row contains two values:

- McKinley "Muddy Waters" Morganfield
- April 4, 1915.

- Escaping Characters

The following characters must always be quoted or escaped:

- Backslash
- Single quote
- Double quote
- Delimiter, that is, the delimiter used in the same CSV file.

Import from Database Job

This job uses a database table or database query as the source of the information, maps the information to CIs, and imports the CIs into UCMDB.

This section includes the following topics:

- ["Job Details" below](#)
- ["Discovery Adapter Parameters" below](#)
- ["Tables and Queries" on the next page](#)
- ["Database, Schema, and Table Names" on page 149](#)
- ["Importing Data with an SQL Query" on page 150](#)
- ["Column Types" on page 150](#)
- ["Importing CIs with Root Container Attribute" on page 151](#)

Job Details

The job details are as follows:

- Adapter: Import from DB
- Input CI Type: Database
- Discovered CIs: ConfigurationItem
- Required Protocol: SQL

This job has no trigger queries associated with it. The job tries to get the Instance name and Port using the attributes **Name** and **Application Listening Port Number** of the **Input Database** CI. If these attributes are empty, it uses the Instance Name and Port number defined in Generic DB Protocol (SQL) credentials.

Discovery Adapter Parameters

The following parameters are included by default:

Parameter	Description
bulkSize	This parameter only works if the parameter flushObjects is true , in which case, when sending discovery results, it sets the size of chunks used to that number of CIs. The default is 2,000 CIs.
ciType	Name of CIT to import.

Parameter	Description
flushObjects	<p>This parameter allows customization of the reporting mechanism.</p> <p>If true, the probe divides the discovery result into chunks, and sends each chunk to the UCMDDB Server. This helps prevent out-of-memory issues where a large amount of data is sent. The chunk size can be configured with the bulkSize parameter.</p> <p>If false (the default value), the probe sends the discovery result without dividing it into chunks.</p>
mappingFile	XML file containing the mapping from column to attribute.
mappingString	<p>The string containing mapping information used to map the Database column names and the attributes to import. You define this mapping in the following format:</p> <ul style="list-style-type: none">• mapping elements should be separated by commas;• each mapping element should be specified in a <column name>:<attribute name> format, <p>Example:</p> <p>A_IP_ADDRESS:ip_address, A_IP_DOMAIN:ip_domain</p>
schemaName	The name of the database schema.
sqlQuery	If a SQL query is specified, mapping is performed against its result. This parameter is ignored if tableName is defined.
tableName	If a table name is specified, mapping is performed against the table's columns.

For details on overriding an adapter parameter, see "Override Adapter Parameters" in *Developer Reference section of the UCMDDB Help*.

Tables and Queries

The following use cases are supported by the Import from Database job (a single SQL query is performed):

- Import data using the schema name and table name parameters:

Adapter Parameters	
+ X Pencil	
Name	
ciType	
mappingFile	
mappingString	
schemaName	ddmi_servers
sqlQuery	
tableName	servers

The SQL query is generated from these parameters.

- Import data specifying an arbitrary SQL query as the source of the data:

Adapter Parameters	
+ X Pencil	
Name	
ciType	
mappingFile	
mappingString	
schemaName	
sqlQuery	SELECT servers.* FROM servers LEFT JOIN disks C
tableName	

The SQL query is generated from the defined query. For more details, see ["Importing Data with an SQL Query" on the next page](#).

Database, Schema, and Table Names

SQL naming conventions suggest a usage of a <database.schema.table> syntax for the fully qualified name of a table. Note, however, that each vendor treats the specification in a different way. DFM uses the following notation:

- The **schemaName** parameter specifies the name of a database.
- The **tableName** parameter specifies the name of a table.
- A schema name cannot be specified in a parameter but can be included in a SQL query.

For Oracle, the SQL query is:

```
SELECT * FROM <schemaName.tableName>
```

For Microsoft SQL Server, the SQL query is:

```
SELECT * FROM dbo.tableName
```

Note: The default dbo schema is used for Microsoft SQL Server.

Importing Data with an SQL Query

You can use arbitrarily-complex SQL query expressions, for example, joins, sub-selects and other options, as long as the query is valid and complies with the database usage. Currently, you must use a fully-qualified table name in the query according to the specific database.

Column Types

Types enable you to specify, in the mapping file, the type of column that exists in the external source. For example, a database includes information about column types, and the value of this type needs to be included in the CI's attributes. This is done by adding a **type** element to the **map** element (in `mapping_[your mapping file name].xml`):

```
<column type="int"></column>
```

Supported type attributes are:

- string
- Boolean
- date
- int
- long
- double
- float
- timestamp

Note:

- You use the **type** attribute for database mapping only.
- If the column element does not include a type attribute, the element is mapped as a string.

Example of adding a type attribute

A database column has an integer type and can be either 0 or 1. This integer must be mapped to a Boolean attribute of a CIT in UCMDB. Use the binaryIntToBoolean converter, as follows:

```
<map>
  <attribute>cluster_is_active</attribute>
  <column type="int">cluster_is_active</column>
  <converter module="import_converters">binaryIntToBoolean</converter>
</map>
```

type="int". This attribute specifies that the value of `cluster_is_active` should be retrieved as an integer, and that the value passed to the converter method should be an integer.

If the `cluster_is_active` attribute of the CIT is of type integer, the converter is not needed here, and the mapping file should say:

```
<map>
  <attribute>cluster_is_active</attribute>
  <column type="int">cluster_is_active</column>
</map>
```

Importing CIs with Root Container Attribute

The main purpose of the **Import from database** job is importing CIs that do not require a root container attribute. For example: Node, IP address, etc. However, there are many CIs that depend on root CIs (for example: CPUs, SQL servers, etc) that cannot be deployed into UCMDB without the Node on which they are installed. In such cases, it is better to use a different discovery approach. For example, by importing data from Excel, you can import CIs with corresponding relationships (including the composition relationship). You can import such CIs using the **Import from database** job only if all of the following apply:

1. The root CI (for example: Node) was already populated into UCMDB.
2. It is possible to construct an SQL query to select the UCMDB ID of already imported root CIs from an external database.

In such cases, you can map the selected UCMDB ID of the root CI to the root container attribute of the contained CI.

Note: Validation of data (checking if the root CI already exists in UCMDB) is not supported. It is your responsibility to correctly configure the population of the root container attribute.

Import from Properties File Job

This job imports information from a properties file, maps the information to one CI, and imports that CI into UCMDB.

This section includes the following topics:

Job Details

The job details are as follows:

- Adapter: Import from properties file
- Input CI Type: Shell
- Discovered CIs: ConfigurationItem
- Required Protocols: SSH, NTCMD, Telnet

This job has no Trigger queries associated with it.

Discovery Adapter Parameters

The following parameters are included by default:

Parameter	Description
bulkSize	This parameter only works if the parameter flushObjects is true , in which case, when sending discovery results, it sets the size of chunks used to that number of CIs. The default is 2,000 CIs.
ciType	The name of the CIT to import.

Parameter	Description
flushObjects	<p>This parameter allows customization of the reporting mechanism.</p> <p>If true, the probe divides the discovery result into chunks, and sends each chunk to the UCMDB Server. This helps prevent out-of-memory issues where a large amount of data is sent. The chunk size can be configured with the bulkSize parameter.</p> <p>If false (the default value), the probe sends the discovery result without dividing it into chunks.</p>
mappingFile	For details of the mapping file, see "External Source Mapping Files" on the next page .
mappingString	<p>The string containing mapping information used to map the column indexes and attributes to import. You define this mapping in the following format:</p> <ul style="list-style-type: none"> mapping elements should be separated by commas each mapping element should be specified in the format: <parameter name> :<attribute name> <p>For example, for CIT ip_address:</p> <pre>ip_address:ip_address,name:name,ip_address_type:ip_address_type</pre>
propertyFile	The full path to the properties file located on a remote machine. The Input CI runs the Shell discovery that is used to access this file on the remote machine

For details on overriding an adapter parameter, see "Override Adapter Parameters" in the *Developer Reference section of the UCMDB Help*.

Keys and Values

Keys cannot contain the equals symbol (=).

Each value must be set out in a single line. Use **backslash+n (\n)** to specify a new line. Values can contain anything, including **\n** for a new line, quotes, tabs, and so on.

Comments in Properties Files

To create a commented line in a properties file, add the pound sign (#) as the first character in a line. The job ignores commented lines.

External Source Mapping Files

The data in the external source is mapped to a CI's attributes in UCMDB by means of a mapping file.

The mapping files are located in the **Adapter Management > Resources pane > Packages >**

External_source_import > Configuration Files folder:

- **mapping_template.xml**. A template that serves as a source for creating the mapping file.
- **mapping_schema.xsd**. The XML schema used to validate the XML mapping file. The XML mapping file must be compliant with this schema.
- **mapping_doc.xml**. A file that contains Help on creating a mapping file, including all valid elements.

The mapping file describes the mapping only and does not include information about how data should be obtained. In this way, you can use one mapping file across different jobs.

All the adapter files in the **External_source_import** package include a `mappingFile` parameter, for example:

```
<parameter name="mappingFile" type="string" description="Mapping file located in  
&quot;Configuration Files&quot; folder of this package" />
```

name="mappingFile". The value of this parameter is the mapping XML file. The mapping file is always located on the server and is downloaded to the Data Flow Probe machine upon job execution.

Troubleshooting and Limitations – Importing Data from External Sources

- **Problem:** When CIs imported from a CSV file are displayed in the Statistics Results pane, one more CI than expected is included in the results. This is because the first row of the CSV file contains column headings that are considered as CIs.

Solution: For details on defining from which row DFM should read the CSV file, see ["CSV Files with Column Titles in First Row" on page 133](#).

- **Problem:** When importing large CSV or properties files on the network, there may be time-out issues.

Solution: Make sure the files are not large.

- **Limitation:** When importing data from an external database, and the data includes a null value, it is sent to UCMDB with an attribute value of None.
- **Limitation:** The DFM Probe breaks down the imported data into 20 KB chunks. This can cause identification issues.

Chapter 13: Import from Excel Workbook Discovery

This chapter includes:

Overview	158
Supported Versions	158
Topology	158
How to Import Data from Excel Workbook	158
How to Set Up an Import File in Excel	161
Import from Excel Workbook Job	169
Troubleshooting and Limitations – Import From Excel Workbook Discovery	173

Overview

This chapter describes the usage and functionality of the XLS_Import discovery package developed for importing UCMDB topology from a Microsoft Excel (*.xls, *.xlsx) file.

Supported Versions

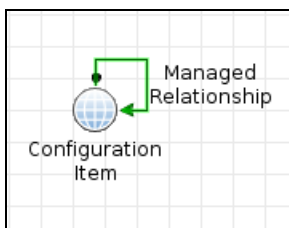
This discovery supports

- Microsoft Excel files, versions 97, 2000, XP, and 2003 (*.xls)
- Office Open XML format for Excel 2007 (*.xlsx)

Topology

The following image displays the topology of the Import from Excel discovery.

Note: The topology discovered by the Import from Excel Workbook job relies on import file content, so only root objects are enumerated as discovered CITs. For a list of discovered CITs, see ["Discovered CITs" on page 173](#).



How to Import Data from Excel Workbook

This task describes how to run the Import from Excel discovery. The Import from Excel Workbook job imports data from the Probe's file system (or accessible network share), so no credentials are required.

Note:

- The Import from Excel Sample job is similar to the Import from Excel Workbook job. It differs only by reference to the sample import file.

- This integration is configured to run in remote process.

This task includes the following steps:

1. **Prerequisite- Set up the Import file in Excel**

For details on setting up the import file, see ["How to Set Up an Import File in Excel" on page 161.](#)

2. **Prerequisite - Set up permissions**

Give the Data Flow Probe read permissions on the location on the file system where the import files are stored.

3. **Run the discovery**

In the Integration Studio, create a new integration point.

- a. Provide a name and description for the integration point.
- b. Under **Integration Properties > Adapter**, select the **Import topology from Excel Workbook** adapter.
- c. Under **Adapter Properties > Data Flow Probe**, select the Data Flow Probe.
- d. Under **Adapter Properties > Trigger CI instance** select one of the following:
 - **Select Existing CI** (if you have a valid, existing CI). The **Select Existing CI** pane appears. Select the CI and click OK.
 - **Create New CI** (if you need to create a new CI). The **Topology CI Creation Wizard** appears. Create the CI using the Wizard.

Note: For details on the Topology CI Creation Wizard, see "Topology CI Creation Wizard" in the *Data Flow Management section of the UCMDB Help*.

- e. Save the Integration Point.
- f. Run the job.

Note: For details on running an integration job, see "Integration Studio" in the *Data Flow Management section of the UCMDB Help*.

How to Set Up an Import File in Excel

This section describes how to define an import file. The following topology is created:

- Two hosts
- Two IPs contained by each host
- Network (the IPs mentioned above are members of the network)
- An application with a corresponding process running on the host

This task includes the following steps:

1. ["Prerequisite" below](#)
2. ["Add a CI type" below](#)
3. ["Create Comment sheets - optional" on the next page](#)
4. ["Define CI key attributes " on the next page](#)
5. ["Create Comment columns - optional" on page 163](#)
6. ["Add CIs with containers" on page 164](#)
7. ["Define relationships" on page 166](#)
8. ["Add relationship attributes" on page 167](#)
9. ["Convert attribute types to UCMDB attribute types" on page 168](#)

1. Prerequisite

Open a new Excel file and name it **tutorial.xls**.

2. Add a CI type

Double-click the **Sheet1** tab and rename it with the desired CI type. For this tutorial, use the name **node**.

Note:

- Only use the CI type name, not the display name.
- Type names are case sensitive.

3. Create Comment sheets - optional

You can create Comment sheets that will not be imported into UCMDB, but that can be used to describe the data contained in the imported document.

Double-click one of the Sheet tabs and rename it **#Comment sheet**.



Note: Comment sheet names must begin with the # sign.

4. Define CI key attributes

Depending on the CIT, to store a CI in UCMDB, you must specify CI key attributes or attributes that participate in reconciliation rules. The names of the imported attributes can be defined as the column headings.

Our **node** object only has one key attribute - **host_key**.

Key	Display Name	Name
🔑	Host Key	host_key
	Host Model	host_model
	Host Name	host_hostname

To import a node CI into UCMDB, you must set the **host_key** attribute. You may do this by one of the following methods:

- Set **host_key** in the view **<IP address> <Domain>**. (For example: 192.168.100.100 DefaultDomain.) This is enough to import a node CI into UCMDB.
- Set **host_key** as the lowest MAC address of the attached network interface. This is not enough to import a node CI into UCMDB. You must also configure the following attributes:
 - i. Set **host_iscomplete** to **true**.
 - ii. Set values for the node attributes that allow the node to be identified by the reconciliation rule. Note that the node reconciliation rule also allows identification of the nodes linked to this node IP address or network interface CIs. If you prefer to identify nodes by linked CIs, you must ensure the Excel document also has the imported IP/Network interface CIs, and the relationships between node CIs and IP/ network interface CIs.

Note:

- The column headings must be attribute names, not display names.
- Attribute names are case sensitive.

You can show the node name and the operating system.

	A	B	C
1	host_key	name	discovered os name

- a. Define two nodes.

	A	B	C
1	host_key	name	discovered os name
2	192.168.100.100 MyDomain	SampleHost	Windows XP
3	192.168.100.200 MyDomain	SampleServer	Windows 2008

Note: Each row in the sheet (except the first one) represents a single CI.

- b. Use the same procedure to define IP addresses in a second Excel sheet, for example, **Sheet2**.

	A	B	C
	ip_address	routing_domain	name
	192.168.100.100	MyDomain	192.168.100.100
	192.168.100.101	MyDomain	192.168.100.101
	192.168.100.200	MyDomain	192.168.100.200
	192.168.100.201	MyDomain	192.168.100.201

- c. Use the same procedure to define a network CI in a third Excel sheet, for example, **Sheet3**.

	A	B	C	D	E
1	name	network_netmask	routing_domain	network_netclass	ip_prefix_length
2	192.168.100.0	255.255.255.0	MyDomain	C	1

running_software and process definitions are described in ["Add CIs with containers" on the next page](#)

5. Create Comment columns - optional

If you want to have a **Comment** column with explanations of data, use the # sign before the column heading. Any data placed in this column will not be imported into UCMDB.

	A	B	C	D
1	host_key	name	discovered_os_name	#Comment
2	192.168.100.100 MyDomain	SampleHost	Windows XP	PC near the entrance
3	192.168.100.200 MyDomain	SampleServer	Windows 2008	Our server

6. Add CIs with containers

Objects that are contained within other objects cannot exist without them. For example, processes and running software cannot exist without the node they are running on. To show this relationship, a **root_container** attribute is needed. Because the container is in another CI, a reference to it is needed.

Objects can be referenced in one of the following ways:

- **By creating an Excel definition reference to the object.**

The Excel definition referencing style is recommended because only the tab name (CI type name) and row number (the row number of the CI defined on the tab) are needed to identify any imported CI - the presence or absence of any key fields is not necessary, reconciliation rules are defined in UCMDB, and so on.

Typical links appear as **=node!A2**, meaning that the **node** tab on the CI defined at row **2** is being referenced. It does not matter which column you are referencing; only the rows numbers are significant.

Note: Such references cannot be used if the Excel file was created from a CSV file or using some other non-Excel format.

For more information about references, see Microsoft Excel documentation.

- **By setting a composition of the desired object key fields divided by the pipe symbol ('|').**

For example, to reference an IP address, the **ip_address** and **routing_domain** attributes are needed: **192.168.100.100|MyDomain**.

Note:

- The order of the key fields in the definition is important!
- Many objects have no keyed attributes and are identified with reconciliation rules. For this reason, Excel references are preferred.

- **By creating an Excel definition reference to the object.**

The Excel definition referencing style is recommended because only the tab name (CI type name) and row number (the row number of the CI defined on the tab) are needed to identify any imported CI - the presence or absence of any key fields is not necessary, reconciliation rules are defined in UCMDB, and so on.

Typical links appear as **=node!A2**, meaning that the **node** tab on the CI defined at row **2** is being referenced. It does not matter which column you are referencing; only the rows numbers are significant.

Note: Such references cannot be used if the Excel file was created from a CSV file or using some other non-Excel format.

For more information about references, see Microsoft Excel documentation.

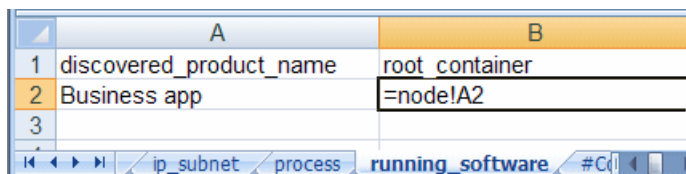
- **By setting a composition of the desired object key fields divided by the pipe symbol ('|').**

For example, to reference an IP address, the **ip_address** and **routing_domain** attributes are needed: **192.168.100.100|MyDomain**.

Note:

- The order of the key fields in the definition is important!
- Many objects have no keyed attributes and are identified with reconciliation rules. For this reason, Excel references are preferred.

a. Create a **running_software** using Excel references.



	A	B
1	discovered_product_name	root_container
2	Business app	=node!A2
3		

Note: To define an Excel reference, type an equal sign (=) in a cell, select the desired reference cell, and press ENTER.

- b. Create a **process** using a composite key.

	A	B	C
1	name	process_cmdline	root_container
2	Sample	C:\sample.ddm	192.168.100.100 MyDomain
3			

ip_subnet process running_software #Com...

7. Define relationships

To define relationships, create a sheet called **relationships**.

Note: You cannot import relationship CIs.

All links (relationships) in UCMDB are directed. This means each link has a start and end point. Also, links have names that might have some attributes similar to other CIs.

A link definition in an import file looks as follows:

Start object reference -> link name -> End object reference [-> Attributes]

Link attribute definitions are described in ["Add relationship attributes" on the next page](#).

The first row (column headings) displays the reason for the information. On this sheet, only the order of the parameters is important.

- a. Using Excel references, add informative captions and define member links between the IP subnet and first two IP addresses.

	A	B	C
1	start	relation_type	end
2	=ip_subnet!A2	membership	=ip_address!A2
3	=ip_subnet!A2	membership	=ip_address!A3

relationships node ip_address ip_subnet process

In this image, defined formulas are displayed (for example, **=ip_address!A2**). In actuality, the values of referenced cells are shown.

- b. Using key composition, define the relationships between the two IP addresses and their routing domains as follows:

IP key fields are ip_address and routing_domain. The composite key looks like **192.168.100.100|MyDomain**.

The relationship tab looks as follows:

	A	B	C
1	start	relation_type	end
2	192.168.100.0	membership	192.168.100.100 MyDomain
3	192.168.100.0	membership	192.168.100.101 MyDomain
4	192.168.100.0	membership	192.168.100.200
5	192.168.100.0	membership	192.168.100.201

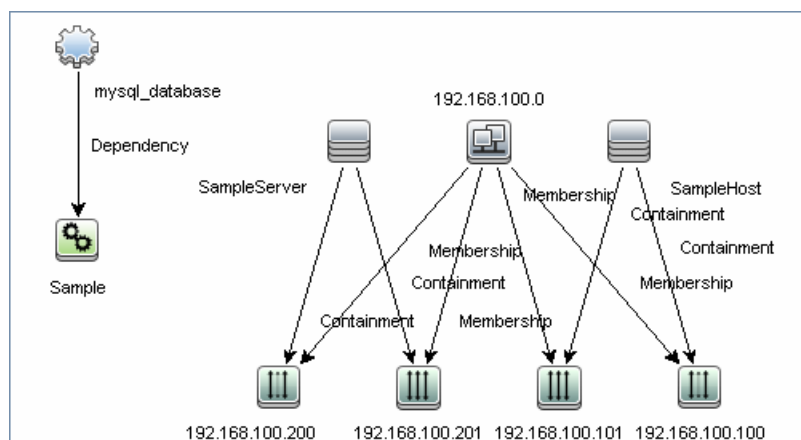
Note:

- Any type of reference can be chosen. You can use only one reference type in a cell.
- Since the IP subnet CI has no key attributes in UCMDB 9.0x, they can be referenced only by Excel reference.

- c. Add a **containment** reference from **node** to **ip_address** and add a **dependency** reference from **running_software** to **process**:

	A	B	C
1	start	relation_type	end
2	192.168.100.0	membership	192.168.100.100 MyDomain
3	192.168.100.0	membership	192.168.100.101 MyDomain
4	192.168.100.0	membership	192.168.100.200
5	192.168.100.0	membership	192.168.100.201
6	192.168.100.100 MyDomain	containment	192.168.100.100 MyDomain
7	192.168.100.100 MyDomain	containment	192.168.100.101 MyDomain
8	192.168.100.200 MyDomain	containment	192.168.100.200
9	192.168.100.200 MyDomain	containment	192.168.100.201
10	Business app	dependency	C:\sample.ddm

After importing this Excel file, the topology appears as follows:



8. Add relationship attributes

Note: This use case is not widespread, but the Import from Excel Workbook job offers such capability.

Since many different types of links can be defined on the **relationships** tab in Excel, it is impossible to name columns with attribute names. For this purpose, the following notation is used:

<Attribute name>< relationship_attr_delimiter><Attribute value>

By default, for **relationship_attr_delimiter**, a pipe symbol ('|') is used.

The description definition for the link **dependency** from running_software to process looks like **description|The Business app depends from the Sample process.**

Now the **relationships** tab appears as follows:

	A	B	C	D
1	start	relation_type	end	
2	192.168.100.0	membership	192.168.100.100	
3	192.168.100.0	membership	192.168.100.101	
4	192.168.100.0	membership	192.168.100.200	
5	192.168.100.0	membership	192.168.100.201	
6	192.168.100.100 MyDomain	containment	192.168.100.100	
7	192.168.100.100 MyDomain	containment	192.168.100.101	
8	192.168.100.200 MyDomain	containment	192.168.100.200	
9	192.168.100.200 MyDomain	containment	192.168.100.201	
10	Business app	dependency	C:\sample.ddm	description The Business app depends from the Sample process'

If many attributes must be added, they must be defined in additional columns in the **dependency** row.

Note: On the **relationships** tab, no captions are needed for the attribute columns. If the column heading is present, these columns are treated as **comment** columns.

9. Convert attribute types to UCMDB attribute types

At the importing stage, each attribute is converted to the type defined in the UCMDB class model. This means that if an attribute is defined in UCMDB with a text value (for example, the attribute **port** in Service Address), but in the Excel file it has an integer value (for example, **5**), it will be converted to the corresponding type.

The following UCMDB attribute types are supported:

boolean	date	double	enumerations
float	integer	integer_list	long
string	string_list	xml	

Note: If the attribute cannot be converted to the type defined in UCMDB, it is skipped and you receive a warning in the UI.

Two list types exist in UCMDB: **integer_list** and **string_list**. To import such types, the value delimiters are intended. They are **integer_list_delimiter** and **string_list_delimiter** respectively. The default values are separated by a comma (','), but this can be changed to a job parameter.

If there is an attribute named **some_int_list** and it needs to be set using an integer list from 1 to 5, the cell in the **relationships** tab will look like:

```
some_int_list|1,2,3,4,5
```

- **Enumerate attribute types**

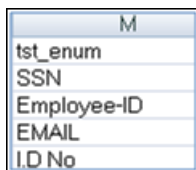
Enumeration data types are supported for attributes. The job assumes the enumeration has been entered in human readable form and performs a search of the internal integer representation used in UCMDB.

If a value is entered that is not an enumeration value, it is ignored and you receive a warning in the log.

Note: It is not supported to enter an integer as an enumeration value. Replacing an enumeration value with an integer is also not supported. When defining CI attributes using the **Enumeration/List** option in the Add/Edit Attribute dialog box, make sure you specify the type of input values as enumeration, instead of integer.

Because enumeration values are case sensitive in UCMDB, they are also case sensitive in Excel.

For example, if **SSN** in the image below had been written in lower case letters, **ssn**, the job would send an error message because it would not find the **ssn** string in UCMDB.



M	N
tst_enum	
SSN	
Employee-ID	
EMAIL	
I.D No	

Import from Excel Workbook Job

Note: The Import from Excel Sample job is similar to the Import from Excel Workbook job. It differs only by reference to the sample import file.

This section includes:

- ["Discovery Mechanism" below](#)
- ["Trigger Query" on the next page](#)
- ["Job Parameters" on the next page](#)
- ["Adapter" on page 172](#)
- ["Created/Changed Entities" on page 172](#)
- ["Discovered CITs" on page 173](#)

Discovery Mechanism

Each tab in the Excel file reflects a specific CI type. The CIT must be defined in the UCMDB data model prior to importing file content. If only out-of-the-box CITs are imported, you do not have to create the CITs because they already exist in UCMDB.

All attributes defined for a CIT must also already exist in UCMDB or the data will be rejected. Any special rules for attributes—such as data type, obligation, formatting, and so on—must also be acceptable by UCMDB for the data to be successfully imported into UCMDB.

The data type of the attribute —string, long, integer, boolean, and so on— depends on the UCMDB data model. You do not need to set attribute types manually. You must specify the attribute name in the document header line.

Discovery performs the following validations:

1. Verifies that the CITs on the tabs in the Excel spreadsheet exist in UCMDB.
2. Verifies that the attributes (the column names in the Excel spreadsheet) exist in UCMDB.
3. Checks the presence of key attributes on the Excel spreadsheet.
4. Processes all CITs that contain a **root_container** attribute after CITs that do not have this type of attribute. This helps to ensure that the parent CI is created before a contained CI.
5. Processes the **relationships** tab last to create relationships between CIs that do not use the containment (**container_f**) relationship.

For the relationship to be created, the keyed attributes of a CI must be used in the relationships tab.

6. Relation attributes also must exist in the UCMDB class model.

7. Verifies that the CITs on the tabs in the Excel spreadsheet exist in UCMDB.
8. Verifies that the attributes (the column names in the Excel spreadsheet) exist in UCMDB.
9. Checks the presence of key attributes on the Excel spreadsheet.
10. Processes all CITs that contain a **root_container** attribute after CITs that do not have this type of attribute. This helps to ensure that the parent CI is created before a contained CI.
11. Processes the **relationships** tab last to create relationships between CIs that do not use the containment (**container_f**) relationship.

For the relationship to be created, the keyed attributes of a CI must be used in the relationships tab.

12. Relation attributes also must exist in the UCMDB class model.

Trigger Query

The Import from Excel Workbook job has no trigger query. Therefore, you must manually add the Probe that imports the data. For details, see "Probe Selection Pane" in the *Data Flow Management section of the UCMDB Help*.

Job Parameters

Parameter	Description
file_name	The import file name. An absolute path accessible from the used probe must be used. For details on setting up this file, see "How to Set Up an Import File in Excel" on page 161 .
integer_list_delimiter	The delimiter used to handle values in the spreadsheet that are to be treated as the UCMDB data type integer_list .
string_list_delimiter	The delimiter used to handle values in the spreadsheet which would be mapped as the UCMDB data type string_list .
relationship_attr_delimiter	On the Relationship tab of the source file object, the linked attributes could be added. The default is attribute_name attribute_value (a pipe symbol is used between the attribute name and value). This should be aligned with actual data.

Adapter

- Input Query

Input CIT: discoveryprobemanager

Input query: Because the Import from Excel Workbook job's input CIT is Discovery Probe Gateway, there is no need to supply an input TQL query.

- Scripts Used

The following scripts are used to import data from an Excel workbook.

- import_from_excel.py
- xlsutils.py

Note: The Import from Excel Workbook job may also use library scripts supplied in the Auto Discovery content package.

Created/Changed Entities

Entity Name	Entity Type	Entity Description
Import from Excel Workbook	Job	Main importing job
Import from Excel Sample	Job	Sample job that imports the predefined sample import file
XLS_Parser	Adapter	Discovery adapter
import_from_excel.py	Script	Main import script
xlsutils.py	Script	Contains utility methods for class model validation and fetching objects from Excel worksheets
ciimports_for9.xls	Resource	Sample import file
poi-3.7-beta1-20100620.jar	Resource	Java library for working with Excel 97-2003 file format
poi-ooxml-3.7-beta1-20100620.jar	Resource	Java library for working with Excel 2007 file format

Entity Name	Entity Type	Entity Description
poi-ooxml-schemas-3.7-beta1-20100620.jar	Resource	Java library with XML schemas used in Excel 2007 files
geronimo-stax-api_1.0_spec-1.0.jar	Resource	Geronimo implementation of standard XML processing API (used by POI)
xmlbeans-2.3.0.jar	Resource	Library for accessing XML by binding it to Java types (used by POI)

Discovered CITs

- ConfigurationItem
- Managed Relationship

Note: To view the topology, see ["Topology" on page 158](#).

Troubleshooting and Limitations – Import From Excel Workbook Discovery

- **Problem:** The **Import from Excel Workbook** job compile time errors and problems working with the Excel files.

Solution: Verify that you have performed the instructions in the Prerequisite section of the this discovery. For details, see ["Prerequisite - Set up permissions" on page 159](#).

- **Problem:** Importing a CI with the qualifier **RANDOM_GENERATED_ID_CLASS**, but without defined reconciliation rules, leads to duplicating such CIs.

Solution: Currently this problem is not resolvable on the job side. This can only be resolved by defining reconciliation rules.

- **Problem:** The **Import from Excel Workbook** job date errors.

Solution: The date cannot be imported if it is represented in text format. This issue is not resolvable because of localization. Represent the date in numerical format.

- **Limitation:** The DFM Probe breaks down the imported data into 20 KB chunks. This can cause identification issues.
- **Problem:** The integration using the **Import from Excel Workbook** job fails when using the integration probe and the **Run In Separate Process** property is set to **true**.

Solution: In case of using integration service, for the Import from Excel Workbook integration, the JDBC driver should be specified in the classpath as well.

If your UCMDB server uses Oracle, you need to add **../lib/mcoracle.jar** to the **remoteJVMClasspath** field as Adapter Properties when configuring the integration point; If your UCMDB server uses MS SQL, add **../lib/mssqlserver.jar** to the **remoteJVMClasspath** field.

Chapter 14: Microsoft IAMUI Integration

This chapter includes:

Overview	176
Supported Versions	176
How to Push Data from UCMDB to IAMUI	177
Sample Integration Push Query	180
IAMUI Push Adapter	181
Troubleshooting - Microsoft IAMUI Integration	183

Overview

Microsoft Intelligent Asset Manager 2018 (IAM) Universal Inventory (IAMUI) is a free product from Microsoft and the successor of Microsoft Workspace. This milestone release includes a number of significant improvements including GDPR compliance, process simplification and focus on value delivery over licensing compliance. Universal Inventory turns raw inventory data in to meaningful insights.

The direct integration between Microsoft IAMUI and UCMDB, that is, the UCMDB push integration with IAMUI, allows UD to push discovered software inventory data into IAMUI.

How the UCMDB-IAMUI integration works

The UCMDB-IAMUI integration enables UD to push discovered software inventory data into IAMUI as follows:

1. Universal Discovery administrators deploy the IAMUI Push Adapter and create an integration point.
2. The UCMDB-IAMUI integration pushes discovered software inventory data into the SQL Server or Azure SQL Sever database for IAMUI.
3. IAMUI reads software inventory data from the SQL Server or Azure SQL Sever database and performs necessary software license validation.

Supported Versions

This integration solution supports the following versions:

- Microsoft IAMUI is installed
- UCMDB version 11.0 and later


How to Push Data from UCMDB to IAMUI


1. Prerequisites

- Microsoft IAMUI is already deployed and configured in your organization.
- You have the following credential related information for accessing the SQL server or Azure SQL server database for IAMUI: username, password, URL, port, and schema name.

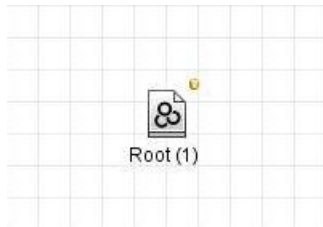
2. Set up UCMDB

Create a UDDI Registry CI named **iamui_trigger** and run the IAMUI Cleanup Job Trigger (or IAMUI Import Job Trigger) TQL query.

- a. In UCMDB UI, go to **Modeling > IT Universe Manager**.
- b. In the Related CIs tab window, click **New CI** .


The New CI dialog opens.
- c. In the Select CI Type pane, select **UDDI Registry**.
- d. In the Define New CI Properties pane, type **iamui_trigger** for the **Name** property.
- e. Specify other properties if necessary.
- f. Click **Save**.
- g. Go to the **Modeling Studio > Resources**.
- h. In the Resources tab window, select queries as resource type, expand **Root > Integration > IAMUI > Push > IAMUI Cleanup Job Trigger**.
- i. Double-click **IAMUI Cleanup Job Trigger**, and then in the right pane, click **Calculate Query Result Count** .




Make sure the returned result is greater than **0**, like the following:





3. Create an integration point

Define the integration point as follows:

- a. Log in to UCMDB as an administrator.
- b. Navigate to **Data Flow Management > Integration Studio**. A list of existing integration points is displayed.
- c. Click **New Integration Point** . The New Integration Point dialog box opens.
- d. Complete the **Integration Properties** and **Adapter Properties** fields as shown in the following table:

Field (*Required)	Description
Integration Properties section	
*Integration Name	Type the name (unique key) of the integration point.
Integration Description	Type a description of the current integration point.
Adapter	Click  and then select Third Party Products > IAMUI Push Adapter .
Is Integration Activated?	Select this option to indicate the integration point is active.
Adapter Properties section	
*Hostname/IP	The host name or IP address of the remote machine.
*Port	The port number of the SQL server. Default: 1433 .
*Credentials ID	<p>Credentials for the SQL Server user that you created in Generic Protocol.</p> <p>Click the Select Credential Id  button. The Generic Protocol is already selected in the Protocol pane and in the Credentials list.</p> <p>Note:</p> <ul style="list-style-type: none"> • Other credentials are not supported. • Ensure that the account has Read access to Microsoft IAMUI. <p>To configure the credential for this integration point,</p> <ol style="list-style-type: none"> i. Ensure the Generic Protocol is selected, and then click .


Field (*Required)	Description
	<p>The Generic Protocols Parameters dialog box opens.</p> <p>ii. Provide values for the following fields and click OK:</p> <ul style="list-style-type: none"> • Network Scope: Use the default value ALL. • User Label: Type a label for the credential. • Username: Provide the user name for the Microsoft SQL server user account. • Password: Click  and provide the password for the Microsoft SQL server user account. <p>iii. Click OK twice.</p>
*DB Type	Type of the database. Select one from the drop-down list. The available values include SQL Server and Azure SQL Server .
Azure Connection String	<p>(Applicable for Azure SQL Server only) The connection string used in Azure for connecting to Azure. The value is ignored if not Azure.</p> <p>For examples of the Azure connection string, see Connecting with SSL Encryption.</p>
*Data Source Id	The data source ID to indicate the source of the data push. It must end with xxx_<number> , for example, ucmdb_1 .
*Schema Name	The name of target DB schema to push data to.
*Data Flow Probe	The name of the Data Flow Probe/Integration service used to execute the synchronization from.
Additional Probes	Select additional probes to use when pushing to IAMUI in order to increase redundancy.

e. Click **Test connection**  to make sure there is a valid connection.

f. Click **OK**.


The integration point is created and its details are displayed.

Note:


- Your settings are not saved to the server until you click **OK**.
- You can edit this integration point by selecting the Integration Point name in the Integration Point pane and then clicking .

Push CI Data from UCMDB to the IAMUI database

1. Log in to UCMDB as an administrator.
2. Navigate to **Data Flow Management > Integration Studio**. A list of existing integration points is displayed.
3. Select the integration point that you created for IAMUI.
4. Run the job manually to see if the integration job works properly.

Go to the **Data Push** tab, click the **Full Synchronization**  button to push all the relevant data for the job.

Note: **Delta Synchronization** is not supported for the UCMDB-IAMUI integration.

5. Wait for the job to complete. Click the **Refresh**  button to check the status of the job.
6. When the job is completed, the job status becomes one of the following depending on the results:
 - Completed successfully
 - Failed
7. Click the **Statistics** tab to view the results; If any errors occur, click the **Query Status** tab and **Job Errors** tab for more details.

If the job completes successfully, you can view the UCMDB CI data in the SQL server or Azure SQL server database for IAMUI.

Related topics

["Troubleshooting - Microsoft IAMUI Integration" on page 183](#)

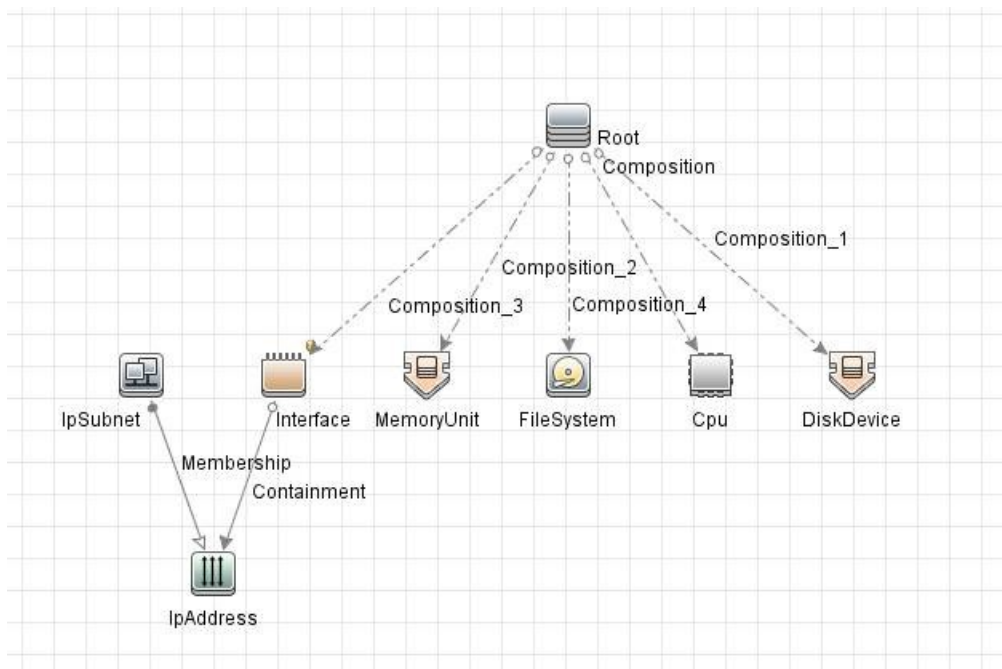
Sample Integration Push Query

All the TQL queries are located in **Modeling > Modeling Studio > Resources > Root > Integration > IAMUI > Push**:

- IAMUI Database Push 1.0
- IAMUI Device Push 1.0
- IAMUI Mail Server Push 1.0

- IAMUI Service Push 1.0
- IAMUI Software Push 1.0
- IAMUI User Push 1.0
- IAMUI VM Push 1.0
- IAMUI Web Application Push 1.0

The following image displays an example of IAMUI Device Push 1.0 to be pushed to IAMUI:



IAMUI Push Adapter

This integration uses the IAMUI Push Adapter (IAMUIPushAdapter), which is based on the push adapter of the generic adapter category.

Input CIT

destination_config

Parameters

Parameter	Detail
connectionString	The connection string used in Azure for connecting to Azure. Will be ignored if not Azure.
credentialsId	The credentials to be used.
dataSourceId	The dataSourceId for Microsoft IAM Universal Inventory (IAMUI).
dbtype	The type of database.
host	The host name or IP address of the remote machine.
port	The remote machine's connection port.
probeName	The values will be automatically replaced by the UCMDB UI.
schemaName	The name of the DB schema to push data to.

IAMUI Push Adapter Configuration Files

The adapter includes the following configuration files:

- **IAMUI Database Push 1.0.xml**. Defines field mapping between **Database** and **tblDatabaseServers**.
- **IAMUI Device Push 1.0.xml**. Defines field mappings between **Node**, **Cpu**, **MemoryUnit**, **DiskDevice**, **FileSystem**, **Interface** and **tblDevices**.
- **IAMUI Mail Server Push 1.0.xml**. Defines field mapping between **Mail Server** and **tblMailServers**.
- **IAMUI Service Push 1.0.xml**. Defines field mapping between **Windows Service** and **tblServices**.
- **IAMUI Software Push 1.0.xml**. Defines field mappings between **Installed Software** and **tblSoftware**.
- **IAMUI User Push 1.0.xml**. Defines field mapping between **OSUser** and **tblUsers**.
- **IAMUI VM Push 1.0.xml**. Defines field mappings between the following:
 - **Hypervisor** and **tblVmFarms**
 - **VM Node** and **tblVms**

- **VM Host** and **tblVMHosts**
- **IAMUI Web Application Push 1.0.xml**. Defines field mapping between **Web Application** and **tblMailServers**.
- **generic_adapter.xsd**. This XSD (XML Schema Definition) is referenced by the mapping file and can provide the valid value check for each setting.

For details about adapter configuration, see "Achieving Data Push using the Generic Adapter" in the Data Flow Management section of the UCMDB Help.

Troubleshooting - Microsoft IAMUI Integration

This section describes troubleshooting information for the UCMDB-IAMUI integration.

Troubleshooting

- **Problem:** Test Connection fails.

Solution: Check the **<DataFlowProbe>\runtime\log\WrapperProbeGw.log** for details about the failure. Usually the failure is caused by network issue or an invalid account is used.
- **Problem:** Some fields are empty in the SQL server or Azure SQL server database.

Solution: Check if the mappings are correct. To do so, in UCMDB UI, go to **Data Flow Management > Adapter Management**, in the Resource pane, locate **IAMUIPushAdapter**, and then expand **Configuration Files** and find the mapping file based on CI type .
- **Problem:** Missing Root in a TQL Query.

Solution: The integration TQL query must contain a Root Element (1 if it is a CI). Update your TQL query by renaming one of the query Elements to Root. Make sure your mapping XML file is updated accordingly.
- **Problem:** The `java.lang.OutOfMemoryError: Metaspace` error message appears in the **<UCMDBServer>\runtime\log\error.log** file.

Solution: To resolve the issue, do the following:
 - a. Open the **<UCMDBServer>\bin\wrapper-platform.conf** file using a text editor.
 - b. Locate the following line:

```
wrapper.java.additional.31=-XX:MaxMetaspaceSize=256m
```

- c. Update the value to 512m.
- d. Save the file.
- e. Restart the UCMDB server.

Logs

The integration job generates logs in the following folders:

- On the Data Flow Probe machine:
<DataFlowProbe>\runtime\log
- If using the integration service, on the UCMDB server:
<UCMDB_Server>\Integrations\runtime\log

Chapter 15: Microsoft SCCM/SMS Integration

This chapter includes:

Overview	186
Supported Versions	186
SMS Adapter	186
How to Populate the CMDB with Data from SCCM/SMS	188
How to Federate Data with SCCM/SMS	190
How to Customize the Integration Data Model in UCMDB	190
Predefined Query for Population Jobs	192
SCCM/SMS Integration Package	192
SMS Adapter Configuration Files	194
Troubleshooting and Limitations – Microsoft SCCM/SMS Integration	195

Overview

This document includes the main concepts, tasks, and reference information for integration of Microsoft System Center Configuration Manager (SCCM)/Systems Management Server (SMS) with Universal CMDB.

Integration occurs by populating the UCMDB database with devices, topology, and hierarchy from SCCM/SMS and by federation with SCCM/SMS supported classes and attributes.

According to UCMDB reconciliation rules, if a CI (in SCCM/SMS) is already mapped to a CI in the CMDB, it is updated; otherwise, it is added to the CMDB.

Microsoft System Center Configuration Manager/Systems Management Server are used by IT administrators to manage client computers and servers.

SCCM/SMS enable you to:

- Manage computers that roam from one location to another
- Track deployment and use of software assets, and use this information to plan software procurement and licensing
- Provide IT administrators and management with access to data accumulated by SCCM/SMS
- Provide scalable hardware and software management
- Manage security on computers running Windows operating systems, with a minimal level of administrative overhead

Supported Versions

Integration has been developed and tested on Universal CMDB version 8.03 or later, with SCCM versions 2007, 2012, and 2016, and SMS version 2003.

SMS Adapter

Integration with SCCM/SMS is performed using an SMS adapter, which is based on the Generic DB Adapter. This adapter supports full and differential population for defined CI types as well as federation for other CI types or attributes.

The SMS Adapter supports the following features:

- Full replicating of all instances of the selected CI types.
- Identifying changes that have occurred in SCCM/SMS, to update them in the UCMDB.
- Simulating the touch mechanism capabilities:

When a CI is removed from SCCM/SMS, it is physically deleted from the database and there is no way to report about it. The SMS Adapter supports a full synchronization interval. This means that the adapter transfers data for which the aging mechanism has been enabled, and provides the time interval to run a full synchronization that simulates the touch mechanism.

- Federation of selected CI types and attributes.

Out-of-the-box integration with SCCM/SMS includes population of the following classes:

- Node (some of the attributes are populated and some are federated)
- Layer2 connection
- Location that is connected to the node
- IP address
- Interface

In addition, the following classes can be defined as federated from SCCM/SMS:

- CPU
- File system
- Installed software
- Windows service

The following classes and attributes should be marked as federated by the SCCM/SMS adapter for the proper functionality of the Actual State feature of Service Manager:

- Classes
 - CPU
 - Installed software
 - Windows service
- Node attributes

- DiscoveredOsVendor
- DiscoveredModel
- Description
- DomainName
- NetBiosName

Note: Avoid marking the **LastModifiedTime** attribute as federated, as it may lead to unexpected results.



How to Populate the CMDB with Data from SCCM/SMS

This task describes how to install and use the SMS adapter.

This task includes the following steps:

- ["Define the SMS integration" below](#)
- ["Define a population job \(optional\)" on the next page](#)
- ["Run the population job" on the next page](#)

1. Define the SMS integration

- a. In UCMDB, navigate to **Data Flow Management > Integration Studio**.
- b. Click the **New Integration Point**  button to open the new integration point Dialog Box.
 - Click , select the Microsoft SMS adapter, and click **OK**.

Each out-of-the-box adapter comes predefined with the basic setup needed to perform integration with UCMDB. For information about changing these settings, see "Integration Studio Page" in the *Data Flow Management section of the UCMDB Help*.

- Enter the following information, and click **OK**:

Name	Description
Credentials	Allows you to set credentials for integration points. For credential information, see "Supported Protocols" in the <i>UCMDB Discovery and Integrations Content Guide - Supported Content</i> document.

Name	Description
Hostname/IP	The host name of the machine where the database of SCCM/SMS is running.
Integration Name	The name you assign to the integration point.
Is Integration Activated	Select this check box to create an active integration point. You clear the check box if you want to deactivate an integration, for instance, to set up an integration point without actually connecting to a remote machine.
Port	The port through which you access the MSSQL database.

- c. Click **Test connection** to verify the connectivity, and click **OK**.
- d. Click **Next** and verify that the following message is displayed: **A connection has been successfully created**. If it does not, check the integration point parameters and try again.



2. Define a population job (optional)

The Microsoft SMS adapter comes out-of-the-box with the **hostFromSMS Population** job, which runs the following predefined query: **hostDataFromSMS**. For details about this query, see ["Predefined Query for Population Jobs" on page 192](#). This job runs according to a default schedule setting.

You can also create additional jobs. To do this, select the Population tab to define a population job that uses the integration point you defined in ["Define the SMS integration" on the previous page](#). For details, see "New Integration Job/Edit Integration Job Dialog Box" in the *Data Flow Management section of the UCMDB Help*.

3. Run the population job

Activate the population job in one of the following ways:

- To immediately run a full population job, click . In a full population job, all appropriate data is transferred, without taking the last run of the population job into consideration.
- To immediately run a differential population job, click . In a differential population job, the previous population time stamp is sent to SCCM/SMS, and SCCM/SMS returns changes from that time stamp to the present. These changes are then entered into the UCMDB database.
- To schedule a differential population job to run at a later time or periodically, define a scheduled task. For details, see "Define Tasks that Are Activated on a Periodic Basis" in the *Administer section of the UCMDB Help*.

Note: the replicated CIs are controlled by the integration TQL that is used. You can create additional TQL queries that contain different topologies for use in other jobs.

How to Federate Data with SCCM/SMS

The following steps describe how to define the CI types that will be federated with SCCM/SMS.

1. In UCMDB, navigate to **Data Flow Management > Integration Studio**.
2. Select the integration point that you defined in ["Define the SMS integration" on page 188](#).
3. Click the **Federation** tab. The panel shows the CI types that are supported by the SMS adapter.
4. Select the CI types and attributes that you want to federate.
5. Click **Save**.

Note:

- CI types that populate UCMDB should not be selected for federation. Specifically, avoid federating node, IP address, interface, location, and Layer2, which populate UCMDB out-of-the-box.
- Other CI types can be used in federation only after the node data has been replicated to CMDB by the hostDataImport query. This is because the default reconciliation rule is based on node identification.

How to Customize the Integration Data Model in UCMDB

Out-of-the-box CIs for SCCM/SMS integration can be extended in one of the following ways:

To add an attribute to an existing CI type:

If the attribute you want to add does not already exist in the CMDB, you need to add it. For details, see "Add/Edit Attribute Dialog Box" in the *Modeling section of the UCMDB Help*.

1. Go to the **orm.xml** file as follows: **Data Flow Management > Adapter Management > SMS Adapter > Configuration Files > orm.xml**.

2. Locate the **generic_db_adapter.[CI type]** to be changed, and add the new attribute.
3. Ensure that the TQL queries that include this CI type have the new attribute in their layouts as follows:
 - a. In the Modeling Studio, right-click the node where you want to include the attribute.
 - b. Select **Query Node Properties**.
 - c. Click **Advanced Layout Settings** and select the new attribute.

For details about selecting attributes, see "Layout Settings Dialog Box" in the *Modeling section of the UCMDB Help*. For limitations on creating this TQL query, see ["Troubleshooting and Limitations – Microsoft SCCM/SMS Integration" on page 195](#).

To add a new CI Type to the Generic DB Adapter:

1. In UCMDB, create the CI Type that you want to add to the adapter, if it does not already exist. For details, see "Create a CI Type" in the *Modeling section of the UCMDB Help*.
2. Go to the **orm.xml** file as follows: **Data Flow Management > Adapter Management > SMS Adapter > Configuration Files > orm.xml**.
3. Map the new CI type by adding a new entity called **generic_db_adapter.[CI type]**.

For more details, see "The orm.xml File" in the *Developer Reference section of the UCMDB Help*.

4. Create queries to support the new CI types that you have added. Make sure that all mapped attributes are selected in the Advanced Layout settings:
 - a. In the Modeling Studio, right-click the node where you want to include the attribute.
 - b. Select **Query Node Properties**.
 - c. Click **Advanced layout settings** and select the new attribute.

For details about selecting attributes, see "Layout Settings Dialog Box" in *Modeling section of the UCMDB Help*. For limitations on creating this TQL query, see ["Troubleshooting and Limitations – Microsoft SCCM/SMS Integration" on page 195](#).

5. In UCMDB, navigate to **Data Flow Management > Integration Studio**.
6. Edit the SMS integration point to support the new CI type by selecting it either for population or for federation.
7. If the new CI type is for population, edit the population job that you created above.

Predefined Query for Population Jobs

The following TQL query is provided out-of-the-box if you use the Microsoft SMS adapter when you create an integration point:

- **hostDataFromSMS**. Imports nodes and their related data. Information also includes each node's IP address and interface.

SCCM/SMS Integration Package

This section includes:

- ["Transformations" below](#)
- ["SCCM/SMS Plug-in " on page 194](#)
- ["Reconciliation" on page 194](#)

Transformations

Following is the list of transformations that are applied to values when they are transferred to or from the SCCM/SMS database:

CMDB Class	Attribute	Transformation
windows	nt_servicepack	Represents number of the Windows service pack. SCCM/SMS DB: Service Pack 2 UCMDB: 2.0 Transformer: standard GenericEnumTransformer, mapped in the nt.nt_servicepack.transformer.xml file.
node	host_isdesktop	A Boolean value that determines whether a machine is a desktop or a server. SCCM/SMS DB: Workstation or Server UCMDB: true or false Transformer: standard GenericEnumTransformer, mapped in the node.host_isdesktop.transformer.xml file.

CMDB Class	Attribute	Transformation
node	host_os	<p>Represents the node's operation system.</p> <p>SCCM/SMS DB. Microsoft Windows XP Professional</p> <p>UCMDB. Windows XP</p> <p>Transformer. Standard GenericEnumTransformer, mapped in the node.discovered_os_name.transformer.xml file.</p> <p>If the SCCM/SMS operation system value is not listed in the transformer.xml file, the original value is sent to UCMDB.</p> <p>By default, only Windows operating systems are mapped.</p>
node	host_osinstalltype	<p>Represents the Windows OS edition.</p> <p>SCCM/SMS DB. Microsoft Windows XP Professional</p> <p>UCMDB. Professional</p> <p>Transformer. Standard GenericEnumTransformer, mapped in the host.host_osinstalltype.transformer.xml file.</p> <p>Note: The same column in the SCCM/SMS database is mapped to two different UCMDB attributes, using different transformers.</p>
disk device	name	<p>Represents the partition name.</p> <p>SCCM/SMS DB. C:</p> <p>UCMDB. C</p> <p>Transformer. standard AdapterToCmdbRemoveSuffixTransformer that removes the colon.</p>
interface	interface_macaddr	<p>Represents the MAC address of NIC.</p> <p>SCCM/SMS DB. AB:CD:EF:01:23:45</p> <p>UCMDB. ABCDEF012345</p> <p>Transformer. custom SmsMacAddressTransformer that removes the colons from the SCCM/SMS MAC address while making it compatible with the UCMDB MAC addresses.</p>

SCCM/SMS Plug-in

The **SmsReplicationPlugin** provides enhanced functions to those found in the Generic Database Adapter. It is called when:

- full topology is requested (**getFullTopology**) – this returns all the CIs that were found in the external SCCM/SMS database.
- topology layout is requested (**getLayout**)
- topology of changes is requested (**getChangesTopology**) – this returns only the CIs that are modified or added after a specific time. The topology of the changes is calculated as follows:
 - There is a specific date (**fromDate**) after which all changes are requested.
 - Most of the entities in the SCCM/SMS database contain a Timestamp column that contains the date and time of the last modification. This Timestamp column is mapped to the **root_updatetime** attribute of a CI. Currently, some entities do not contain any creation time information. The entities that have a timestamp column must be listed in the **replication_config.txt** file.
 - In the integration TQL query, the node CI is named **Root**.
 - Using the plug-in, the integration TQL query is dynamically modified so that each **Root** entity and all entities that are listed in the **replication_config.txt** file have an additional condition causing the value of the **root_updatetime** attribute to be greater than or equal to the **fromDate** value.
 - This modified TQL query is then used to obtain the data.

Reconciliation

The adapter uses the default reconciliation rule-based mapping engine.

SMS Adapter Configuration Files

The adapter includes the following configuration files:

- **orm.xml**. The Object Relational mapping file, which maps between SCCM/SMS database tables and columns, and UCMDB classes and attributes. Both CIs and links are mapped.
- **fixed_values.txt**. Used by the Generic DB Adapter to set the **ip_domain** of IP Address CIs to

DefaultDomain.

- **plugins.txt.** Contains configuration information for the Generic DB Adapter. Also defines three plug-ins that are used during replication: `getFullTopology`, `getChangesTopology`, and `getLayout`.
- **transformations.txt.** Contains the configuration for transformation of attribute values. For a list of the transformations, see ["Transformations" on page 192](#).
- **node.discovered_os_name.transformer.xml.** Mapping used by the transformer for the **host_isdesktop** attribute.
- **node.host_osinstalltype.transformer.xml.** Mapping used by the transformer for the **host_os** attribute.
- **host.host_osinstalltype.transformer.xml.** Mapping used by the transformer for the **host_osinstalltype** attribute.
- **nt.nt_servicepack.transformer.xml.** Mapping used by the transformer for the **nt_servicepack** attribute.
- **replication_config.txt.** Contains a comma-separated list of non-root CIs and relations types that have a **timestamp** condition in the SCCM/SMS database. This status condition indicates the last time the entity was updated.
- **reconciliation_types.txt.** Defines the CI types that are used for reconciliation.

For details on adapter configuration, see "Developing Generic Database Adapters" in the *Developer Reference section of the UCMDB Help*.

Troubleshooting and Limitations – Microsoft SCCM/SMS Integration

- Queries that are used in population jobs should contain one CI type that is labeled with a Root prefix, or one or more relations that are labeled with a Root prefix.

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 the Node CI Type, that graph node is labeled as Root and the resources are not labeled Root.

- The TQL graph must not have cycles.
- 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.
- Entities that are added in SCCM/SMS are sent as updates to UCMDB by the SMS Adapter during differential population.
- ID conditions on the integration TQL query are not supported.
- The TQL graph should contain only CI types and relations that are supported by the SCCM/SMS adapter.

Chapter 16: NetApp SANscreen/OnCommand Insight Integration

This chapter includes:

Overview	198
Supported Versions	198
Topology	198
How to Discover NetApp SANscreen (NetApp OnCommand Insight)	200
SANscreen Integration by WebServices Job	202
Adapter	202
Parameters	202
Integration Flow	202
SANscreen Adapter	203
Troubleshooting and Limitations – NetApp SANscreen Integration	204

Overview

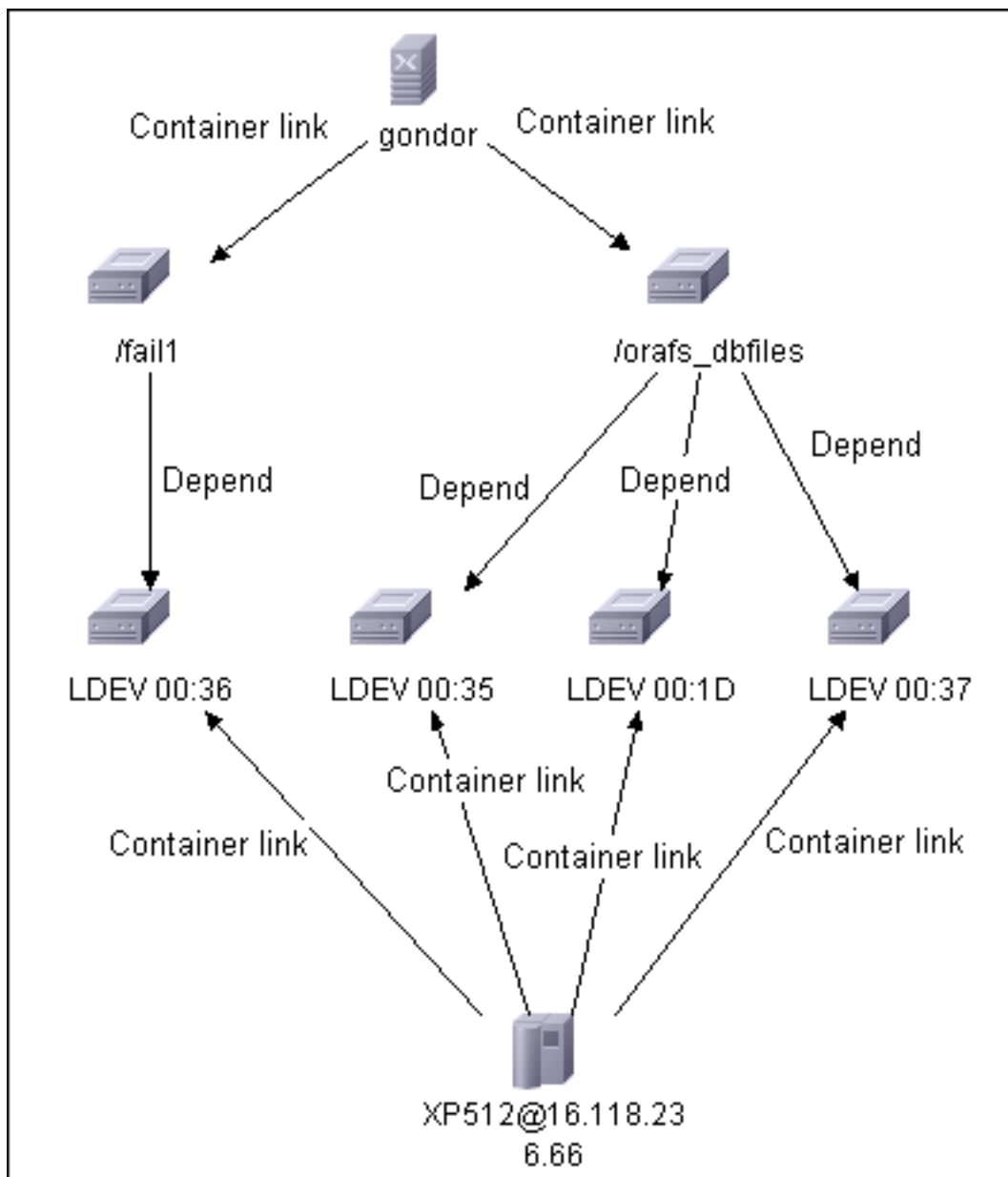
Integration between NetApp SANscreen and DFM involves a UCMDB initiated integration adapter on the SANscreen WebService API, and synchronizes devices, topology, and hierarchy of storage infrastructure in UCMDB. This enables Change Management and Impact Analysis across all business services mapped in UCMDB from a Storage point of view.

Supported Versions

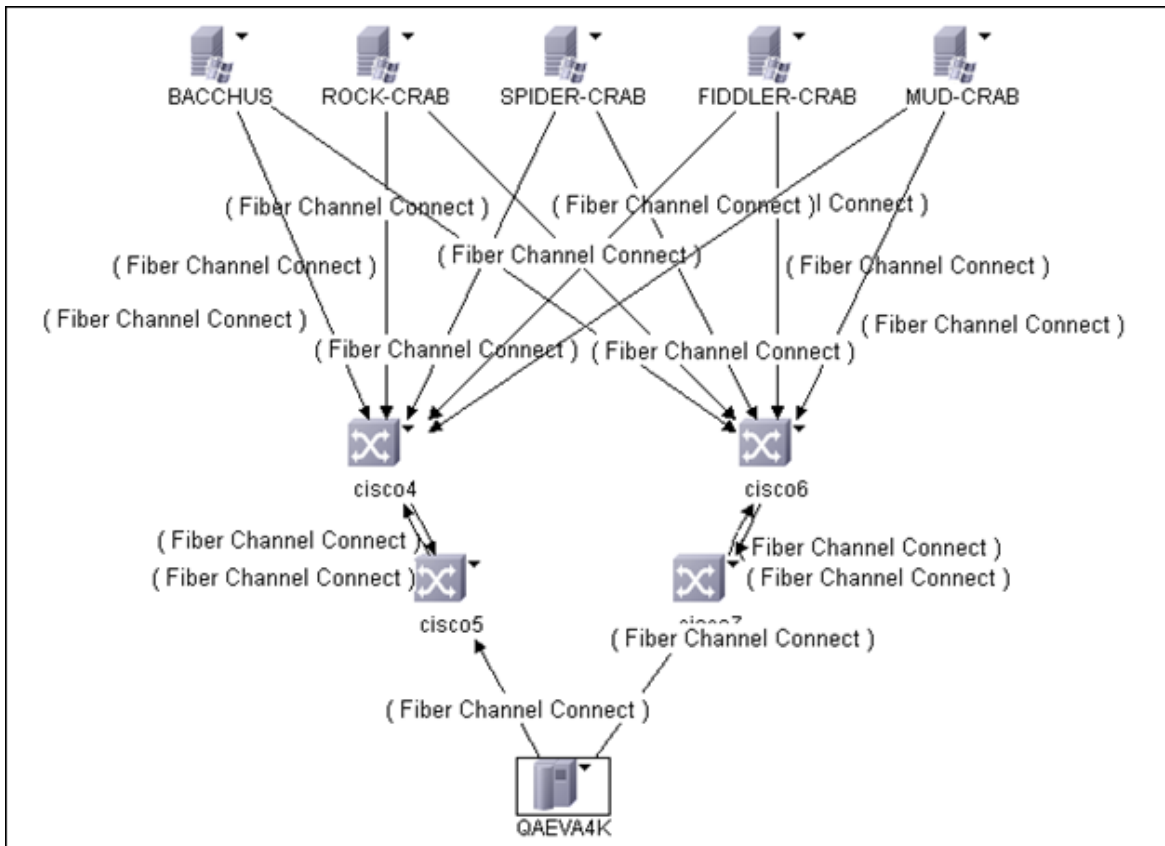
SANscreen integration supports version 5.1.2 (275) of NetApp SANscreen and version 6.2x, 6.3, and 6.4 of the product which has been renamed NetApp OnCommand Insight.

Topology

The following diagram illustrates the storage topology and shows the relationships between logical volumes on a storage array and those on servers:



The following diagram illustrates the SAN Topology and shows the fiber channel paths between storage arrays, switches, and servers:



How to Discover NetApp SANscreen (NetApp OnCommand Insight)

This task includes the following steps:

1. Prerequisite - Shell Connectivity

Ensure there is Shell connectivity to one or more nodes of the SANscreen domain.

For credential information, see "Supported Protocols" in the *UCMDB Discovery and Integrations Content Guide - Supported Content* document.

2. Prerequisite - Update Jar Files

- a. On the NetApp server, navigate to the following location: **<SANscreen root**

folder>\jboss\server\onaro\tmp<...\onaro-home.war>\WEB-INF\lib\java-sdk.jar

where **<SANscreen root folder>** is the location where you installed NetApp.

- b. Rename **java-sdk.jar** to **sanscreen_api.jar**.
- c. Stop the Data Flow Probe.
- d. On the Data Flow Probe, navigate to the following location: **<DataFlowProbe_Home>
\\root\lib\collectors\discoveryProbe\discoveryResources\SANscreen\sanscreen_api.jar**
where **<DataFlowProbe_Home>** is the location where you installed the Data Flow Probe.
- e. Replace the file on the Data Flow Probe with the renamed file from the NetApp server.
- f. Restart the Data Flow Probe.

3. Run the Discovery

- a. Run the **Range IPs by ICMP** job in order to discover the target IPs.
- b. Run the **TCP Ports** job in order to discover SANscreen WebService ports.

4. Define the integration point

In **Data Flow Management > Integration Studio**, define a new integration point:

- a. Provide a name and description
- b. Select the **NetApp SANscreen or On Command Insight** adapter and enter the required properties as follows:

Attribute	Description
ChunkSize	The number of CIs to pull from SANscreen/OnCommand Insight per query. The default is 1000.
Data Flow Probe	Select the name of the Probe on which this integration will run.
Trigger CI Instance	Select the IpServiceEndpoint at which the SANscreen/OnCommand Insight service is running.

A predefined job appears by default. Define a synchronization schedule if required. Jobs can also be run without a schedule.

- c. Save the job definition and then the integration point.
- d. Run a full synchronization for each job at least once.

SANscreen Integration by WebServices Job

Adapter

This job uses the **NetApp SANscreen/OnCommand** integration adapter.

Parameters

ChunkSize

Default: 1000

Integration Flow

The adapter works as follows:

1. Connect to the SANscreen WebService API using credentials from the SANscreen protocol.
2. Query for storage arrays and create STORAGE ARRAY CIs.
3. Query for logical volumes, fiber channel adapters (Host Bus Adapters), and fiber channel ports on each storage array and create LOGICAL VOLUME, HBA, and FC PORT CIs.
4. Query for fiber channel switches and create FC SWITCH CIs.
5. Query for fiber channel adapters and ports on each fiber channel switch and create HBA and FC PORT CIs.
6. Query for hosts/servers and create appropriate COMPUTER, WINDOWS, or UNIX CIs.
7. Query for logical volumes, fiber channel adapters (Host Bus Adapters), and fiber channel ports on each host/server and create LOGICAL VOLUME, HBA, and FC PORT CIs.
8. Query for paths between hosts/servers and storage arrays and add FCONNECT relationships between respective hosts/servers, switches, and storage arrays.
9. Query for logical volume mapping between logical volumes on hosts/servers and logical volumes

on storage arrays and add DEPEND relationships between respective hosts/servers and storage arrays.

SANscreen Adapter

This section contains information about the SANscreen adapter.

Input CIT

IpServiceEndpoint

Input Query



Triggered CI Data

Name	Value
ip_address	\${SOURCE.bound_to_ip_address}
port	\${SOURCE.network_port_number}

Used Script

SANscreen_Discovery.py

Discovered CITs

- Composition
- Containment
- Cpu
- Dependency
- FabricZoneSet

- FiberChannelZone
- Fiber Channel Connect
- Fibre Channel HBA
- Fibre Channel Port
- Fibre Channel Switch
- IpAddress
- LogicalVolume
- Membership
- Node
- Storage Array
- Storage Fabric
- Storage Processor
- Unix
- Windows

Global Configuration Files

None

Parameters

ChunkSize: The number of CIs to pull from SANscreen/OnCommand Insight per query. The default is 1,000.

Troubleshooting and Limitations – NetApp SANscreen Integration

Problem: Depending on the version of NetApp SANscreen or OnCommand Insight in use, discovery may fail with the following error message in the Probe wrapper logs:

```
SANscreen: Internal error. Details: java.lang.ClassCastException:  
com.sun.xml.messaging.saaj.soap.ver1_1.Message1_1Impl
```

Solution: In this case, replace

<DDM>\root\lib\collectors\discoveryProbe\discoveryResources\SANscreen\sanscreen_api.jar
with the corresponding file from the SANscreen server in use.

Chapter 17: NetApp OnCommand Insight (OCI) Pull Integration

This chapter includes:

Overview	207
Supported Versions	207
How to Discover NetApp OnCommand Insight (OCI)	207
NetApp OCI Integration Pull Adapter	208

Overview

NetApp OnCommand Insight (OCI, formerly known as SANScreen) is a storage resource-management system.

NetApp OnCommand Insight (OCI) pull integration is to populate data from NetApp OCI to UCMDB.

Note: Data Store can only be modeled as VMware Data Store in the NetApp OCI pull integration.

Supported Versions

The NetApp OnCommand Insight (OCI) pull integration supports NetApp OCI 7.1 and the REST API version 1.2.

How to Discover NetApp OnCommand Insight (OCI)

This task includes the following steps:

1. Prerequisite - Set up the HTTP protocol

Create or select an HTTP protocol. In the HTTP Protocol Parameters dialog box, the following parameters are required to fill:

- **Username.** The user name to access NetApp OCI.
- **Password.** The password to access NetApp OCI.
- **Protocol.** Select **https** or **http**. In general, NetApp OCI uses **https**.
- **Port Number.** Enter **443** for HTTPSs, or **80** for HTTP.

For credential information, see "Supported Protocols" in the *UCMDB Discovery and Integrations Content Guide - Supported Content* document.


2. Define the integration point

In **Data Flow Management > Integration Studio**, define a new integration point:

- a. Provide a name and description.
- b. Select the **NetApp OCI Pull Integration** adapter and enter the required properties as follows:

Attribute	Description
Credentials ID	The HTTP protocol to access the OCI server.
Hostname/IP	The host name or IP address of the OCI server.
Data Flow Probe	Select the proper Data Flow Probe that can access the NetApp OCI server.
Trigger CI Instance	Select the Discovery Probe Gateway that can access the NetApp OCI server.

A predefined job appears by default. Define a synchronization schedule if required. Jobs can also be run without a schedule.

- c. Save the job definition and then the integration point.
- d. Click  to run a full synchronization.

NetApp OCI Integration Pull Adapter

This section contains information about the **NetApp OCI Pull Integration** adapter.

Input CIT

Discovery Probe Gateway

Input Query



SOURCE

Used Script

- rest_cache.py
- rest_json.py

- `rest_requests.py`
- `oci_common.py`
- `oci_entities.py`
- `oci_pull.py`

Discovered CITs

- Composition (fcswitch, fcport)
- Composition (storagearray, fcport)
- Composition (storagearray, logical_volume)
- Composition (storagearray, physicalvolume)
- Computer
- Dependency (host_node, vmware_datastore)
- Dependency (storagepool, logical_volume)
- Fiber Channel Connect (fcport, fcport)
- Fibre Channel Port
- Fibre Channel Switch
- LogicalVolume
- Membership (storagefabric, fcswitch)
- Physical Volume
- Storage Array
- Storage Fabric
- Storage Pool
- Usage (storagepool, physicalvolume)
- VMware Datastore

Adapter Parameters

Parameter	Description
credentialsId	The HTTP protocol to access the OCI server.
host	The Host or IP address of the OCI server.

Chapter 18: ServiceNow Integration

This chapter includes:

Overview	212
Supported Versions	212
Push Integration with ServiceNow	212
How to Push Data from UCMDB to ServiceNow	213
Push Integration Mechanism	215
Sample Integration Push Query	216
Supported CITs	217
Population from ServiceNow	219
How to Populate UCMDB with Data from ServiceNow	219
Population Flow	221
Jython Scripts for Population	222
Mapping Files for Population Flow	223
Supported CITs	225
Troubleshooting and Limitations – ServiceNow Integration	226
Troubleshooting – ServiceNow Integration	226
Limitations – ServiceNow Integration	236

Overview

UCMDB-ServiceNow integration consists of two independent, bi-directional parts: **Data Push into ServiceNow** and **Population from ServiceNow**.

The jobs enable you to integrate CIs and relationships between UCMDB and ServiceNow. The integrations both use XML mapping frameworks that enable you to dynamically map CI types between UCMDB and ServiceNow, without requiring code changes.

- **Data Push into ServiceNow** provides the ability to push CIs and relationships from UCMDB to ServiceNow.
- **Population from ServiceNow** provides the ability to import CIs and relationships from ServiceNow into UCMDB.

Supported Versions

This integration solution supports pushing CIs to ServiceNow from Universal CMDB version 9.02 and later and populating UCMDB with CIs from Universal CMDB version 10.01 and later.

Push Integration with ServiceNow

This section includes:

How to Push Data from UCMDB to ServiceNow	213
Push Integration Mechanism	215
Sample Integration Push Query	216
Supported CITs	217
Pushing Additional CITs	217

How to Push Data from UCMDB to ServiceNow

1. Configure queries

The CIs and relationships to be pushed to ServiceNow have to be queried from UCMDB using TQL queries. Create integration type queries to query the CIs and relationships that have to be pushed to ServiceNow.

For the details of Push Query, you can refer to **UCMDB Help > Developer Reference > Creating Discovery and Integration Adapters > Developing Push Adapters > Build an Adapter Package**.

An example of such a query, [ServiceNowSampleQuery](#), is included with this integration package, and can be viewed in the Modeling Studio. For details on viewing queries in the Modeling Studio, see the *Modeling section of the UCMDB Help*.

2. Create XML mapping files

For every query created in the step above, create an XML mapping file with exactly the same name (case-sensitive) as the integration query in the following directory:

**<UCMDB>\UCMDBServer\runtime\fcmdb\CodeBase\ServiceNowPushAdapter\
mappings**

A sample mapping file for this integration, **ServiceNowSampleMapping.xml**, is provided out of the box with the package.

For more information about mapping files, see the *Developer Reference section of the UCMDB Help*.

3. Create the integration point

Define the integration point as follows:

- a. In UCMDB, go to **Data Flow Management > Integration Studio**.
- b. In the New Integration Point dialog box, enter a name and description for the integration point.

Ensure the **Is Integration Activated** option is enabled.

- c. In the Adapter field click the **Select Adapter**  button.


- d. In the Select Adapter list, select **Push to Service-Now** and click **OK**.
- e. In the Adapter Properties section, enter the following required properties:

Field	Value
Service-Now Domain	Domain name used to access the ServiceNow instance. Usually service-now.com .
Service-Now Instance	The ServiceNow instance being used. For the demonstration instance, enter demo .
Port	Default: 443 , for HTTPS.
Protocol	HTTP or HTTPS.
Proxy Server Name/IP	If an HTTP(S) proxy service is used to access the Internet, enter the proxy server name.
Proxy Server Port	HTTP(S) proxy server port.
Credentials ID	Define the ServiceNow instance user name and password in the Generic Protocol. For the demonstration website, use admin/admin . For credential information, see Supported Protocols in the <i>UCMDB Discovery and Integrations Content Guide</i> .
Retry Count	The retry count when the push failed. Default: 3 .
Retry Delay Seconds	The retry delay seconds when the push failed. Default: 5 .
Fail Bulk	Indicates whether to discard the whole bulk when there is inconsistent data. Default: True .
Insert Multiple	Indicates whether to use the Insert Multiple function of ServiceNow. Using this function can accelerate the push integration. Default: False . If you want to use this function, first activate the API in ServiceNow, and then regenerate all JAR files that need the API from the new WSDL. For details, see How to generate JAR files for ServiceNow SOAP APIs .
Insert	The Bulk size of the Insert Multiple.

Field	Value
Multiple Bulk Size	Default: 100 .
Data Flow Probe	Select the name of the Data Flow Probe to run this integration.

- f. Test the connection to the target CMDB server. If the connection fails, verify that the information provided is correct.
- g. Click **OK** to save the integration point.
- h. Add a new job definition to the integration point. Provide a name for the job definition and select the queries to use to synchronize data from UCMDB to ServiceNow. Define a synchronization schedule if required.

Note: Jobs can also be run without a schedule.

- i. Save the job definition, and then the integration point.
- j. Click  to run a full synchronization for each job at least once.

Push Integration Mechanism

The components responsible for the ServiceNow integration are bundled in the ServiceNow Integration package, **ServiceNow_Integration.zip**.

The integration mechanism works as follows:

1. The Integration job queries UCMDB for CIs and relationships.

When an ad-hoc job is run from the integration point in the Integration Studio, the integration receives the names of the integration queries defined in the job definition, for that integration point.

It queries UCMDB for the results of these queries (new, updated and deleted CIs and relationships) and then applies the mapping transformation according to the predefined XML mapping files for every TQL query.

It then pushes the data to the Data Flow Probes.

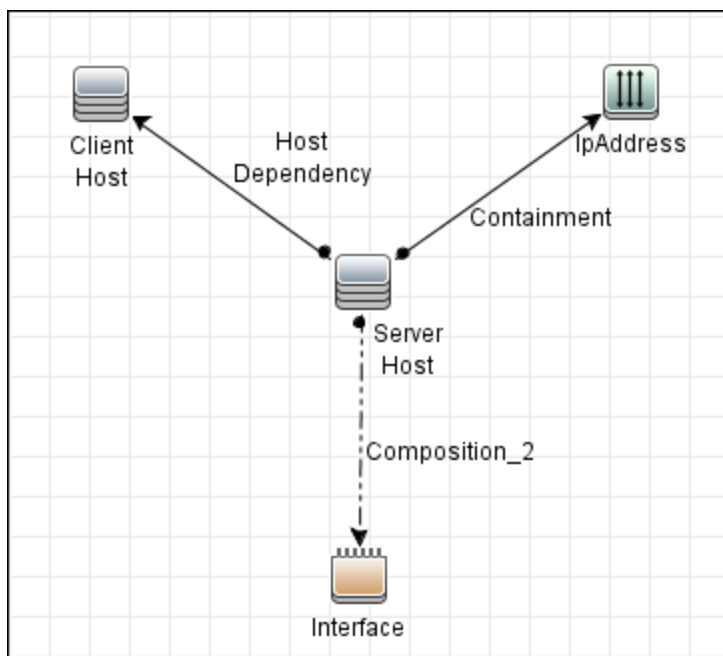
2. The integration job sends the data to ServiceNow.

Next, on the Data Flow Probe side, the integration process receives the CI and relationship data sent from the UCMDB server, connects to the ServiceNow server using the Direct Web Services SOAP API, and transfers the CIs and relationships.

Since the ServiceNow coalescing (CI reconciliation) mechanism is not available for the Direct Web Services API, a mapping of UCMDB CI IDs to ServiceNow SysIds is maintained on the Data Flow Probe. This mapping is used to update and delete CIs and relationships in ServiceNow.

Sample Integration Push Query

The following image displays a sample query to be pushed to ServiceNow:



Supported CITs

The following CIs and their relationships are supported out-of-the-box for push integrations:

- Apache Tomcat
- Apache
- DB2
- ESX Server
- Host
- IIS Web Server
- Interface
- Ip Address
- JBoss AS
- MySQL
- Net Device
- Oracle
- SQL Server
- Switch
- Unix Server
- Weblogic AS
- WebSphere AS
- Windows Server

Pushing Additional CITs

To push additional CITs, these CITs should be added to the mapping file. For details, see the *Developer Reference section of the UCMDB Help*.

Pushing additional CI Types requires corresponding JAR files to be generated using the WSDL URL for each CI Type. The WSDL URL can be generated using information from the **Direct Web Services** section at:

http://wiki.service-now.com/index.php?title=SOAP_Web_Service.

The resulting JAR files should be placed in the following directory on the UCMDB Data Flow Probe server:

<DataFlowProbe_Home>\runtime\probeManager\discoveryResources\Service-Now

Note: JAR files should be re-generated following CI Type updates in ServiceNow.

JAR files may be generated using WSDL2JAVA or other similar utilities. An example using this utility is at :

<http://roseindia.net/webservices/axis2/axis2-client.shtml>.

For CIs containing reference fields, the target data type of the attribute being mapped to a reference field should be set to the name of the reference table. For example, to populate the **Manufacturer** field on a **Windows Server** CI, the data type should be the reference table name **core_company** as shown below:

```
<source_ci_type name="nt" namespace="" query="">
  <target_ci_type name="cmdb_ci_win_server">
    <targetprimarykey>
      <pkey>ID</pkey>
    </targetprimarykey>
    <target_attribute name="virtual" datatype="boolean">
      <map type="direct" source_attribute="host_isvirtual"/>
    </target_attribute>
    <target_attribute name="category" datatype="String">
      <map type="direct" source_attribute="os_family"/>
    </target_attribute>
    <target_attribute name="short_description" datatype="String">
      <map type="direct" source_attribute="discovered_description"/>
    </target_attribute>
    <target_attribute name="correlation_id" datatype="String">
      <map type="direct" source_attribute="global_id"/>
    </target_attribute>
    <target_attribute name="model_number" datatype="String">
      <map type="direct" source_attribute="discovered_model"/>
    </target_attribute>
    <target_attribute name="name" datatype="String">
      <map type="direct" source_attribute="display_label"/>
    </target_attribute>
    <target_attribute name="os" datatype="String">
      <map type="direct" source_attribute="discovered_os_name"/>
    </target_attribute>
    <target_attribute name="model_id" datatype="cmdb_model">
      <map type="direct" source_attribute="discovered_model"/>
    </target_attribute>
    <target_attribute name="manufacturer" datatype="core_company">
      <map type="direct" source_attribute="discovered_vendor"/>
    </target_attribute>
  </target_ci_type>
</source_ci_type>
```

Population from ServiceNow

This section includes:

How to Populate UCMDB with Data from ServiceNow	219
Population Flow	221
Jython Scripts for Population	222
Mapping Files for Population Flow	223
Supported CITs	225

How to Populate UCMDB with Data from ServiceNow

1. Prerequisites

Install the **ServiceNow_pull_integration_patch.zip** patch on the Data Flow Probe. Note the address and port of the HTTP/HTTPS proxy server between the Probe machine and the ServiceNow instance.

To install the patch, extract the patch archive into the Probe installation folder. As a result, the folder **<probe_installation_folder>/jython/lib/suds** is created, and two files, **<probe_installation_folder>/jython/lib/httplib.py** and **<probe_installation_folder>/jython/lib/urllib2.py** should be updated. If UCMDB is on version 10.22 or later, make sure that these two files are not overwritten. You do not need to restart the Probe.

Patch content:

- **jython/lib/suds** – a library to work with WSDL-based web-services.
- **jython/lib/suds/mx/appender.py** – this script is modified to properly handle ServiceNow query parameters, such as **_use_view**, and **_encoded_query**. See line 191.
- **jython/lib/httplib.py**, **jython/lib/urllib2.py** – modified Jython 2.5.3 scripts, the following patch is applied:

<http://bugs.python.org/issue1424152>

2. Create XML Mapping Files.



The XML mapping file describes the topology that should be pulled from the ServiceNow instance and how it should be mapped to the UCMDB class model. Place one or more mapping files into the folder **<probe_installation_folder>\runtime\probeManager\discoveryConfigFiles\ServiceNow** on the Probe.

3. Create credentials for the ServiceNow Protocol.

Create entries only for the username and password fields.

4. Create the Integration Point.

Define the integration point as follows:

- In UCMDB, go to **Data Flow Management > Integration Studio**.
- Click the **New Integration Point**  button.
- In the New Integration Point dialog box, enter a name and description for the integration point.
- In the Adapter field click the **Select Adapter**  button.
- In the Select Adapter list, select **ServiceNow to UCMDB** and click **OK**.
- In the Adapter Properties section, enter the following required properties:

Field	Value
Chunk size	The number of record to pull from ServiceNow per web service call. Default: 200 . It is not recommended to change this value. Changing this value may affect the performance.
Credentials ID	Select the credential entries you created in step 3 for the ServiceNow Protocol.
HTTPs Proxy	Enter the IP address of the proxy server used to connect to the ServiceNow instance. If you are not using a proxy, leave this field empty.
Mapping File	The name of the XML mapping file you created in step 2, located in the \discoveryConfigFiles\ServiceNow folder. Leave blank if you want all the mapping files in this folder to be processed.

Field	Value
Remote JVM Class Path	The Remote JVM Class Path. Default: %minimal_classpath%;../content/lib/nnm/ucmdb_wrapper.jar;../runtime/probeManager/discoveryResources/IntegrationAPI.jar
Run in Separate Process	Indicated whether to run in a separate process. Default: true .
ServiceNow Instance URL	The address where the ServiceNow instance can be reached, including the protocol prefix and port. For example, https://demo.service-now.com .
Data Flow Probe	Select the Probe on which the integration will run.
Default owner name	The name of the owner to be used for CIs from population and federation when no owner is defined.
Trigger CI Instance	Select the DataFlowProbe CI, which refers to the same Probe as Data Flow Probe property above.

- g. Click **OK** to save the integration point. At this point, a default integration job definition is automatically created.

5. Define a synchronization schedule for the integration job, if desired.

Jobs can also be run manually, without a schedule.

6. Run full data synchronization.

Click  to synchronize all data.

Population Flow

The following stages comprise the flow of populating UCMDB with CIs from ServiceNow:

- Querying CIs

The integration job reads the mapping files to identify which CIs and relationships should be pulled from ServiceNow. It then connects to ServiceNow's Web Service API.

- a. A SOAP client is created for each CI type and relationship type defined in the mapping file, for example, **`https://demo.service-now.com/cmdb_ci_unix_server.do?wsdl`**.
- b. The `getRecords([query])` method is invoked to obtain instances of this CI type or relationship.

- Mapping data

CIs that are obtained from ServiceNow are mapped to UCMDB CIs.

- Querying relationships

The integration job uses **`https://demo.service-now.com/cmdb_rel_type.do?wsdl`** and **`https://demo.service-now.com/cmdb_rel_ci.do?wsdl`** to link CIs.

- Sending topology to UCMDB

The topology is sent from the Data Flow Probe to UCMDB.

Jython Scripts for Population

The following Jython scripts are used:

- **`mapping_interfaces.py`** – major interfaces involved in the integration workflow.
- **`replication.py`** – replication algorithm.
- **`mapping_implementation.py`** – basic implementation of some of the mapping interfaces.
- **`mapping_file_manager.py`** – abstract entity to represent mapping file.
- **`old_mapping_file_manager.py`** – mapping file parser.
- **`connection_data_manager.py`** – an entity that encapsulates pieces of data required to establish connection to ServiceNow instance.
- **`decorators.py`** – internal functions.
- **`pull_from_service_now.py`** – actual integration implementation script. All ServiceNow-related code is encapsulated inside the `ServiceNowSourceSystem` class.

Mapping Files for Population Flow

- **Query** – You can use a ServiceNow query to filter replicated CIs.

```
...  
<source_ci_type name="cmdb_ci_vcenter_datacenter" query="name=prod">  
...
```

For details on creating queries, see http://wiki.servicenow.com/index.php?title=Encoded_Query_Strings#Creating_Encoded_Query_Strings.

- **Foreign keys** – Sometimes CIs in ServiceNow do not have a relationship defined between them, but one CI keeps the `sys_id` of another CI in one of its attributes. For example, `cmdb_ci_network_adapter` has the attribute `cmdb_ci`, which contains the `sys_id` of the host where this adapter resides. To handle this situation in the mapping file, use the special `__Reference` type of link, and provide the name of the attribute with the foreign `sys_id`

```
<link source_link_type="__Reference" reference_attribute="cmdb_ci" target_  
link_type="composition"  
source_ci_type_end1="cmdb_ci_unix_server" source_ci_type_end2="cmdb_ci_  
network_adapter">  
  <target_ci_type_end1 name="unix"/>  
  <target_ci_type_end2 name="interface"/>  
</link>
```

By using the special `__Reference` link type, you can split one ServiceNow CI into multiple UCMDB CIs. For example, `cmdb_ci_esx_server` in ServiceNow should be transformed into a pair `ESX -> Hypervisor` in UCMDB. To do that, create a mapping file like this:

```
<?xml version="1.0" encoding="UTF-8"?>  
<integration>  
  <info>  
    <source name="ServiceNow" versions="" vendor="ServiceNow" />  
    <target name="UCMDB" versions="10.x" vendor="HP" />  
  </info>  
  <targetcis>  
    <source_ci_type name="cmdb_ci_esx_server">  
      <target_ci_type name="vmware_esx_server">  
        <target_attribute name="name">  
          <map type="direct" source_attribute="name" />  
        </target_attribute>  
      </target_ci_type>  
    </targetcis>  
  </integration>
```

```

</source_ci_type>
<source_ci_type name="cmdb_ci_esx_server">
  <target_ci_type name="hypervisor">
    <target_attribute name="name">
      <map type="direct" source_attribute="name" />
    </target_attribute>
    <target_attribute name="product_name">
      <map type="const" value="vmware_hypervisor" />
    </target_attribute>
  </target_ci_type>
</source_ci_type>
</targetcis>
<targetrelations>
  <link source_link_type="__Reference" reference_attribute="sys_id"
target_link_type="composition"
source_ci_type_end1="cmdb_ci_esx_server" source_ci_type_end2="cmdb_ci_esx_
server">
    <target_ci_type_end1 name="vmware_esx_server"/>
    <target_ci_type_end2 name="hypervisor"/>
  </link>
</targetrelations>
</integration>

```

Note that `cmdb_ci_esx_server` is mapped into two UCMDB CIs: `vmware_esx_server` and `hypervisor`. There is a relationship definition, that leverages the `__Reference` link type and `sys_id` as a foreign key. This means that when using `sys_id` as a reference attribute, the CI references itself. This allows you to map `cmdb_ci_esx_server` to two CIs and create a relationship between them.

- **Attribute mapping types:**

- `direct` – Takes the attribute value of a ServiceNow CI and sets it to a UCMDB CI attribute unchanged.
- `const` – Sets a UCMDB attribute to some constant value.

```

<source_ci_type name="cmdb_ci_esx_server">
  <target_ci_type name="hypervisor">
    <target_attribute name="name">
      <map type="direct" source_attribute="name" />
    </target_attribute>
    <target_attribute name="product_name">
      <map type="const" value="vmware_hypervisor" />
    </target_attribute>
  </target_ci_type>
</source_ci_type>

```

- **Link failure policy** – Sometimes relationships are mandatory and you do not want certain CIs to

enter UCMDB unless they have required relationships. To enforce this behavior, use `failure_policy` to define what to do, such as when a relationship cannot be created between a particular pair of CIs. Possible values are `exclude_end1`, `exclude_end2`, `exclude_both` and `ignore` (this is the default).

```
<link source_link_type="Defines resources for::Gets resources from" target_
link_type="composition"
source_ci_type_end2="cmdb_ci_vcenter_cluster" source_ci_type_end1="cmdb_ci_
esx_resource_pool"
failure_policy="exclude_end1" direction="reverse">
  <target_ci_type_end1 name="vmware_resource_pool"/>
  <target_ci_type_end2 name="vmware_cluster"/>
</link>
```

- **Link direction** – If you need relationship to go into the opposite direction than it is defined in ServiceNow, you can use the `direction="reverse"` attribute on a link mapping definition. Note that `target_ci_type_end1` and `target_ci_type_end2` should not be switched when this attribute is applied (see the previous example).
- **Validators** – Use validators when it is necessary to validate an attribute's values before sending topology to UCMDB. The following example filters out all `cmdb_ci_ip_address` instances where the attribute name is not actually, a valid IP address.

```
<target_attribute name="ip_netmask">
  <map type="direct" source_attribute="netmask" />
  <validators>
    <validator>validators.mandatory</validator>
    <validator>netutils.isValidIp</validator>
  </validators>
</target_attribute>
```

A validator's value is actually any Python function that returns a Boolean result. You can either use existing functions for validation, such as `netutils.isValidMac`, or write your own validation functions to get the most flexibility. The package contains a Jython script called `validators.py`, where you can add your custom validation functions.

Supported CITs

The population integration supports any CI and relationship that is available in ServiceNow.

Troubleshooting and Limitations – ServiceNow Integration

This section describes troubleshooting and limitations for the UCMDB-ServiceNow integrations.

- ["Troubleshooting – ServiceNow Integration" below](#)
- ["Limitations – ServiceNow Integration" on page 236](#)

Troubleshooting – ServiceNow Integration

This section describes troubleshooting for the UCMDB-ServiceNow integrations.

How to enable debug logs and how to read the logs

1. Go to **Data Flow Management > Adapter Management > Resources> Packages > ServiceNow_Integration > Scripts > pushToServiceNow.py**.
2. Change `DEBUGLEVEL = 0` to `DEBUGLEVEL = 5`.
3. The debug logs appear in the **probeMgr-adaptersDebug.log** file.

Below is a debug log example:

```
<2014-12-04 06:28:37,435> [DEBUG] [AdHoc:AD_HOC_TASK_PATTERN_ID-38-1417645713412] - =====
<2014-12-04 06:28:37,454> [DEBUG] [AdHoc:AD_HOC_TASK_PATTERN_ID-38-1417645713412] - Starting Push to Service-Now...
<2014-12-04 06:28:37,487> [DEBUG] [AdHoc:AD_HOC_TASK_PATTERN_ID-38-1417645713412] - [Push_to_SN logger] [DiscoveryMain] Service-Now URL:
<https://<instance name>.service-now.com:443>, using proxy <16.116.1.1:8080>
<2014-12-04 06:28:37,487> [DEBUG] [AdHoc:AD_HOC_TASK_PATTERN_ID-38-1417645713412] - [Push_to_SN logger]
*****
<2014-12-04 06:28:37,487> [DEBUG] [AdHoc:AD_HOC_TASK_PATTERN_ID-38-1417645713412] - [Push_to_SN logger] ***** addResult
*****
<2014-12-04 06:28:37,487> [DEBUG] [AdHoc:AD_HOC_TASK_PATTERN_ID-38-1417645713412] - [Push_to_SN logger] <?xml version="1.0" encoding="UTF-8"?>
<root>
```

```

<data>
  <objects>
    <Object mode="" name="cmdb_ci_win_server" operation="add"
mamId="fc5455b1751f6663b6607c930ca43c86" id="UCMDB%0Ant%0A1%0Ainternal_
id%3DSTRING%3Dfc5455b1751f6663b6607c930ca43c86%0A">
      <field name="virtual" key="false" datatype="boolean"
length="">true</field>
      <field name="category" key="false" datatype="String"
length="">windows</field>
      <field name="correlation_id" key="false" datatype="String"
length="">fc5455b1751f6663b6607c930ca43c86</field>
      <field name="model_number" key="false" datatype="String"
length="">VMware Virtual Platform</field>

      <field name="name" key="false" datatype="String"
length="">ddmivm40</field>
      <field name="os" key="false" datatype="String" length="">Windows
2008 R2</field>
    </Object>
  </objects>
  <links />
</data>
</root>

```

4. Check the **probeMgr-adaptersDebug.log** file to make sure that the mappings are created properly.

The push integration starts with the keyword Starting Push to Service-Now.... The XML results in the log are the mapping results based on the TQL and mapping file that are used by the ServiceNow push integration job.

How to set the default mapping

1. Go to **Data Flow Management > Adapter Management > Resources> Packages > ServiceNow_Integration > Configuration Files > ServiceNowPushAdapter/push.properties**.
2. Set the value of **mappingFile.default** (the default value of **mappingFile.default** is **ServiceNowSampleMapping**).

Note:

- The default mapping file is used only when there is no corresponding mapping file for the push TQL. The name of the mapping file should be the same as that of the TQL; otherwise the TQL

will use the default mapping file.

- Make sure that the default mapping file exists, and do not include the **.xml** extension in the name of the default mapping file. Otherwise, it may cause unexpected issues.

Mapping file syntax

For details about the mapping file syntax, see the *Create Mappings*, *Mapping File Reference* and *Mapping File Schema* sections in the *Developing Push Adapters* Chapter of the *Developer Reference section of the UCMDB Help*.

Tip:

- Set the **datatype** to **datetime** in the mapping file if the datetime type attribute needs to be pushed to ServiceNow.
- Set the **map type** to **constant** in the mapping file if the constant attribute needs to be pushed to ServiceNow.
- Below is an example of the mapping file:

```
<target_attribute name="first_discovered" datatype="datetime" >  
  <map type="direct" source_attribute="create_time" />  
</target_attribute>  
<target_attribute name="last_discovered" datatype="String" >  
  <map type="constant" value="2015-08-13 00:55:29" />  
</target_attribute>
```

How to delete the ID mapping for the fresh integration

Before performing the UCMDB-ServiceNow push integration, you may want to do a testing to push CIs from UCMDB to ServiceNow. Then in order to push these CIs to ServiceNow again without creating a new integration point, you must delete the ID mapping for the fresh integration.

Do the following:

1. Open the JMX Console, enter **removeIdMappingsOfDataStore** in the quick search field and click the link that appears.
2. In the **Value** box for the parameter **customerID**, enter the customer ID.
3. In the **Value** box for the parameter **dataStore**, enter the ServiceNow integration point name.
4. Click **Invoke**.

Note: This is only for the testing purpose and should not be used in the production environment; otherwise it will cause duplicate CIs in ServiceNow.

How to validate a successful connection

The following problems may cause the connection issue:

- **Problem:** The network connection fails.

Below is a log example:

```
<2015-10-23 12:01:02,926> [DEBUG] [AdHoc:AD_HOC_TASK_PATTERN_ID-84-1445572778467] -  
Traceback (most recent call last):  
File "pushToServiceNow", line 598, in processCIs  
File "pushToServiceNow", line 289, in insert  
File "pushToServiceNow", line 281, in retryHandler  
AxisFault: org.apache.axis2.AxisFault: Connection timed out: connect
```

Solution: Review the network connectivity.

- **Problem:** The configuration for the integration point is incorrect. For example, the credential is invalid.

Below is a log example (invalid credential):

```
<2015-10-23 13:32:16,476> [DEBUG] [AdHoc:AD_HOC_TASK_PATTERN_ID-114-1445578335150] - Error while executing:insert  
Traceback (most recent call last):  
File "pushToServiceNow", line 274, in retryHandler  
AxisFault: org.apache.axis2.AxisFault: Transport error: 401 Error:  
Unauthorized
```

Solution: Review the configuration, such as the credential and proxy, in the ServiceNow configuration panel.

- **Problem:** There are no corresponding JAR files for CI type in ServiceNow.

Below is a log example:

```
<2015-10-23 13:35:20,994> [WARN ] [AdHoc:AD_HOC_TASK_PATTERN_ID-123-1445578520682] - [DiscoveryMain] Exception: <  
Traceback (most recent call last):  
File "pushToServiceNow", line 884, in DiscoveryMain  
File "pushToServiceNow", line 634, in processCIs
```

```
Exception: java.lang.Exception: [processCIs] [getStub] [getStub] Unable to
get SN API stub for table <cmdb_ci_linux_server>
```

Solution: Regenerate the JAR files. For details, see ["How to generate JAR files for the ServiceNow SOAP APIs" on the next page.](#)

- **Problem:** The HTTPS certificate of the ServiceNow host, usually on non-cloud or standalone ServiceNow instances, is invalid or expired.

Below is a log example:

```
<2015-10-23 13:30:39,288> [DEBUG] [AdHoc:AD_HOC_TASK_PATTERN_ID-107-
1445578236621] -
Traceback (most recent call last):
File "pushToServiceNow", line 598, in processCIs
File "pushToServiceNow", line 289, in insert
File "pushToServiceNow", line 281, in retryHandler
AxisFault: org.apache.axis2.AxisFault:
sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to find
valid certification path to requested target
```

Solution: Download the certificate and import the certificate into the Data Flow Probe Trusted Store. To do so,

- Download the HTTPS certificate of the ServiceNow host. Perform the following steps:
 - Open the Microsoft Internet Explorer and go to **https://<ServiceNowHost>/** where **<ServiceNowHost>** is the host name of the running ServiceNow.
 - Click the **Lock** button on the **Address** bar, and then click **View certificates**.
 - In the Certificate dialog box, click the **Details** tab, and then click **Copy to File**.
 - In the Certificate Export Wizard dialog box, click **Next**.
 - Select the format **DER encoded binary X.509 (.CER)**.
 - Type a certificate file name, for example, **snow_host.cert**, and then save it to a temporary location.
 - Click **Finish**. You can see the **The export was successful** message.
- Locate the JRE security folder. By default it is located at **<DataFlowProbe installation folder>\bin\jre\lib\security**.
- Back up the **cacerts** file by copying it to another folder.
- Import the previously created certificate file **snow_host.cert** into the Data Flow Probe Trusted

Store.

Open a Command Prompt window and run the following commands on the local Data Flow Probe:

```
cd <DataFlowProbe installation folder>\bin\jre\bin  
keytool.exe -import -storepass <truststore pass> -keystore <DataFlowProbe  
installation folder>\bin\jre\lib\security\  
cacerts -trustcacerts -file <the path of the certificate file snow_  
host.cert>
```

- e. In the Command Prompt window, when the message **Trust this certificate?** appears, enter **yes**.
- f. Restart the Data Flow Probe service.

Tip: SoapUI tool is helpful to test the connection and to see if data is inserted into ServiceNow. This tool is a free and open source cross-platform Functional Testing solution. You can download it from <http://www.soapui.org/downloads/soapui/open-source.html>.

How to generate JAR files for the ServiceNow SOAP APIs

When customization is made in ServiceNow CI types (CITs) during this integration, the JAR files need to be re-generated. This is a limitation from the ServiceNow side.

You must create JAR files for each ServiceNow CIT that needs to be pushed.

To generate JAR files for the ServiceNow SOAP APIs, do the following:

1. Prerequisites

- a. Download the Apache Axis2 version 1.5.4 installation package **axis2-1.5.4-bin.zip** from <https://archive.apache.org/dist/axis/axis2/java/core/1.5.4/> and extract it to a temporary location, for example, to **C:\axis2-1.5.4**.

Note: Make sure to use the exact version 1.5.4 of Axis2. The JAR files that are generated by the latest version of Axis2 cannot work with the version of Axis2 provided with the Data Flow Probe.

- b. Download the Apache Ant version 1.8.4 installation package **apache-ant-1.8.4-bin.zip** from <http://archive.apache.org/dist/ant/binaries/> and extract it to a temporary location, for example, to **C:\apache-ant-1.8.4**.
- c. Download and install Java SE Development Kit (JDK) 8u65 from

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>.

- d. Set the following System environment variables:

```
JAVA_HOME = <JDK installation directory>  
AXIS2_HOME = <axis2-1.5.4 installation directory>
```

Take the following for example:

```
JAVA_HOME = C:\Program Files\Java\jdk1.8.0_65  
AXIS2_HOME = C:\axis2-1.5.4
```

Note: The JDK version in **JAVA_HOME** should be the same as the Data Flow Probe JRE version. Java 8 generated stubs will only work with the UCMDB 10.21 Data Flow Probe. For older Data Flow Probe versions with Java 7 JRE included, use Java 7 JDK instead.

- e. Make sure that the **Use unique targetNamespace for WSDL definition** option in ServiceNow is cleared.
- Go to **<https://<instance name>.service-now.com>**.
 - In the **Filter** search box, enter **Web Service**, and then click the link that appears.
 - In the right-hand pane, make sure that the **Use unique targetNamespace for WSDL definition** option is cleared.

2. Generate source Java files from the WSDL for ServiceNow CITs

Take the Business Service CIT (cmdb_ci_service) for example.

- a. Run the following commands in the Command Prompt window.

```
cd C:\axis2-1.5.4\bin  
  
C:\axis2-1.5.4\bin\wsdl2java.bat -uri https://<instance name>.service-  
now.com/cmdb_ci_service.do?WSDL.
```

- b. (Optional) The above commands may fail (you may get the connection timeout or 401 error) because of proxy settings. In such situations, you can download the WSDL file locally. Perform the following steps:
- Open Microsoft Internet Explorer and go to **https://<instance name>.service-now.com/cmdb_ci_service.do?WSDL**.
 - Right-click the document that appears, and then select **View source**.

- iii. Copy the contents, paste them in a text editor, and then save this file as **cmdb_ci_service.do** in the **C:** drive.
- iv. Go to the directory where **bin** exists and run the following commands in the Command Prompt window:

```
cd C:\axis2-1.5.4\bin\  
C:\axis2-1.5.4\bin\wsdl2java.bat -uri C:\cmdb_ci_service.do -o  
C:\temp\files
```

Note: The **-o** flag is used to store the generated folder in the directory that you want.

- v. The normal run shows the following:

```
C:\axis2-1.5.4\bin\wsdl2java.bat -uri C:\cmdb_ci_service.do -o  
C:\temp\files  
Using AXIS2_HOME: C:\axis2-1.5.4\  
Using JAVA_HOME: C:\Program Files\Java\jdk1.8.0_65  
Retrieving document at 'C:\cmdb_ci_service.do'.  
C:\axis2-1.5.4\bin
```

- c. After running the above commands, you will see the following:
 - The **src** folder
 - The **build.xml** file
- d. In the Command Prompt window, go to the directory where **src** and **build.xml** exist and run the **ant** command to generate JAR files from the source Java files as follows:

```
cd C:\temp\files  
C:\temp\files>C:\apache-ant-1.8.4\bin\ant
```

3. Results

You will receive the **BUILD SUCCESSFUL** message after completion. The JAR file will be generated under the **..\build\lib** folder.

4. Copy the generated JAR files to Data Flow Probe under the following directory:

<DataFlowProbe installation folder>\runtime\probeManager\discoveryResources\Service-Now

5. Restart the Data Flow Probe.

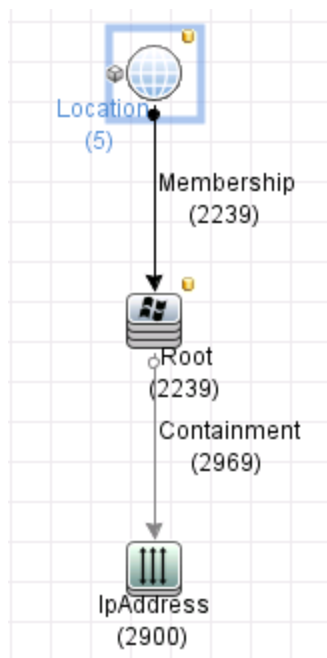
Cannot push the CI attribute values to the related CIs

Problem:

CI attribute values cannot be pushed to the related CIs (parent or child CIs) when the number of calculated TQL results is larger than the default chunk size (1000).

Workaround:

1. Rename a CIT to Root in TQL. For example, rename the Window CI to Root as follows.



2. Enable the instance-based push in the **ServiceNowPushAdapter.xml** file as follows:

```
<adapterInfo>
<adapter-capabilities>
  <support-replication-data>
    <target>
      <instance-based-data>true</ instance-based-data>
    </target>
  </support-replication-data>
</adapter-capabilities>
```

UCMDB to ServiceNow Push creates duplicate records instead of updating the existing records

Problem:

When you populate data from ServiceNow to UCMDB and then push data back to ServiceNow, duplicated records are created on the ServiceNow side.

Workaround:

The mapping file of the ServiceNow push integration should include the mapping `data_externalid > sys_id`. For example:

```
<source_ci_type_tree name="nt">
  <target_ci_type name="cldb_ci_computer">
    <target_attribute name="sys_id" datatype="String" >
      <map type="direct" source_attribute="data_externalid" />
    </target_attribute>
```

Reference fields cannot be populated directly from ServiceNow to UCMDB

Problem:

Reference fields cannot be populated directly from ServiceNow to UCMDB.

Workaround:

When creating a mapping for a **reference** field, in addition to the **sys_id**, the response element name for the display value field should be prefixed with **dv_**. For example, the **reference** field **u_service_manager** in ServiceNow should be **dv_u_service_manager** in the mapping file as follows:

```
<target_attribute name="u_service_manager">
  <map type="direct" source_attribute="dv_u_service_manager" />
</target_attribute>
```

Limitations – ServiceNow Integration

This section describes limitations for the UCMDB-ServiceNow integrations.

Push integration

- **Limitation:** The integration mapping file only allows mapping concrete CITs and relationships to the CITs and relationships in ServiceNow. That is, a parent CIT cannot be used to map its children CIs.
- **Limitation:** Since this adapter uses the ServiceNow Direct Web Services API, which does not support CI coalescing (reconciliation), if some CIs being pushed from UCMDB are already present in the ServiceNow CMDB, before the integration with UCMDB is installed, and if those CIs are (a) also in UCMDB; and (b) pushed into ServiceNow by the integration, those CIs are duplicated. (This is because UCMDB does not know these CIs are already in the ServiceNow CMDB.) After the adapter is installed, UCMDB keeps track of the CIs it pushes to ServiceNow, to prevent duplication.
- **Limitation:** ServiceNow Web Service Import Sets are currently not supported.

Population integration

- **Limitation:** Data replication in chunks is not supported.
- **Limitation:** Only full synchronization is supported (not delta synchronization).
- **Limitation:** Integration only works with Jython 2.5.3 and later.

Chapter 19: ServiceNow Integration Using Enhanced Generic Adapter

This chapter includes:

Overview	239
Supported Versions	239
ServiceNow Prerequisites	239
Out of the Box Connector Update Set	240
Setting up ServiceNow for the UCMDB Integration	243
Enable the Multiple Insert Plug-in in ServiceNow	243
Create Inbound Services for Class Models	243
Create Transform Maps between Staging Tables and Target Tables	244
Create a Role with Minimal Rights for the Integration	245
Push Integration with ServiceNow	247
How to Push Data from UCMDB to ServiceNow	247
Push Integration Mechanism	251
How to Enable Web Services Security (WS-Security, WSS) for the Push Integration	252
CI Reconciliation	255
Sample Integration Push Query	256
Supported CITs	257
Pushing Additional CITs	257
Population from ServiceNow	258
How to Populate UCMDB with Data from ServiceNow	258
Population Flow	260
Sample Integration Population Query	261
Supported CITs	263
External Class Model Flow	263
Incremental Update	264
Connectors	264
Performance	265

Configuration	265
adapter.properties Configuration	265
ServiceNowConfig.xml Configuration	268
Create a Custom coalesce Field instead of Using the Correlation ID Field	270
ServiceNow Changes	270
Adapter Changes	272
ServiceNow Generic Adapter Enhancements in CP27	272
Changes in ServiceNowConfig.xml	275
Topology Population Flow	281
Troubleshooting and Limitations – ServiceNow Integration Using Enhanced Generic Adapter ...	285
Troubleshooting – ServiceNow Integration Using Enhanced Generic Adapter	286
Limitations – ServiceNow Integration Using Enhanced Generic Adapter	289

Overview

UCMDB-ServiceNow integration using the enhanced generic adapter supports Data Push into ServiceNow and Population from ServiceNow.

The ServiceNow enhanced generic adapter is based on the Generic Adapter Framework. For more information about developing Generic Adapter, see the *Developer Reference section of the UCMDB Help*.

The jobs enable you to integrate CIs and relationships between UCMDB and ServiceNow. The integration uses XML mapping frameworks that enable you to dynamically map CI types between UCMDB and ServiceNow, without requiring code changes. You can use Graphic Mapping Tool and speed up the mapping file creation. It has also capability of ServiceNow class model retrieval.

- Data push into ServiceNow provides the ability to push CIs and relationships from UCMDB to ServiceNow.
- Data population from ServiceNow provides the ability to import CIs and relationships from ServiceNow into UCMDB.
- External class model retrieval allows a user to use a Graphical Mapping Tool to create mapping files used by the adapter.

Supported Versions

This integration solution supports pushing and populating CIs to/from ServiceNow from Universal CMDB version 10.22 and higher. ServiceNow is supported from Fuji release and higher.

Note: In UCMDB 10.22, this integration does not completely support the Graphic Mapping Tool. You can edit mapping files in their source XML files.

ServiceNow Prerequisites

This integration requires:

- Enable the multiple insert plug-in.
- Create Inbound Services for all class models that you want to integrate.

- Create the Transform maps between the staging tables and target tables.
- Create a role with minimal rights to serve the integration.

The ServiceNow enhanced generic adapter is coming with a configuration package that will configure all prerequisites for ServiceNow side (except creating an integration user). If you want, you can set up those settings manually, by following instructions in "[Setting up ServiceNow for the UCMDB Integration](#)" on page 243.

Out of the Box Connector Update Set

Along with this connector, there is an Update Set that can be loaded into ServiceNow. The update set will create a global application to serve this integration. The Update Set contains the following:

Object type	Details
Application menu	<p>Application menu holding the structure of the UCMDB–ServiceNow Integration application:</p> <ul style="list-style-type: none">• Create New Inbound Service• List all inbound services• List all transform maps• Business rules• Contact support
Inbound services	<p>Inbound services starting with <code>ucmdb_integ_*</code> for the following class models:</p> <ul style="list-style-type: none">• AIX Server• HP-UX Server• Windows server• Linux Server• Unix Server• Computer• Network Adapter• IP Address• File System• Storage Device• Business Service• Application• Printers

Object type	Details
	<ul style="list-style-type: none"> • ESX Server • Web Server • Database • Load Balancer • Network Gear • Cluster Virtual IP • Cluster Resource • Cluster Node • Windows Cluster • Cluster • CI Relationship
Transform maps	Transform maps for the inbound services listed below
Role	User role for integration with minimal rights to create update CIs and load OOB and external class models.
ACLs	<ul style="list-style-type: none"> • A create, write and read ACL for each created inbound service • A read ACL for the following sys tables to be able to load class models and transform maps. <ul style="list-style-type: none"> ◦ sys_db_object ◦ sys_dictionary ◦ sys_import_set_row ◦ sys_transform_entry ◦ sys_glide_object

To load the XML file in the **<DataFlowProbe_Home>\runtime\probeManager\discoveryResources\ServiceNowGenericAdapter\UCMDB_SNOW_update.zip** folder, do the following:

1. Log in to ServiceNow as an administrator.
2. Go to **System Update Sets > Retrieved Update Sets**.
3. Click **Import Update Sets from xml** and load the XML file.
4. Run a preview of the imported Update Set to detect any collisions with existing configuration.
5. Solve the exception if needed and then click **Commit Update Set**.

To cover additional class models that are not included in this package, see ["Create Inbound Services for Class Models"](#).

Setting up ServiceNow for the UCMDB Integration

This section includes:

Enable the Multiple Insert Plug-in in ServiceNow	243
Create Inbound Services for Class Models	243
Create Transform Maps between Staging Tables and Target Tables	244
Create a Role with Minimal Rights for the Integration	245

Enable the Multiple Insert Plug-in in ServiceNow

To enable the multiple insert plug-in:

1. Log in to ServiceNow as an administrator.
2. Go to **System Definition > Plugins**.
3. Locate **Insert Multiple Web Service** and click it.
4. If the status is inactive, click **Activate/Upgrade**. This action will activate the support for insertMultiple in SOAP.

Create Inbound Services for Class Models

Web service import sets provides a web service interface for import set tables. By default, this type of web service will transform the incoming data synchronously based on the associated transform maps.

Before creating the Inbound Services, make sure you have listed all class models you want to include in the integration. As a general information all class models in ServiceNow are extending the Configuration Item table [cmdb_ci], except the relationship table. You will need an Inbound Service for different CI types and also an additional one for the relationship table.

For more information about Web Services Import Sets, see ServiceNow documentation [Web Service Import Sets](#) .

To create an Inbound Service:

1. Log in to ServiceNow as an administrator.
2. Under **System Web Services > Inbound**, click **Create New**.
 - a. Fill in the label field. For example, **Windows Server Inbound Service**.
 - b. Fill in the name field. Note that the target table's name looks like **cmdb_<x>**, the inbound should be named as follows: **ucmdb_integ_<x>**. For example, **ucmdb_integ_ci_win_server**.

Note: The OOTB connector handles Inbound Services starting with **ucmdb_integ_**. If you have a different naming convention, configure the UCMDB side accordingly.

3. Enable the **Copy fields from target table** check box.
4. Keep the **Create transform map** enabled if you want to display the create transform map form after saving the inbound service.
5. Select the target table you want to map it to. In this example, **cmdb_ci_win_server**.
6. Click **Create**. The displayed form is the form to create the transform mapping between the staging table and the target table.

To configure the transform map, see the section that follows.

Create Transform Maps between Staging Tables and Target Tables

Transform maps assure the mapping between staging tables and target tables. They provide the chance to process and filter the data before committing it in the target table. The transform maps will also be used to prevent duplicate from being created in ServiceNow by choosing a coalesce field.

All tables extending **cmdb_ci** have a **correlation_id** field. This field is used to contain the **global_id** sent from UCMDB. The value is unique and will serve as coalesce field.

The ServiceNow enhanced generic adapter offers the capability to not use the Correlation ID as coalesce field in case this field is used by a different integration and cannot be overwritten by UCMDB. In order to spare the Correlation ID field, see ["Create a Custom coalesce Field instead of Using the Correlation ID Field" on page 270](#).

The transform map for the relationship table is the only one different than every other class model. The coalesce field is the combination of three attributes: **parent**, **child**, and **type**.

To create the Transform map for all class models extending cmdb_ci

1. Click **Transform Maps** under **System Import Sets > Administration**, and then click **New**.
2. Fill in the name field. For example, **UCMDB - SNOW Integration TM Windows server**.
3. Make sure the source table points to the inbound service created.
4. Make sure the target table field points to the table targeted by the inbound service. (If you keep the **Create transform map** check box enabled, the source table and target table are automatically populated)
5. Click the **Auto Map Matching Fields** link.
6. In the **Fields Maps** tab, locate the field **correlation_id** and set coalesce to **true**.

To create the Transform map for the Relationship table

1. Click **Transform Maps** under **System Import Sets > Administration**.
2. Fill in the name field. For example, **UCMDB - SNOW Integration TM CI Relationship**.
3. Make sure the source table points to the inbound service created.
4. Make sure the target table field points **cmdb_rel_ci**. (If you keep the **Create transform map** check box enabled, the source table and target table are automatically populated.)
5. Click the **Auto Map Matching Fields** link.
6. In the **Fields Maps** tab, locate the **parent**, **child**, and **type** fields, and then set coalesce to **true** for the three attributes.
7. Click **Update** to save and exit. You might be invited to Index the Coalesce Fields.

Create a Role with Minimal Rights for the Integration

To create a user role for the integration:

1. Go to **User Administration > Roles**.
2. Specify a name and a description.
3. Under the **Contains Roles** tab, add the following:
 - a. Soap update
 - b. Soap Create

- c. Soap
 - d. Rest Service
4. Click **Update**.

The next step is to create ACLs to tighten up security. For this adapter you will need **Create**, **Read**, and **Write** ACLs for the staging tables and read only ACLs on the following:

- sys_db_object
- sys_dictionary
- sys_import_set_row
- sys_transform_entry
- sys_glide_object

The **Read** ACLs on the **fourfive sys** tables are necessary to be able to load class models in the UCMDB Mapping tool.

To create ACLs, you need elevated admin privileges.

Do the following:

1. Elevate the admin privileges by clicking the lock icon and check **security_admin**.
2. Go to **System Security > Access Control (ACL)**.
3. Click **New**.
 - a. Select record as type.
 - b. In operation select **create**.
 - c. Keep active and admin overrides check box checked.
 - d. In name select the inbound service created.
 - e. Add a description to remind you why you created this ACL.
 - f. In the **Requires role** section double click on **Insert a new row** and select the ucmdb integration role.
 - g. Click **Save** and exit.

Include the roles **Import Set loader** and **import set transformer** in the integration role.

Push Integration with ServiceNow

This section includes:

How to Push Data from UCMDB to ServiceNow	247
Push Integration Mechanism	251
How to Enable Web Services Security (WS-Security, WSS) for the Push Integration	252
CI Reconciliation	255
Sample Integration Push Query	256
Supported CITs	257
Pushing Additional CITs	257

How to Push Data from UCMDB to ServiceNow

To push data from UCMDB to ServiceNow, do the following:

1. Configure queries

The CIs and relationships to be pushed to ServiceNow have to be queried from UCMDB using TQL queries. Create integration type queries to query the CIs and relationships that have to be pushed to ServiceNow.

For the details of Push Query, you can refer to **UCMDB Help > Developer Reference > Creating Discovery and Integration Adapters > Developing Generic Adapters > Achieving Data Push using the Generic Adapter**. The Generic Adapter works in instance mode (meaning it does not work with single CI types, but with collections of CIs grouped together by a main root CI).

The adapter comes with the set of OOTB TQL queries that can be found under **Modeling Studio > Queries > Integrations > ServiceNow > Push** folder.

ServiceNow enhanced generic adapter OOTB push TQL queries

Push TQL name	Description
SNOW AIX Push 2.0	Push of AIX servers and related file systems, network adapters and IP addresses
SNOW AIX Relations Push	Push of relationships between AIX servers and file systems,

ServiceNow enhanced generic adapter OOTB push TQL queries, continued

Push TQL name	Description
1.0	network adapters and IP addresses
SNOW Apache Push 1.0	Push of Apache Web Servers
SNOW Business Element Push 1.0	Push of Business Applications
SNOW Business Service Push 1.0	Push of Business Services
SNOW Business Service Relations Push 1.0	Push of relationships between Business Services and Business Applications
SNOW Cluster Push 1.0	Push of clusters and related servers
SNOW Computer Push 1.0	Push of Computers (excludes Windows, ESX, Linux, HP-UX and AIX servers)
SNOW DB2 Push 1.0	Push of DB2 database servers
SNOW ESX Push 1.0	Push of ESX servers and related file systems, network adapters and IP addresses
SNOW ESX Relations Push 1.0	Push of relationships between ESX servers and file systems, network adapters and IP addresses
SNOW HP-UX Push 2.0	Push of HP-UX servers and related file systems, network adapters and IP addresses
SNOW HP-UX Relations Push 1.0	Push of relationships between HP-UX servers and file systems, network adapters and IP addresses
SNOW LoadBalancer Push 1.0	Push of Load Balancers
SNOW Linux Push 2.0	Push of Linux servers and related file systems, network adapters and IP addresses
SNOW Linux Relations Push 1.0	Push of relationships between Linux servers and file systems, network adapters and IP addresses
SNOW Local Printer Push 1.0	Push of Local Printers
SNOW IIS Web Server Push 1.0	Push of IIS Web Servers
SNOW MSSQLDB Push	Push of MS SQL database Servers

ServiceNow enhanced generic adapter OOTB push TQL queries, continued

Push TQL name	Description
1.0	
SNOW MySQL Push 1.0	Push of MySQL database Servers
SNOW Net Printer Push 1.0	Push of Network Printers
SNOW Node Relations Push 1.0	Push of relationships between Nodes and file systems, network adapters and IP addresses
SNOW OracleDB Push 1.0	Push of Oracle database Servers
SNOW PostgreSQL Push 1.0	Push of PostgreSQL database Servers
SNOW Router Push 1.0	Push of Routers
SNOW Switch Push 1.0	Push of Switches
SNOW Unix Push 2.0	Push of UNIX servers and related file systems, network adapters and IP addresses
SNOW Windows Cluster Push 1.0	Push of Windows Clusters
SNOW Windows Push 2.0	Push of Windows servers and related file systems, network adapters and IP addresses
SNOW Windows Relations Push 1.0	Push of relationships between Windows servers and file systems, network adapters and IP addresses

For details on viewing queries in the Modeling Studio, see the *Modeling section of the UCMDB Help*.

2. Create XML mapping files

For every query created in the step above, create an XML mapping file with exactly the same name (case-sensitive) as the integration query with the following prefix:

ServiceNowGenericAdapter/mappings/push/<mapping file name>.xml


A set of mapping files for the TQL queries is provided out of the box with the package.

For more information about mapping files, see the *Developer Reference section of the UCMDB Help*.

3. Create the integration point

Define the integration point as follows:


- a. In UCMDB, go to **Data Flow Management > Integration Studio**.
- b. In the New Integration Point dialog box, enter a name and description for the integration point.
Ensure the **Is Integration Activated** option is enabled.

- c. In the Adapter field click the **Select Adapter**  button.
- d. In the Select Adapter list, select **ServiceNowEnhancedAdapter** and click **OK**.
- e. In the Adapter Properties section, enter the following required properties:

Field	Value
Protocol	HTTP or HTTPS. Default: https
Service-Now Domain	The ServiceNow domain. Default: service-now.com
Service-Now Instance	The ServiceNow instance being used.
Port	For HTTPS the default is: 443
Credentials ID	Define the ServiceNow instance user name and password in the Generic Protocol. For credential information, see Supported Protocols in the <i>UCMDB Discovery and Integrations Content Guide</i> .
Proxy Host	The hostname, or address, of the proxy server.
Proxy Port	The port number of the proxy server.
Data Flow Probe	Select the name of the Data Flow Probe to run this integration.

- f. Test the connection to the target ServiceNow server. If the connection fails, verify that the information provided is correct.
- g. Click **OK** to save the integration point.
- h. Add a new job definition to the integration point. Provide a name for the job definition and select the queries to use to synchronize data from UCMDB to ServiceNow. Define a synchronization schedule if required.

Note: Jobs can also be run without a schedule.

- i. Save the job definition, and then the integration point.
- j. Click  to run a full synchronization for each job at least once.

Push Integration Mechanism

The components responsible for the ServiceNow integration are bundled in the ServiceNow Integration package, **SNOWGenericAdapter.zip**.

The adapter receives the CI and relationship data sent from UCMDB server and transfers this data to ServiceNow using either SOAP or JSONv2 connector. For more details about the connectors, see ["Connectors" on page 264](#).

The ServiceNow enhanced generic adapter can push the CIs and relations both directly or via import sets. However, since the ServiceNow coalescing ("[CI Reconciliation](#)") mechanism is not available for the direct table approach import sets are the preferred way of pushing data to ServiceNow. The information about the ServiceNow tables to insert the data into as well as the record fields to insert into the tables is specified in mapping files.

Consider the following example from the mapping file:

```
<target_entities>
  <source_instance query-name="SNOW Windows Push 1.0" root-element-
name="Root">
    <target_entity name="ucmdb_integ_ci_win_server">
      <target_mapping datatype="STRING" name="correlation_id" value="Root
['global_id']"/>
      <target_mapping datatype="STRING" name="name" value="Root['display_
label']"/>
      ...
      <for-each-source-entity count-index="i" source-
entities="Root.FileSystem">
        <target_entity name="ucmdb_integ_ci_file_system">
          <target_mapping datatype="STRING" name="correlation_id"
value="Root.FileSystem[i]['global_id']"/>
          <target_mapping datatype="STRING" name="name"
value="Root.FileSystem[i]['display_label']"/>
          <target_mapping datatype="STRING" name="computer"
value="Root['global_id']"/>
        </target_entity>
      </for-each-source-entity>
    </target_entity>
```

```
...  
</source_instance>  
</target_entities>
```

In this example the adapter will insert a record with the fields **name** and **correlation_id** into the staging table **ucmdb_integ_ci_win_server**.

Note that when pushing data to **ServiceNow** via import sets, there are some mandatory fields that need to be specified on both CIs and relations:

1. Each CI must have the **correlation_id** (or custom ServiceNow field) property set. The reconciliation is done on this field on ServiceNow side. If it does not have this property set, then it is ignored by the push job. See ["CI Reconciliation" on page 255](#). Furthermore, this field needs to be set to UCMDB CI **global_id**.
2. Each child CI must have the corresponding ServiceNow child-parent relation field set to the parent UCMDB CI **global_id**. When we do that the adapter finds this field in the push job and replaces its value with ServiceNow parent CI **sys_id** before it sends it to ServiceNow. In the example above the child entity **ucmdb_integ_ci_file_system** has the **computer** field set to the **global_id** value of the parent entity **ucmdb_integ_ci_win_server**.
3. Each relation must have **parent** and **child** properties set. If it does not have these properties set, then it is ignored by the push job. See ["CI Reconciliation" on page 255](#).
4. While the mapping is done from UCMDB CI to ServiceNow staging table, the deleted UCMDB CIs need to be deleted from ServiceNow target table. The adapter tries to find the staging table to target table mapping in ["ServiceNowConfig.xml Configuration"](#), so the additional mapping needs to be defined there.

The integration mechanism is explained in *Developer Reference section of the UCMDB Help*.

How to Enable Web Services Security (WS-Security, WSS) for the Push Integration

To enable WSS for the UCMDB-ServiceNow push integration, do the following:

1. Create a Java KeyStore (JKS) with your certificate
 - a. Make sure that Java is installed on your machine and the Java **bin** directory is in your PATH.

- b. Use the keytool embedded in Java to create a JKS as follows.

```
keytool -genkey -keyalg RSA -alias <your keystore alias> -keystore <the keystore file that you want to create> -storepass <your keystore password> -validity <the number of days that your certificate should be valid> -keysize 2048
```

- c. The system will ask you some questions, you just fill in with arbitrary values.
- d. A JKS file is created. It is a Java Keystore that contains your new certificate and the public and private keys.

2. Extract the certificate to a PEM file

- a. Use the keytool to extract the certificate from the Keystore and save it in PEM format as follows.

```
keytool -exportcert -alias <the alias for your certificate in Keystore> -keypass <your key password> -keystore <your created keystore file name> -storepass <your keystore password> -rfc -file <the file name that you want to create in PEM format>
```

- b. A file with the **.pem** extension is created. This file contains your certificate in PEM format.
- c. Navigate to the directory where the **.pem** certificate is exported and run the following command:

```
type <your created PEM file name>
```

3. Configure WSS on the ServiceNow side

- a. Go to [Inbound SOAP Web Service Security](#).
- b. Follow **3.1 Enabling WS-Security Verification**.

Note: Take into account step 3.1.1 if you are using MID Server, ODBC Driver, Remote Update Sets, and high availability cloning SOAP interfaces because they cannot use SOAP WSS authentication due to technical implications. You need to allow them to bypass the WSS authentication requirements by marking their user accounts as internal integration users.

- c. Follow **3.2 WS Security Profiles**.

- i. Create a WSS profile for x.509 Token Profile.

In the WS Security Profile window, make sure that the following fields are configured as follows:

- In the **Type** field, select **X509**.
- In the **X509 Certificate** field, select the PEM file that you created in [step 2](#).

- ii. Create a WSS profile for Username Token Profile.

In the WS Security Profile window, make sure that the following fields are configured as follows:

- In the **Type** field, select **Username**.
- In the **Profile action** field, select **Specify user to authenticate**.

- d. (Optional) You may want to restrict all SOAP requests to use WSS when these requests come into your instance. To do so, go to **System Web Services > Properties** module, and select the check box under **Require WS-Security header verification for all incoming SOAP requests**.

4. Configure the integration point on the UCMDDB side

In the Adapter Properties section, enter the following required properties.

Field	Value
WSSKeystoreAlias	The WSS Keystore alias that you created in step 1 .
WSSKeystoreFile	The path (including the file name) of the WSS Keystore file that you created in step 1 .
WSSKeystoreKeypass	The key password of WSS Keystore that you created in step 1 .
WSSKeystoreStorepass	The WSS Keystore password that you created in step 1 .
WSSKeystoreType	jks
WSSPassword	The password of WSS Username Token Profile that you created in step 3 .
WSSUsername	The user name of WSS Username Token Profile that you created in step 3 .

For more details, see [Create the integration point](#).

5. Activate the dedicated WSS log on the UCMDDB side

Add the following appender in the **<DataFlowProbe_Home>\conf\log\fcmdb.properties** file.

```
#####FCMDB#####  
### fcmdb.appender          ##  
#####  
  
log4j.appender.fcmdb.appender=com.mercury.topaz.cmdb.shared.base.log.BetterRollingFileAppender  
log4j.appender.fcmdb.appender.File=${logs.dir}/fcmdb.log  
  
log4j.appender.fcmdb.appender.MaxFileSize=${def.file.max.size}log4j.appender.fcmdb.appender.MaxBackupIndex=${def.files.backup.count}log4j.appender.fcmdb.appender.layout=org.apache.log4j.PatternLayoutlog4j.appender.fcmdb.appender.layout.ConversionPattern=${msg.layout}
```

6. Activate logging on the ServiceNow side

Refer to [3.4 Enabling WS-Security Logging](#).

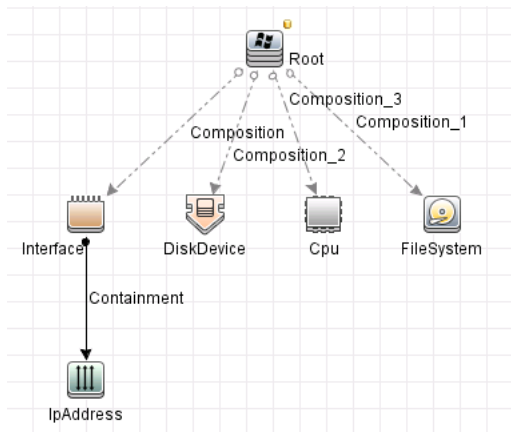
CI Reconciliation

CI reconciliation is done on ServiceNow side using the following approach:

1. For CIs the reconciliation is done on **correlation_id** field (or a custom ServiceNow field) .
UCMDB CI **global_id** property must be mapped to this field in the mapping file.
2. For relations the reconciliation is done on **parent** and **child** fields. In ServiceNow these fields need to have the **sys_id** values of parent and child records respectively. However, since UCMDB CIs don't have **sys_id** values but have **global_id** values the additional step is performed by adapter in order to successfully map UCMDB relation to ServiceNow relation:
 - a. For push job in mapping files UCMDB **global_id** values of the parent and child CIs need to be mapped to relation **parent** and **child** fields. The adapter deduces the **sys_id** values based on these **global_id** (mapped to **correlation_id** or to a custom ServiceNow field) values prior to pushing the relations to ServiceNow.
 - b. For population job in mapping files the relation **parent** and **child** fields contain the **global_id** (mapped to **correlation_id** or to a custom ServiceNow field) values of the parent and child CIs in UCMDB. The adapter takes care of this **sys_id-correlation_id** translation internally.

Sample Integration Push Query

The following image displays a sample query “SNOW Windows Push 2.0” to be pushed to ServiceNow:



Windows Server CIT is marked as Root, so it is a starting point.

The following mapping file is created **ServiceNowGenericAdapter/mappings/push/SNOW Windows Push 2.0.xml** and this is how it looks like in the Graphic Mapping Tool:

The screenshot displays the Mapping Tool interface for a 'Push Scenario'. On the left, a tree view shows the mapping file structure. The central area contains a list of mapping rules, each with a name and a description. The right pane shows the resulting TQL query, which is a complex nested structure representing the hierarchy of the mapping file.

Note how the structure of the mapping file follows the hierarchy in the TQL query.

There are some details that needs to be explained regarding the “For” loop:



When pushing data to ServiceNow, we need to use Inbound Services, in this case `ucmdb_integ_ci_network_adapter` for pushing CIs of Interface CI Type. In ServiceNow `ucmdb_integ_ci_network_adapter` there is a reference pointing to ServiceNow Server table and because of that, it needs to have value of Windows Server `global_id`. That way we keep that reference across solution. Observe similar thing for IP address (it points to the Interface).

Supported CITs

All CI Types and Relationships are supported as long as you find or define one on ServiceNow side.

For details about OOTB CI Types, see [ServiceNow enhanced generic adapter OOTB push TQL queries](#).

Pushing Additional CITs

To push additional CITs, these CITs should be added to the mapping file. For details, see the *Developer Reference section of the UCMDB Help*.

Pushing additional CI Types requires creation of Inbound services and transform maps for specific direct table as explained in "[Setting up ServiceNow for the UCMDB Integration](#)" on page 243.

After that, create the desired TQL query and the corresponding mapping file.

Population from ServiceNow

This section includes:

How to Populate UCMDB with Data from ServiceNow	258
Population Flow	260
Sample Integration Population Query	261
Supported CITs	263

How to Populate UCMDB with Data from ServiceNow

To populate UCMDB with data from ServiceNow, do the following:

1. Configure queries

The population TQL query is created in UCMDB to indicate the data that needs to be populated from ServiceNow.

For the details of Population Queries, you can refer to **Developer Reference > Creating Discovery and Integration Adapters > Developing Generic Adapters > Achieving Data Population using the Generic Adapter**. The Generic Adapter works in instance mode (meaning it does not work with single CI types, but with collections of CIs grouped together by a main root CI).

The adapter comes with the following set of OOTB TQL queries that can be found under the **Modeling Studio > Queries > Integrations > ServiceNow > Population** folder.

ServiceNow enhanced generic adapter OOTB population TQL queries

Population TQL name	Description
SNOW AIX Population 1.0	Population of AIX servers
SNOW Apache Population 1.0	Population of Apache Web Servers
SNOW Business Application Population 1.0	Population of Business Applications
SNOW Biz Containment Biz	Population of Containment relationship between Business

ServiceNow enhanced generic adapter OOTB population TQL queries, continued

Population TQL name	Description
1.0	Service and Business Application
SNOW Biz Containment Computer 1.0	Population of Containment relationship between Business Service and Computer
SNOW Biz Containment Database 1.0	Population of Containment relationship between Business Service and Database
SNOW Biz Containment WebServer 1.0	Population of Containment relationship between Business Service and Web Server
SNOW Business Service Population 1.0	Population of Business Service
SNOW Computer Composition Database 1.0	Population of Composition relationship between Computer and Database
SNOW Computer Composition WebServer 1.0	Population of Composition relationship between Computer and Web Server
SNOW ESX Population 1.0	Population of ESX servers
SNOW HPUX Population 1.0	Population of HP-UX servers
SNOW Linux Population 1.0	Population of Linux servers
SNOW Windows Population 1.0	Population of Windows servers

For details on viewing queries in the Modeling Studio, see the *Modeling section of the UCMDB Help*.

2. Create XML Mapping Files.

The XML mapping file describes the topology that should be pulled from the ServiceNow instance and how it should be mapped to the UCMDB class model. Mapping files for population are created in **ServiceNowGenericAdapter/mappings/population/<name of the mapping file>.xml**.

3. Define a synchronization schedule for the integration job, if desired.

Jobs can also be run manually, without a schedule.

4. Run full data synchronization.

Click  to synchronize all data.

Population Flow

Looking from the adapter perspective, the population flow consists of the following steps:

1. The adapter receives the information about the CI and relation data that needs to be obtained from ServiceNow.
2. Then it connects to ServiceNow and transfers the required data using **ServiceNow REST API**. The CIs and relations are fetched directly from target tables (no staging tables here) and handed over to the next stage in the pipeline.
3. After the CIs and relations are transformed and inserted into UCMDB then the UCMDB sends the information about the assigned **global_id** values for each CI to the adapter
4. The adapter updates the **correlation_id** (or custom) fields of the populated records with the assigned **global_id** values. This step is optional and can be disabled in the adapter configuration file.

The information about the ServiceNow tables to populate the data from is specified in mapping files.

Consider the following example from the mapping file:

```
<target_entities>
  <source_instance query-name="TEST UNIX Population 1.0" root-element-
name="cmdb_ci_unix_server">
    <target_entity name="Root">
      <target_mapping datatype="STRING" name="name" value="cmdb_ci_unix_server
['name']"/>
      <target_mapping datatype="STRING" name="sys_id" value="cmdb_ci_unix_
server['sys_id']"/>
      <target_mapping datatype="STRING" name="global_id" value="cmdb_ci_unix_
server['correlation_id']"/>
      ...
    </source_instance>
  </target_entities>
```

In this example the adapter will query the table **cmdb_ci_unix_server** for target entity named **Root**.

Additionally, there are some fields that deserve special attention in the mapping XML. This can be seen in the above example:

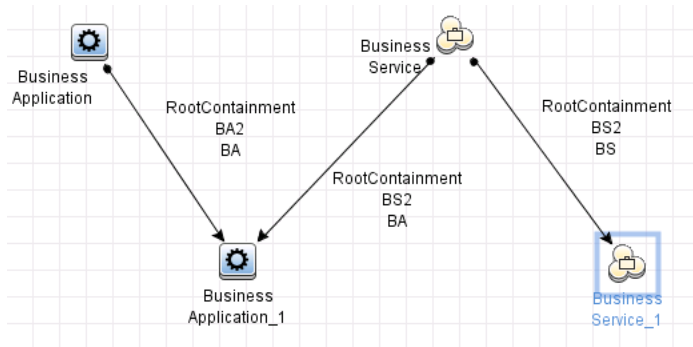
1. **correlation_id** (or a custom ServiceNow field) needs to be mapped to UCMDB **global_id** property. This greatly helps in UCMDB reconciliation process. Please note that if this mapping is not

specified UCMDB will still do the reconciliation on the CI properties themselves but this is not recommended. Using **correlation_id-global_id** mapping exactly correlates the CIs from the two servers. For more details, see ["CI Reconciliation" on page 255](#).

2. ServiceNow **sys_id** field must be mapped to UCMDB CI **sys_id** property. (This is a virtual UCMDB property). This mechanism is essential for adapter in order to update the ServiceNow **correlation_id** (or a custom ServiceNow field) with the assigned **global_id** value. If this mapping is not specified, then the adapter can't deduce the source CI **sys_id** value and cannot perform the update.

Sample Integration Population Query

The following image displays a sample query "SNOW Biz Containment Biz 1.0" to be populated from ServiceNow:



Note that relationships are marked as **Root**, so in this case every **Root** element contains a pair of CIs, parent and child.

The following mapping file is created **ServiceNowGenericAdapter/mappings/population/SNOW Biz Containment Biz 1.0.xml**, and this is how it looks like in XML format:

```
<target_entities>
  <source_instance query-name="SNOW Biz Containment Biz 1.0" root-element-
name="cldb_rel_ci">
    <target_entity is-valid="cldb_rel_ci['parent_table'] == 'cldb_ci_
service'" name="BusinessService">
      <target_mapping datatype="STRING" name="global_id" value="cldb_rel_
ci['parent']"/>
    </target_entity>
    <target_entity is-valid="cldb_rel_ci['parent_table'] == 'cldb_ci_appl'"
name="BusinessApplication">
      <target_mapping datatype="STRING" name="global_id" value="cldb_rel_
```

```
ci['parent']"/>
    </target_entity>
    <target_entity is-valid="cmdb_rel_ci['child_table'] == 'cmdb_ci_
service'" name="BusinessService_1">
        <target_mapping datatype="STRING" name="global_id" value="cmdb_rel_
ci['child']"/>
    </target_entity>
    <target_entity is-valid="cmdb_rel_ci['child_table'] == 'cmdb_ci_appl'"
name="BusinessApplication_1">
        <target_mapping datatype="STRING" name="global_id" value="cmdb_rel_
ci['child']"/>
    </target_entity>
</source_instance>
...
```

Note how the structure of the mapping file follows the hierarchy in the TQL query, relationship as a Root, CIs on both ends.

You can limit the processing the result from the ServiceNow table **cmdb_ci_rel** by defining **is-valid** condition. In this example **is-valid="cmdb_rel_ci['parent_table'] == 'cmdb_ci_service'"**, it checks if the value is coming from the **cmdb_ci_service** table.

Sometimes CIs in ServiceNow do not have a relationship defined between them, but one CI keeps the **sys_id** of another CI in one of its attributes. For example, **cmdb_ci_network_adapter** has the attribute **cmdb_ci**, which contains the **sys_id** of the host where this adapter resides. To handle this situation in the mapping file, you need to have a valid **reference** attribute on the ServiceNow side and use dot-walking in the mapping. For example, to populate Windows CI with related interfaces, create a mapping file like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../mappings_schema.xsd">
    <info>
        <source name="SNOW" vendor="HP" version="9.40"/>
        <target name="SNOW" vendor="ServiceNow" version="7.0"/>
    </info>
    <import>
        <scriptFile path="mappings.scripts.SNOWUtils"/>
    </import>
    <!--
    Populate SNOW ESX Servers to uCMDB.
    -->
    <target_entities>
        <source_instance query-name="Windows with interfaces Population" root-
element-name="cmdb_ci_win_server">
            <target_entity name="Root">
```

```
<target_mapping datatype="STRING" name="global_id" value="cldb_ci_win_
server['correlation_id']"/>
<target_mapping datatype="STRING" name="display_label" value="cldb_ci_
win_server['name']"/>
<target_mapping datatype="STRING" name="name" value="cldb_ci_win_server
['name']"/>
<target_mapping datatype="STRING" name="sys_id" value="cldb_ci_win_
server['sys_id']"/>
</target_entity>
<for-each-source-entity count-index="i" source-entities="cldb_ci_win_
server.cldb_ci_network_adapter">
  <target_entity name="Interface">
    <target_mapping datatype="STRING" name="name" value="cldb_ci_win_
server.cldb_ci_network_adapter[i]['name']"/>
    <target_mapping datatype="STRING" name="sys_id" value="cldb_ci_win_
server.cldb_ci_network_adapter[i]['sys_id']"/>
    <target_mapping datatype="STRING" name="interface_name" value="cldb_
ci_win_server.cldb_ci_network_adapter[i]['name']"/>
    <target_mapping datatype="STRING" name="mac_address" value="cldb_ci_
win_server.cldb_ci_network_adapter[i]['mac_address']"/>
    <target_mapping datatype="STRING" name="global_id" value="cldb_ci_win_
server.cldb_ci_network_adapter[i]['correlation_id']"/>
    <target_mapping datatype="STRING" name="root_container" value="cldb_
ci_win_server['correlation_id']"/>
  </target_entity>
</for-each-source-entity>
</source_instance>
</target_entities>
</integration>
```

Supported CITs

The population integration supports any CI and relationship that is available in ServiceNow.

For the list of OOTB CI Types, see [ServiceNow enhanced generic adapter OOTB population TQL queries](#).

External Class Model Flow

Whenever a user opens a graphical mapping tool or clicks the **Refresh External Class Model** button in UCMDB, a request is made against the adapter for the ServiceNow external class model. The adapter queries the following tables:

1. **sys_db_object** – this table contains information about all the tables that compose ServiceNow schema
2. **sys_dictionary** – this table contains all the table fields that compose ServiceNow schema
3. **sys_transform_entry** – this table contains the information about all the staging table to target table mappings in ServiceNow.

Users can specify the list of prefixes of the ServiceNow tables that compose external class model in the adapter configuration. Using this information, the adapter builds the external class model and sends it to UCMDB.

Note that if a table that goes into the class model has a reference to a table that doesn't go into the class model then this reference field is excluded from the external class model.

Incremental Update

Since the external class model is pretty big the adapter offers a configuration property that enables **incremental class model update**. Incremental update is a nice feature that greatly improves performance. Using this feature, after the adapter is started or restarted the complete external class model will be fetched by the adapter only at first request. All the subsequent requests for the class model will only fetch the modifications in the class model.

However, incremental update can't detect deletions in the class model. That is to say, if a user deletes a table or a field in a table in ServiceNow side the incremental update won't detect this. The incremental update only detects new items and modifications in the class model. Even with this limitation this feature is pretty handy and it is recommended to use it.

Users can enable/disable incremental update in the adapter configuration file. The configuration file is explained below.

Connectors

The ServiceNow enhanced generic adapter communicates with ServiceNow using HTTPS protocol and uses two types of connectors: SOAP and REST/JSONv2. Users can choose whichever connector they prefer in the adapter configuration. The configuration file is explained below.

REST/JSONv2 connector uses [JSONv2](#) for push operations and [REST API](#) for population and class model operations. It operates on JSON objects as the data input and output format.

SOAP connector uses [SOAP Web Service](#) for communication to ServiceNow. It uses Java SAAJ under the hood and does not use the WS stubs. This increases user experience since users are not required to compile additional jar files and apply patches to the adapter code. All they need to do is apply the correct mapping files and WS request/response messages are created according to the mapping files. However, for the current version of the adapter only the push operations are implemented.

Performance

Initially the plan was to implement only the REST connector for the adapter. However, the performance tests have shown the performance difference between the two implementations. While REST is faster for read operations SOAP performs faster for write operations. This was the reason for SOAP connector implementation. As said above, only the push operations are implemented for SOAP connector.

Configuration

The ServiceNow enhanced generic adapter configuration is split into three parts: integration point configuration, **adapter.properties** configuration, and **ServiceNowConfig.xml**. For details about integration points configuration, see [Create the integration point](#).

adapter.properties Configuration

This configuration contains various enable/disable features specific to the adapter itself. The configuration resides in the following file:

<adapter package>\discoveryConfigFiles\ServiceNowGenericAdapter\adapter.properties

The following properties are supported at the moment:

- **connector.class**

This property specifies the class name of the underlying connector used for communicating to ServiceNow. At the moment only REST connector implementation is supported.

```
connector.class=com.hpe.ucmdb.adapters.snow.connector.ServiceNowRestConnector
```

- **push.connector.class**

This property specifies the class name of the underlying connector used for push job only. The supported values are **com.hpe.ucmdb.adapters.snow.connector.ServiceNowRestConnector** and **com.hpe.ucmdb.adapters.snow.connector.ServiceNowSoapConnector**. In case this property is not specified then the value of the **connector.class** property is used.

- **SOAP connector internal properties**

These are the standard JAVA SAAJ properties that specify the SOAP SAAJ implementation used for the SOAP connector. Users can use the one that is available on their system. The default implementation used in development is specified by these properties:

```
javax.xml.soap.SOAPConnectionFactory=com.sun.xml.internal.messaging.saaj.client.p2p.HttpSOAPConnectionFactory  
  
javax.xml.soap.MessageFactory=com.sun.xml.internal.messaging.saaj.soap.ver1_1.SOAPMessageFactory1_1Impl  
  
javax.xml.soap.MetaFactory=com.sun.xml.internal.messaging.saaj.soap.SAAJMetaFactoryImpl
```

- **Retry Policy Properties**

These properties control the call retries in case there is a connection outage or the server is temporarily unavailable. **connector.retry.count** specifies the maximum number of retries. The default value is 3 in case this property is not specified. **connector.retry.interval** specifies the interval in seconds between two calls. The default value is 60 seconds.

```
connector.retry.count=3  
connector.retry.interval=60
```

The property **connector.retry.statuses** lists HTTP status codes that act like the triggers for the retry mechanism to be started. By default, it lists as follows.

```
connector.retry.statuses=408,503,-1
```

- **408** – Request Timeout – The 408 Request Timeout error is an HTTP status code that means the request that you send to the website server (for example, a request to load a web page) takes longer than the website's server is prepared to wait.
- **503** – Service Unavailable – This response code is received whenever the web server currently cannot handle the HTTP request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay.

Some servers in this state may also simply refuse the socket connection, in which case a different error may be generated because the socket creation timed out.

- **-1** – This response code is received whenever the instance is unreachable.

Custom error codes can be added to the **adapter.properties** file if you want to do so.

- **population.return.global_id**

Set this feature to **false** in case you do not want the adapter to update the ServiceNow **correlation_id** fields with generated UCMDB **global_id** values for populated CIs. See "[Population Flow](#)" for more information. The default value is **true**.

- **class.model.coalesce.field**

This property contains the field used for the reconciliation on ServiceNow side (see "[CI Reconciliation](#)"). In the push and population mapping files this field must be set to UCMDB **global_id** value. If not specified, the default value is **correlation_id**.

- **class.model.table.prefixes**

This property contains a comma separated list of prefixes of all the ServiceNow tables that should compose external class model. The default and strongly recommended prefixes are **cmdb_ci**, **cmdb_rel_ci**, **cmdb_rel_type**, **sys_import_set_row** and **ucmdb_integ_**. But users can change this list in whatever way they can. For more information, see "[External Class Model Flow](#)" on [page 263](#).

- **class.model.incremental.update**

Set this value to **true** to perform the incremental class model update, or set it to **false** to disable the incremental class model update. See [Incremental Update](#) for more details.

- **adapter.logger**

Specifies the adapter specific logger name. Enable the adapter log to isolate the logs coming from this adapter from all other logs on the probe. In order to achieve this, you need to add additional entries in the **<DataFlowProbe_Install_Dir>\conf\log\fcmdb.properties** file.

In the given example if we need to enable the TRACE logging and we don't want to affect other adapter loggers we can specify this property in **adapter.properties** file:

```
adapter.logger=servicenow.generic.adapter
```

With this property set, we add the following entries in the **<DataFlowProbe_Install_Dir>\conf\log\fcmdb.properties** file:

```
log4j.category.servicenow.generic.adapter=TRACE,servicenow.generic.adapter.ap  
pender  
  
log4j.appender.servicenow.generic.adapter.appender=com.mercury.topaz.cmdb.sha  
red.base.log.BetterRollingFileAppender  
  
log4j.appender.servicenow.generic.adapter.appender.File=${logs.dir}/serviceno  
w.generic.adapter.log  
  
log4j.appender.servicenow.generic.adapter.appender.MaxFileSize=10240KB  
  
log4j.appender.servicenow.generic.adapter.appender.MaxBackupIndex=${def.files  
.backup.count}  
  
log4j.appender.servicenow.generic.adapter.appender.layout=org.apache.log4j.Pa  
tternLayout  
  
log4j.appender.servicenow.generic.adapter.appender.layout.ConversionPattern=$  
{msg.layout}
```

ServiceNowConfig.xml Configuration

This configuration contains the business logic information specific to certain jobs. It resides in the following file:

<adapter package>\discoveryConfigFiles\ServiceNowGenericAdapter\ServiceNowConfig.xml

Here is an example of this file:

```
<config>  
  <ciTypeMappings>  
    <ci ciType="disk_device" stagingTable="ucmdb_integ_ci_storage_device"  
targetTable="cmdb_ci_storage_device">  
      </ci>  
    <ci ciType="file_system" stagingTable="ucmdb_integ_ci_file_system"  
targetTable="cmdb_ci_file_system">  
      </ci>  
    <ci ciType="interface" stagingTable="ucmdb_integ_ci_network_adapter"  
targetTable="cmdb_ci_network_adapter">  
      <relations>  
        <relation childCiType="ip_address" type="containment"  
childCiTargetTableField="nic" />  
      </relations>  
    </ci>  
    <ci ciType="ip_address" stagingTable="ucmdb_integ_ci_ip_address"  
targetTable="cmdb_ci_ip_address">  
      </ci>
```

```
<ci ciType="nt" stagingTable="ucmdb_integ_ci_win_server"
targetTable="cmdb_ci_win_server">
  <relations>
    <relation childCiType="interface" type="composition"
childCiTargetTableField="cmdb_ci" />
    <relation childCiType="file_system" type="composition"
childCiTargetTableField="computer" />
    <relation childCiType="disk_device" type="composition"
childCiTargetTableField="computer" />
  </relations>
</ci>
<ci ciType="containment" stagingTable="ucmdb_integ_rel_ci"
targetTable="cmdb_rel_ci">
</ci>
<ci ciType="composition" stagingTable="ucmdb_integ_rel_ci"
targetTable="cmdb_rel_ci">
</ci>
</ciTypeMappings>
<populationQueries>
  <tql name="SNOW Biz Containment Biz 1.0"/>
  <tql name="SNOW Windows Population 1.0"/>
</populationQueries>
</config>
```

For the current adapter version, this configuration contains these types of data:

- **List of Population Queries**

When we create a new pair of population TQL and mapping file, in order to display this TQL in “Add Query” dialog of the “Edit Integration Job” action for the integration point we need to add the TQL name in this list of population queries.

- **CI/Relation Type Mappings**

These mappings help push and population jobs in determining the correct mapping of UCMDDB CI/relation type to ServiceNow staging table and target table.

Let’s consider the Windows CI push to ServiceNow. In the example above we can see that UCMDDB CI type **nt** has the corresponding ServiceNow staging table **ucmdb_integ_ci_win_server** and target table **cmdb_ci_win_server**. Also it has 3 CI types related to it: **interface**, **file_system** and **disk_device**. In the same way we can see that CI type **interface** has the corresponding ServiceNow staging table **ucmdb_integ_ci_network_adapter** and target table **cmdb_ci_network_adapter**.

In the push job, the CIs are mapped to staging tables. However, the deleted CIs from UCMDDB need to be deleted from target table. For every staging table the adapter tries to read the target table from

this configuration (see ["Push Integration Mechanism"](#)). If the mapping is not there, then deletion is not performed.

Furthermore, if the corresponding child-parent field mapping is not specified in the XML mapping file (see ["Push Integration Mechanism"](#)), then the adapter tries to find the child-parent relation field in this configuration. In the example above, in the defined relation from **nt** to **interface**, we can also see the attribute **childCiTargetTableField="cmdb_ci"**. This attribute points to the ServiceNow child table field that points to the parent table. In this case the ServiceNow relation is defined by the **cmdb_ci_network_adapter.cmdb_ci** field that points to parent table **cmdb_ci_win_server**. The adapter assigns the parent CI **sys_id** value generated by ServiceNow to the specified field of the child CI right before it pushes it to ServiceNow.

The opposite of push is the population job. For the populated CI when the adapter needs to push the generated UCMDB **global_id** value back to ServiceNow **correlation_id** field, from UCMDB it only receives the UCMDB CI type information about the CI along the **sys_id** and **global_id** (see ["CI Reconciliation"](#)). Using this mapping the adapter can deduce the ServiceNow table to update for the specified CI type. However, if the corresponding mapping in this file is not specified then the adapter assumes that the table to update is **cmdb_ci** which is correct since all CMDB tables in ServiceNow extend **cmdb_ci** table.

Create a Custom coalesce Field instead of Using the Correlation ID Field

Instead of using ServiceNow **correlation_id** field (mapped to UCMDB **global_id** field), the adapter allows using custom field that can be used in the reconciliation mechanism. Below are the steps needed in order to enable this feature.

ServiceNow Changes

The idea here is to create a custom field in the Configuration Item table [cmdb_ci] (Because all class models are inheriting from this table). And then make sure that the custom field exists also in the staging table, then change the mapping to map the Global ID to the created custom field.

Add the custom field to the target table

1. Log in to ServiceNow as admin.
2. Under **System Definition** locate and click **Tables**.

3. Filter on **Name** to find the **cmdb_ci** table.
4. Add a custom field as **String**, with the value of **100** as **Max length**.
5. Click **Update** to save and quit.

The Created field will be starting with a prefix to isolate it from OOTB ServiceNow fields, and avoid collisions during upgrades.

The next step consists of making sure that the staging tables have the created custom field. There are different scenarios at this point:

- If you are creating an import set as described in "[Create Inbound Services for Class Models](#)", the custom field will be automatically in the staging table while using the Copy fields from target table feature.
- If you are using the OOTB Update set that comes with the connector. You must update the Inbound Services by adding the custom field and change the mapping by removing the Correlation ID mapping and replace it by the custom field mapping.

Add the custom field to the staging table

1. Log in to ServiceNow as admin.
2. Under **System Web Services > Inbound**, locate and click the inbound service you need to update.
3. Scroll down the list of Web services fields, and double click **Insert a new row**.
4. Populate the label and make sure the name is identical to the custom field you created in the Configuration Item table.
5. Delete the Correlation ID field from the Inbound Service to avoid having it mapped using the auto map matching field during the creation of the Transform Map.

Change the mapping in order to use the new custom field

1. Log in to ServiceNow as admin.
2. Under **System Import Sets > Administration**, click **Transform Maps**.
3. Locate and click the transform map to update.
4. If you are using the OOTB Update set or you created a mapping for the Correlation ID. You must locate and remove the Correlation ID mapping row from the Field Maps tab.
5. To delete a row, check the box and scroll down to the Action on the selected row choice list, and then select **Delete**.

6. Check if the created custom field is mapped correctly to the target table after using the Auto map matching fields feature. If you did not use this feature after creating the custom field, then you have to add the row manually either by:

- clicking **Auto map matching fields**, or,

Note: Make sure that Correlation ID was not left in the Inbound Service or just remove it from the mapping afterward.

- use the map assist from the related links to add only the created custom field.

Related Links

[Auto Map Matching Fields](#)
[Mapping Assist](#)
[Transform](#)
[Index Coalesce Fields](#)

7. Set the Created custom attribute as coalesce field.

Adapter Changes

Change the XML mapping files to use the new custom field

1. Open the push or population XML mapping file of your interest.
2. For each CI in the mapping file, map its UCMDB **global_id** value to the custom field instead of the **correlation_id** field.

Change the **class.model.coalesce.field** property in the **adapter.properties configuration**

1. Open the **adapter.properties** configuration file.
2. Set the **class.model.coalescce.field** property to the name of the custom field in ServiceNow.

ServiceNow Generic Adapter Enhancements in CP27

This section provides information about new features and enhancements added to ServiceNow enhanced adapter in Content Pack 27.

- The following topologies are supported by the adapter in terms of population:
 - Referenced Topology – Implemented the [standard population](#).
 - Referencing Topology – Added the following new topology population supportability:
["SNOW Business Service Referencing Topology Population 1.0" on page 275](#)
 - Relationship-based Topology – Added the following new topology population supportability:
 - ["SNOW Business Application Relationship Topology Population 1.0" on page 277](#)
 - ["SNOW BA and BS to Linux Relationship Topology Population 1.0" on page 279](#)

- Added the following new properties in the **adapter.properties** file:

- Jakarta and Kingston supportability

`class.model.use.internal.tables` – Always uses internal schema tables.

If you set `class.model.use.internal.tables=true`, the ServiceNow enhanced adapter loads the ServiceNow class model from the meta data files packed in this adapter. Otherwise, the adapter queries the ServiceNow instance that is integrated and downloads it.

It is recommended to set this property to `true` if you have customized the OOTB ServiceNow class model.

- Relationship-based topology population

- `preload.rel.table` - Preloads relationship tables when the adapter starts.
- `force.reload` - Forces the reloading of the **CMDB_CI_REL** table cache.

- Advanced logging

- `show.interface.rtn` – Displays Result Tree Nodes (RTNs) just before returning them to the interface in the debug log.

The adapter's logger is improved when it is set to `DEBUG`. The RTNs are printed when they are passed to UCMDB's identification engine. You need to activate the `show.interface.rtn` property.

The following events are logged during a Population scenario:

- When the adapter is initialized
- When the table structure is fetched
- When the RTN records are created from ServiceNow responses
- When the RTNs are added to the cache
- When the integration job starts and completes

Advanced logging parameters and appenders for the standard population of relationship for the referenced topology:

```
# To add additional debugging for specific parent and child CI when
retrieving relationships

# in case of Standard topology population NOT! Relationship topology
population

# you must also add this appender to the <UCMDB_
DataFlowProbe>\conf\log\fcmdb.properties file

#log4j.category.servicenow.generic.adapter.debug.ids=TRACE,servicenow.g
eneric.adapter.debug.ids.appender

#log4j.appender.servicenow.generic.adapter.debug.ids.appender=com.mercu
ry.topaz.cmdb.shared.base.log.BetterRollingFileAppender

#log4j.appender.servicenow.generic.adapter.debug.ids.appender.File=${lo
gs.dir}/debug.ids.servicenow.generic.adapter.log

#log4j.appender.servicenow.generic.adapter.debug.ids.appender.MaxFileSi
ze=50240KB

#log4j.appender.servicenow.generic.adapter.debug.ids.appender.MaxBackup
Index=${def.files.backup.count}

#log4j.appender.servicenow.generic.adapter.debug.ids.appender.layout=or
g.apache.log4j.PatternLayout

#log4j.appender.servicenow.generic.adapter.debug.ids.appender.layout.Co
nversionPattern=${msg.layout}
```

- debug.parent.sys.id – The debug parent sys_id. For example,
debug.parent.sys.id=4961134adb218b0044d476231f9619e9
- debug.child.sys.id – The debug child sys_id. For example,
debug.parent.sys.id=08421b4adb218b0044d476231f9619a1
- sysparm.display.value – Uses display label instead of sys_id for lookup attributes. This works only for one level population; otherwise the adapter cannot use the display value for dot walking.

To resolve an existing issue where **reference** fields display the value of sys_id instead of the actual values, the following configuration is added into **ServiceNowConfig.xml**.

```
sysparm_display_value=true
```

- If this value is set to true, the RTN contains the display value of the **reference** field.
- If this value is set to false, the sys_id value is used.

- SSL proxy improvements

`ssl.trust.all=true` – Trust all remote certificates. This is necessary if you want to have a proxy enabled with an SSL connection but you do not want to add the certificates to the UCMDB Trust Store.

The `ssl.trust.all=true` option is used if authentication is not a concern, for example, if you require private secure communications, and want to save the time and expense in obtaining a CA certificate, and use a self-signed certificate.

This option can also be used in debugging SSL connection issues. For example:

```
ERROR [pool-5-thread-1] - Unable to execute rest request.  
javax.net.ssl.SSLHandshakeException: Remote host closed connection during  
handshake  
at sun.security.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:992)  
at sun.security.ssl.SSLSocketImpl.performInitialHandshake  
(SSLSocketImpl.java:1375)  
...  
ERROR [pool-5-thread-1] - Service unavailable. Remote host closed  
connection during handshake  
Retrying after 60 seconds.
```

- Retry mechanism improvements

The connector's retry mechanism works when a SOAP Fault error occurs. In addition to response codes, the adapter continues to retry whenever a timeout is reached.

This behavior is enabled by default and it improves the adapter's fault tolerance.

Changes in ServiceNowConfig.xml

In this version, the configuration for populating relationship topologies and referencing topologies is specified in the **ServiceNowConfig.xml** configuration file. The parameters are described as follows and are used to build UCMDB relationships. The different parameters are described based on the different scenarios that they are used for.

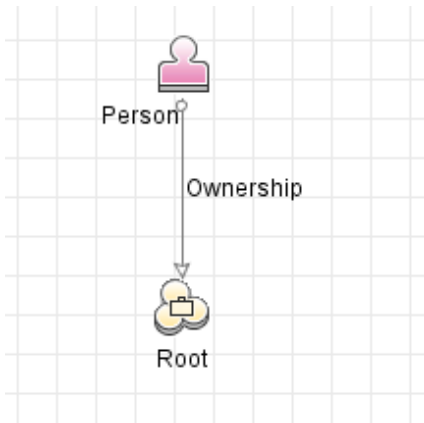
1. Referencing topology example:

SNOW Business Service Referencing Topology Population 1.0

Example

```
<!-- REFERENCING topology example -->
<!-- REFERENCING topology example -->
  <!-- <ci ciType="business_service" targetTable="cmdb_ci_service">
    <relations>
      <relation childCiType="person" type="ownership"
populationType="referencing" childCiTargetTableField="owned_by"/>
    </relations>
  </ci>
  <ci parentRef="owned_by" ciType="person" targetTable="sys_user">
  </ci>
```

TQL query



Mapping file

```
<target_entities>
  <source_instance query-name="SNOW Business Service Referencing Topology
Population 1.0" root-element-name="cmdb_ci_service">
    <target_entity name="Root">
      <target_mapping datatype="STRING" name="name" value="cmdb_ci_service
['name']"/>
      <target_mapping datatype="STRING" name="global_id" value="cmdb_ci_
service['correlation_id']"/>
    </target_entity>
  </source_instance>
</target_entities>
```

```
<target_mapping datatype="STRING" name="owned_by" value="cmdb_ci_
service['owned_by']"/>

<target_mapping datatype="STRING" name="sys_id" value="cmdb_ci_service
['sys_id']"/>

</target_entity>

<for-each-source-entity count-index="i" source-entities="cmdb_ci_
service.sys_user">

  <target_entity name="Person">

    <target_mapping datatype="STRING" name="name" value="cmdb_ci_
service.sys_user[i]['name']"/>

    <target_mapping datatype="STRING" name="global_id" value="cmdb_ci_
service.sys_user[i]['correlation_id']"/>

    <target_mapping datatype="STRING" name="sys_id" value="cmdb_ci_
service.sys_user[i]['sys_id']"/>

  </target_entity>

</for-each-source-entity>

</source_instance>

</target_entities>
```

Note: The mapping for this job **SNOW Business Service Referencing Topology Population 1.0** does not cover the OOTB identification criteria. If you want to make them work in OOTB perspective, you need to change the identification for the corresponding CIT by the key attribute **name**.

2. Relationship topology example:

- **SNOW Business Application Relationship Topology Population 1.0**

Example

```
<!-- RELATIONSHIP topology example -->

<ci ciType="business_application" targetTable="cmdb_ci_app1">

  <relations>

    <relation childCiType="node" type="containment"
populationType="relationship" reverseParenthood="false"
targetRelType="Runs on::Runs"/>

  </relations>

</ci>
```

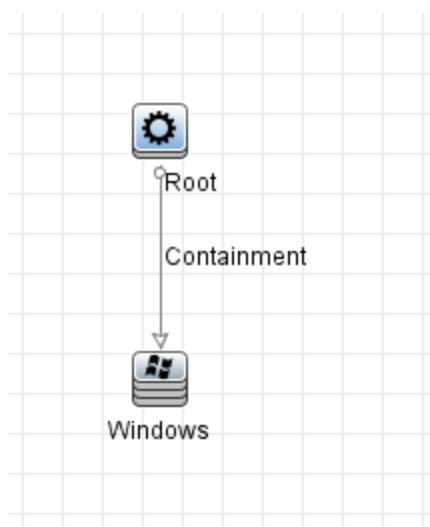
```
<relation childCiType="nt" type="containment"
populationType="relationship" reverseParenthood="false"
targetRelType="Runs on::Runs"/>

<relation childCiType="unix" type="containment"
populationType="relationship" reverseParenthood="false"
targetRelType="Runs on::Runs"/>

</relations>

</ci>
```

TQL query



Mapping file

```
<target_entities>

  <source_instance query-name="SNOW Business Application Relationship
Topology Population 1.0" root-element-name="cmdb_ci_appl">

    <target_entity name="Root">

      <target_mapping datatype="STRING" name="name" value="cmdb_ci_appl
['name']"/>

      <target_mapping datatype="STRING" name="global_id" value="cmdb_ci_
appl['correlation_id']"/>

    </target_entity>

    <for-each-source-entity count-index="i" source-entities="cmdb_ci_
appl.cmdb_ci_win_server">

      <target_entity name="Windows">
```

```
<target_mapping datatype="STRING" name="name" value="cmdb_ci_
appl.cmdb_ci_win_server[i]['name']"/>

<target_mapping datatype="STRING" name="global_id" value="cmdb_
ci_appl.cmdb_ci_win_server[i]['correlation_id']"/>

</target_entity>

</for-each-source-entity>

</source_instance>

</target_entities>
```

Note: The `reverseParenthood` parameter by default is `false`, which means that the Root item in UCMDB TQL is the Parent in the ServiceNow relationship pair. If the value is set to `true`, the Parent in the ServiceNow is the Root item in UCMDB TQL.

- **SNOW BA and BS to Linux Relationship Topology Population 1.0**

Example

```
<ci ciType="unix" targetTable="cmdb_ci_linux_server">

  <relations>

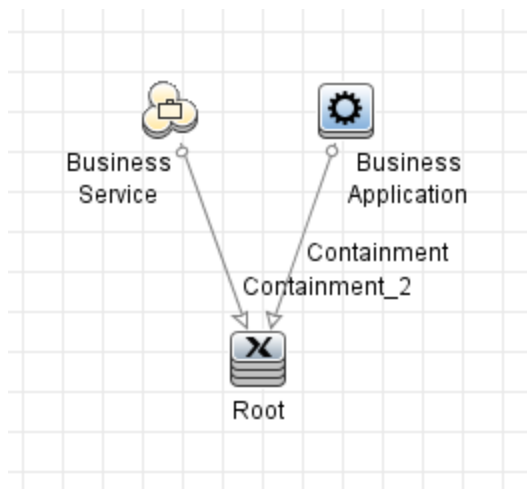
    <relation childCiType="business_application" type="containment"
populationType="relationship" reverseParenthood="true"
targetRelType="Runs on::Runs"/>

    <relation childCiType="business_service" type="containment"
populationType="relationship" reverseParenthood="true"
targetRelType="Runs on::Runs"/>

  </relations>

</ci>
```

TQL query



Mapping file

```
<target_entities>
  <source_instance query-name="SNOW BA and BS to Linux Relationship
Topology Population 1.0" root-element-name="cldb_ci_linux_server">
    <target_entity name="Root">
      <target_mapping datatype="STRING" name="name" value="cldb_ci_linux_
server['name']"/>
      <target_mapping datatype="STRING" name="global_id" value="cldb_ci_
linux_server['correlation_id']"/>
    </target_entity>
    <for-each-source-entity count-index="i" source-entities="cldb_ci_
linux_server.cldb_ci_appl">
      <target_entity name="BusinessApplication">
        <target_mapping datatype="STRING" name="name" value="cldb_ci_
linux_server.cldb_ci_appl[i]['name']"/>
        <target_mapping datatype="STRING" name="global_id" value="cldb_
ci_linux_server.cldb_ci_appl[i]['correlation_id']"/>
      </target_entity>
    </for-each-source-entity>
    <for-each-source-entity count-index="i" source-entities="cldb_ci_
linux_server.cldb_ci_service">
```



```
<target_entity name="BusinessService">
  <target_mapping datatype="STRING" name="name" value="cmdb_ci_
linux_server.cmdb_ci_service[i]['name']"/>
  <target_mapping datatype="STRING" name="global_id" value="cmdb_
ci_linux_server.cmdb_ci_service[i]['correlation_id']"/>
</target_entity>
</for-each-source-entity>
</source_instance>
</target_entities>
```

Note: It is very important to note that configurations inside the **ServiceNowConf** file are dependable to each other. So, for example, if you are populating referencing and relationship topologies, and the CITs that you are fetching overlap, the best approach is to combine the configurations in one configuration. For example:

```
<ci ciType="business_service" targetTable="cmdb_ci_service">
  <relations>
    <relation childCiType="node" type="containment"
populationType="relationship" targetRelType="Runs on::Runs"/>
    <relation childCiType="nt" type="containment"
populationType="relationship" targetRelType="Runs on::Runs"/>
    <relation childCiType="unix" type="containment"
populationType="relationship" targetRelType="Runs on::Runs"/>
    <relation childCiType="person" type="ownership"
populationType="referencing" childCiTargetTableField="owned_by"/>
  </relations>
</ci>
<ci parentRef="owned_by" ciType="person" targetTable="sys_user">
</ci>
```

Topology Population Flow

The adapter's population works by receiving a structure from the expected topology from the UCMDB platform or framework. The structure is tree-like, containing the root node, children and the table's

attributes.

Based on that, the population processes the results in the following order:

1. The Root node
2. The Child nodes
3. Recursively process (if any) children of the child nodes.

Procuring the root node from ServiceNow almost always employs the same mechanism – a query is formed to the required table that gets all of the records from the table. Inside the URL, it should be specified which attributes or fields it expects from ServiceNow.

The adapter then forms a list of Result Tree Nodes (RTNs) which are the representation that UCMDB expects for the CIs.

If there are child elements, they are assigned or linked to their parent element. As a result, the list of RTNs that the adapter returns will contain the entire topology.

Specifics about how parents are related to children are later determined by the UCMDB framework based on the mapping files and original TQL query.

Standard (Referenced) Population

The standard population is the mode that the adapter used before the changes in this version. It can be summed up with the following:

1. Procure the root node for certain ServiceNow tables with required attributes. Store the results in the cache for the specific table.

Example URL:

`api/now/table/<parent_table>?sysparm_fields=<fields_required_by_attributes>`

2. If any children are specified, procure them from ServiceNow. The call should also request the attribute of the children that references the parent's `sys_id`. In the **ServiceNowConfig.xml** file, this is the `childCiTargetTableField` attribute. The adapter filters only those children whose `childCiTargetTableField` references the parent's `sys_ids` from the last step. The parent's `sys_ids` are contained in the cache.

Example URL:

`api/now/table/<child_table>?sysparm_fields=<fields_required_by_attributes>&sysparm_query=<childCiTargetTableField>IN<parent_sys_ids_from_previous_step>`

3. Based on the values of the attribute mentioned above, the adapter finds request for the parent's table from the cache, and then fetch the parent RTNs that match the children's sys_ids.
4. Assign the respective children to the respective parent CIs.

Referencing topology Population

Referencing is essentially the same as standard population; however, the difference is that the parent CIs are not determined by the sys_id field, but rather by a custom field that can be defined in the **ServiceNowConfig.xml** as parentRef. The configuration looks like as follows:

```
<ci ciType="business_service" targetTable="cmdb_ci_service">
  <relations>
    <relation childCiType="person" type="ConsumerProvider"
      populationType="referencing" childCiTargetTableField="owned_by"/>
  </relations>
</ci>

<ci parentRef="owned_by" ciType="person" targetTable="sys_user">
</ci>
```

The population flow is as follows:

1. Procure the root node. This step determines if any of the children of the root node should be procured using referencing topology. If there are any, get their parentRef value from the configuration. Then request the value of the parentRef field from ServiceNow. This is the field that will be used to get the children at the next stages.

Example URL:

api/now/table/<parent_table>?sysparm_fields=<fields_required_by_attributes+parentRef_fields_of_children>

2. Go over the parent's parentRef fields and collect the children's sys_ids. Then use these sys_ids to get the children from ServiceNow.

Example URL:

api/now/table/<child_table>?sysparm_fields=<fields_required_by_attributes>&sysparm_query=sys_idIN<parentRef_field_values_from_previous_step>

3. Match the child entries to the parent by the parentRef values. The parent's parentRef field should equal the child's sys_id. Once the RTNs are assembled, they are ready to return.
4. In certain cases, the ServiceNow topology and UCMDB topology have the inverse direction of the relationship. Therefore, you can keep the parent and child that you want in the adapter, and still get the proper result after the mapping is finished.

Relationship Topology Population

Relationship topology population relies on a prerequisite during the adapter startup that the adapter loads the **the cmdb_rel_ci** table.

1. Prerequisite

During the adapter startup, the **cmdb_rel_ci** table is loaded from ServiceNow. On the first launch, the table is always fetched from ServiceNow. However, on any subsequent launches, the adapter checks if the size of the table in ServiceNow is the same as that of your file. If not, the adapter updates it.

2. Configuration for relationship topology in the **ServiceNowConfig.xml** looks like as follows.

```
<ci ciType="business_application" targetTable="cmdb_ci_appl">
  <relations>
    <relation childCiType="node" type="usage" populationType="relationship"
reverseParenthood="false" targetRelType="Runs on::Runs"/>
    <relation childCiType="unix" type="containment"
populationType="relationship" targetRelType="Runs on::Runs"/>
  </relations>
</ci>

<preloaded_types>
  <relationship_type name="Runs on::Runs" />
  <relationship_type name="Owns::Owned by" />
</preloaded_types>
```

targetRelType indicates the direction of the relationship in ServiceNow. It can be between the parent CI type (in this case cmdb_ci_appl) and the child CI type (Node or UNIX).

The preloaded types must match the ones used as `targetRelType` in `Prerequisite`. This happens after the adapter is initially loaded.

3. Fetching the root CI happens just like in standard mode.

Example URL:

`api/now/table/<parent_table>?sysparm_fields=<fields_required_by_attributes>`

4. After the root CI properties are discovered, if `populationType` is `relationship`, the flow is as follows:
 - a. Get the relationship between parent and child.
 - b. Filter out only those entries in the **`cmdb_rel_ci`** table cache that use this relationship.
 - c. If `reverseParenthood` is set to `true`, match all the IDs from the step 3 to the IDs in the **`child`** column of the **`cmdb_rel_ci`** table, and then take the IDs from the **`parent`** column.
 - d. If `reverseParenthood` is set to `false`, match all the IDs from step 3 to the IDs in the **`parent`** column of the **`cmdb_rel_ci`** table, and then take the `sys_ids` from the **`child`** column.
5. At the end of either step 4c or 4d, the job has a list of the child IDs getting from ServiceNow. They are then requested using a filter by **`sys_id`**.

Example URL:

`api/now/table/<child_table>?sysparm_fields=<fields_required_by_attributes>&sysparm_query=sys_idIN<ids_from_step_4>`

Troubleshooting and Limitations – ServiceNow Integration Using Enhanced Generic Adapter

This section describes troubleshooting and limitations for the UCMDB-ServiceNow integration using the enhanced generic adapter.

- ["Troubleshooting – ServiceNow Integration Using Enhanced Generic Adapter" on the next page](#)
- ["Limitations – ServiceNow Integration Using Enhanced Generic Adapter" on page 289](#)

Troubleshooting – ServiceNow Integration Using Enhanced Generic Adapter

This section describes troubleshooting for the UCMDB-ServiceNow integration using the enhanced generic adapter.

When troubleshooting UCMDB-ServiceNow integration issues, make sure that the corresponding log severities are set for the Data Flow Probe. Change the content of the files in folder **<Data flow probe>\conflog** in order to configure the logging correctly. The log files containing helpful information are these:

- **<Data flow probe>\runtime\log\servicenow.generic.adapter.log** – this one contains the log entries specific only to the adapter itself. See **adapter.logger** for information on how to enable this. It is recommended to set the log to TRACE level.
- **<Data flow probe>\runtime\log\WrapperProbeGw.log** – this one contains the log entries for everything happening on the Data Flow Probe itself.
- **<Data flow probe>\runtime\log\fcmdb.adapters.log** and **<Data flow probe>\runtime\log\fcmdb.adapters.<integration_point_name>.log** – these contain the log entries specific to the integration points themselves.

These are the common failures and ways how to resolve them:

1. **PROBLEM:** “Test connection” in the “Edit Integration Point” dialog brings up an error dialog

Workaround: To resolve the issue, do the following:

- a. Make sure the Adapter Properties are set up correctly.
- b. Make sure the ServiceNow instance is accessible from the Data Flow Probe.
- c. Make sure the **connector.class** property in the **adapter.properties** file is set to REST connector.
- d. Check the adapter logs that the UCMDB request for connection test got to the adapter. If the request came through then in the logs, there should be an INFO entry "Testing ServiceNow connection". If the test was successful in the adapter itself, then in the logs there should also be an INFO entry "Test successful".

2. **PROBLEM:** Bringing up the mapping tool editor displays an empty “External Class Model”.

Workaround: To resolve the issue, do the following:

- a. Check [step 1.a](#) through [step 1.c](#).
 - b. Check that the integration is activated in the Edit Integration Point dialog box.
 - c. Check that the request came to the adapter from UCMDB. In the adapter log there should be an INFO entry "Getting class model" and later on the DEBUG entries "Populating ServiceNow schema", "Building CI types" and "Class model created". In case of an error on the adapter side, the adapter log will clearly show it.
 - d. Check that the integration user has the required ServiceNow ACLs in order to read the required tables (see "[External Class Model Flow](#)").
3. **PROBLEM:** Some tables are missing from external class model.

Workaround: To resolve the issue, do the following:

- a. Ensure the tables exist in ServiceNow.
 - b. Ensure the tables are configured as part of the external class model in the **adapter.properties** file. The property you should check for this is [class.model.table.prefixes](#). If this is not the case then the adapter disregards these tables. In order to display them you need to include the referenced tables in the external class model also. After this is done, restart the adapter in order to display these fields.
 - c. In the **adapter.properties** file set the property [class.model.incremental.update](#) to **false** and see if this solves the problem.
4. **PROBLEM:** Some table fields are not displayed in external class model.

Workaround: To resolve the issue, do the following:

- a. Ensure that the fields exist in ServiceNow for the tables and that the integration user has the required ACLs to retrieve these fields.
 - b. Check if these fields are references to the tables that are not a part of the external class model (see the workaround for issue [3](#)). If so, the adapter disregards these fields. In order to display them, you need to include the referenced tables in the external class model also. After this is done, restart the adapter in order to display these fields.
 - c. In the **adapter.properties** file set the property [class.model.incremental.update](#) to **false** and see if this solves the problem.
5. **PROBLEM:** While executing the push job some or all of the UCMDB CIs are ignored.

Workaround: To resolve the issue, do the following:

- a. Make sure the Adapter Properties are set up correctly.
- b. Make sure the ServiceNow instance is accessible from the Data Flow Probe.

- c. Check that the integration is activated in the Edit Integration Point dialog box.
 - d. Make sure the **connector.class** property in the **adapter.properties** file is set to either SOAP or REST connector.
 - e. Check that the request came to the adapter from UCMDB. If the request came from UCMDB to the adapter, then in the adapter log there should be an entry like "Executing pushTreeNodes ...".
 - f. In the mapping files, ensure that the CIs are properly mapped (see ["Push Integration Mechanism"](#)).
 - g. In case of an error on the adapter side, all the logs mentioned above should clearly show the error.
 - h. In case one of the HTTP requests to ServiceNow timed out, then do one of the following:
 - Lower the value of push chunk size in UCMDB, or
 - In case you are using SOAP connector, try to increase the **glide.soap.request_processing_timeout** property value in ServiceNow, or
 - In case you are using REST connector, then try to change the [REST Request Timeout transaction quota](#). If this is not possible, then use the SOAP connector instead.
6. **PROBLEM:** While executing the push job duplicates are created in ServiceNow side for the UCMDB CIs.
- Workaround:** In the mapping file, ensure that the push goes through import set tables and the **correlation_id** (or custom) fields are properly mapped to UCMDB **global_id** fields (see ["Push Integration Mechanism"](#)).
7. **PROBLEM:** While executing the push job, deleted CIs in UCMDB are not deleted from ServiceNow.
- Workaround:** To resolve the issue, do the following:
- a. In [ServiceNowConfig.xml configuration](#), make sure that a valid ServiceNow staging table to target table mapping is specified in CI/Relation Type Mappings (see ["Push Integration Mechanism"](#)).
 - b. In case of an error on the adapter side, all the logs mentioned above should clearly show the error.
8. **PROBLEM:** While executing population, the CIs are not inserted into UCMDB.
- Workaround:** To resolve the issue, do the following:

- a. Make sure the Adapter Properties are set up correctly.
 - b. Make sure the ServiceNow instance is accessible from the Data Flow Probe.
 - c. Check that the integration is activated in the Edit Integration Point dialog box.
 - d. Make sure the **connector.class** property in the **adapter.properties** file is set to REST connector. Only REST is supported at the moment.
 - e. Check that the request came to the adapter from UCMDB. If the request came from UCMDB to the adapter, then in the adapter log there should be an entry like "Executing populate ...".
 - f. In case of an error on the adapter side, all the logs mentioned above should clearly show the error.
9. **PROBLEM:** While executing the population job, the ServiceNow CI **correlation_id** (or custom) fields are not updated by the adapter.

Workaround: To resolve the issue, do the following:

- a. Check if the **population.return.global_id** property in the **adapter.properties** file is set to **false**. If so, the adapter will not update the ServiceNow. Set this parameter to **true**.
 - b. Ensure that the ServiceNow **sys_id** field is mapped to UCMDB CI **sys_id** property in the mapping file (see ["Population Flow"](#)).
10. **PROBLEM:** External class model takes a lot of time to load or refresh.

Workaround: This is a common case. The ServiceNow class model is large, so a lot of time is spent in communication between the adapter and ServiceNow as well as between the adapter and the UCMDB. See [Incremental Update](#) for information about how to get around this issue.

Limitations – ServiceNow Integration Using Enhanced Generic Adapter

This section describes limitations for the UCMDB-ServiceNow integration using the enhanced generic adapter.

Push integration

- **Limitation:** Deleted CI Relationships in UCMDB cannot be deleted in ServiceNow when using the ServiceNow Enhanced Generic Adapter.

Chapter 20: Troux Integration

This chapter includes:

Overview	291
Integration Overview	291
Supported Versions	292
Use Cases	292
How to Work with the Troux Push Adapter	293
How to Run a Troux Population Job	298

Overview

Troux is a vendor in the EA (Enterprise Architecture) tools market. EA tools allow business users to understand the gaps between business demands and initiatives. Reviewing how your fixed budget aligns to business capabilities and how your discretionary spending is allocated across initiatives. Future-state scenario investigation can be accomplished prior to locking down your roadmap.

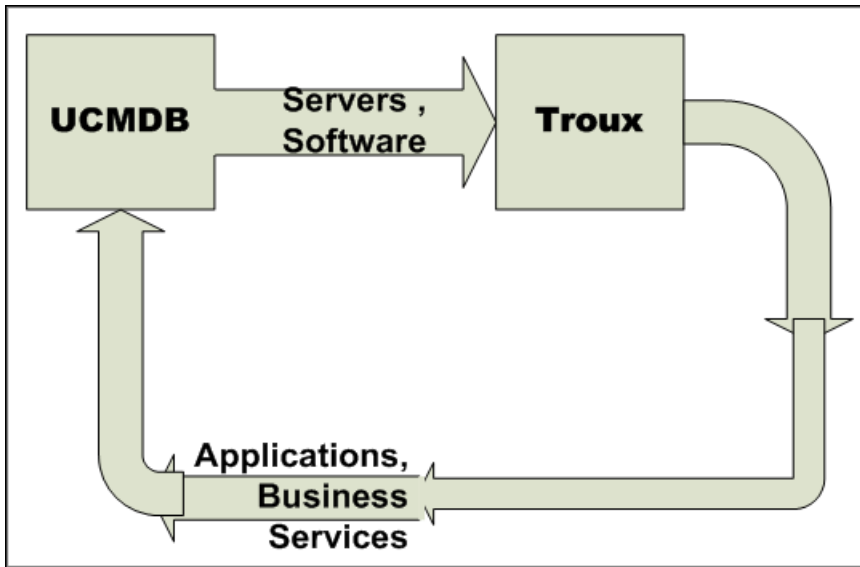
Although many use cases can be achieved using EA tools, two specific use cases were chosen for the UCMDB-Troux integration. This does not preclude additional use cases in the future. Depending on the use case, a provider of record is determined. For example, UCMDB would be the provider of record for inventory information such as the server operating system, server hardware, database, and other infrastructure CIs. Troux on the other hand provides component lifecycles for server operating system, server hardware, and database versions.

Integration Overview

UCMDB-Troux integration consists of two independent, bi-directional parts: the **Troux Push Adapter**, and the **Troux Population Adapter**.

- The Troux Push Adapter in UCMDB replicates CIs and relationships to Troux. The Troux Push Adapter is necessary to achieve both the Technology Standards Assessment and Business Impact Analysis use cases discussed in the introduction above. The adapter also allows the user to push to Troux CIs that are aged out of UCMDB or deleted.
- The Troux Population Adapter pulls CIs and relationships from Troux to UCMDB. It is necessary only for the Business Impact Analysis use-case.

Data transfer occurs using XML files between configured directories. Mapping files are used to apply conversion from TUX format to UCMDB and vice versa.



Supported Versions

Supported versions of the products are listed below.

Target Platform	DFM Protocol	UCMDB Version
Troux 9.x	None	9.02 and later

Use Cases

The use cases chosen for UCMDB-Troux integration are:

- **Technology Standards Assessment.** The ability to look at a lifecycle of software products to determine viability within an enterprise.
- **Business Impact Analysis.** Definition of the definitive source of application CIs to align IT with business. These application CIs in Troux are related to server operating system, server hardware, database, and other CIs discovered by UCMDB. Impact Analysis can be determined using application, business function, and organization for planned change or unplanned disruption of service.

How to Work with the Troux Push Adapter

This adapter allows replication of CIs and links from UCMDB to Troux. This is accomplished by definition of queries and mapping files that define the CIs to be transferred and the naming/mapping of CIs and attributes to Troux components. This adapter also allows the user to push to Troux CIs that are aged out of UCMDB or deleted.

This task includes the following steps:

Define queries

1. Create a query that defines the CIs and attributes you want to replicate to Troux. Two example queries are supplied in the **Integration > Troux** folder.

For details, see "Topology Query Language" in the *Modeling section of the UCMDB Help*.

2. Define the properties of each of the CITs.

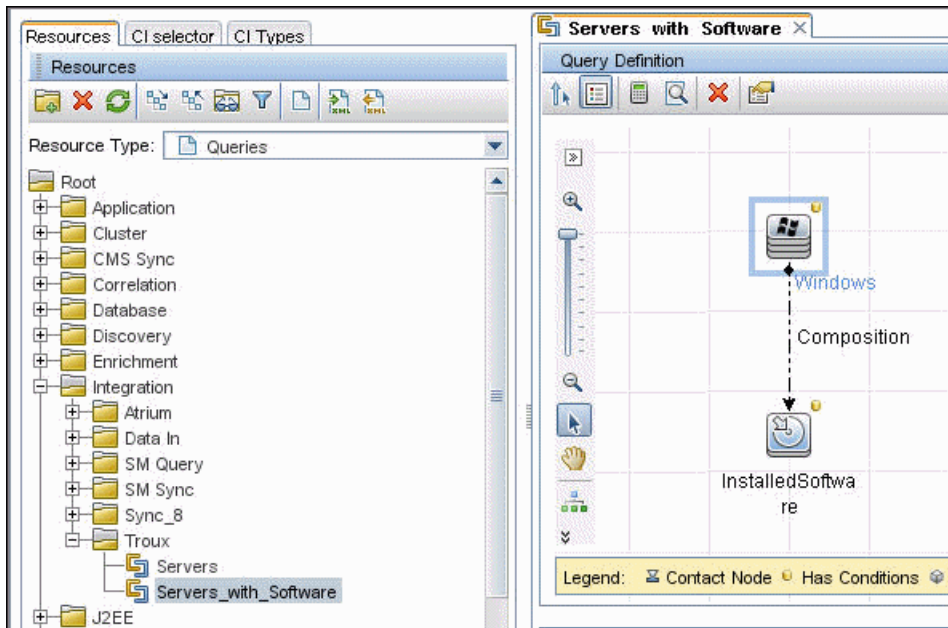
Note: This step is critical to the operation of the push adapter. You must define the attributes that will be transferred to Troux.

For details, see "Query Node/Relationship Properties Dialog Box" in the *Modeling section of the UCMDB Help*.

- a. Define the criteria for the Query Node properties
- b. Define the advanced properties for each of the attributes.
- c. Select attributes to be transferred to Troux.

Example: Computers_for_Troux

In this example, the query requests UCMDB to send all computers with installed software to Troux. You must define the mapping file with the same name as the query in order for the push adapter to recognize the query.



Create mapping files

A mapping file is the translation template that defines the CITs and the relationships to be converted from UCMDB to Troux. For the push adapter to create output, this mapping file must have definitions for the attributes and CITs or relationships for export. The mapping file is located in:

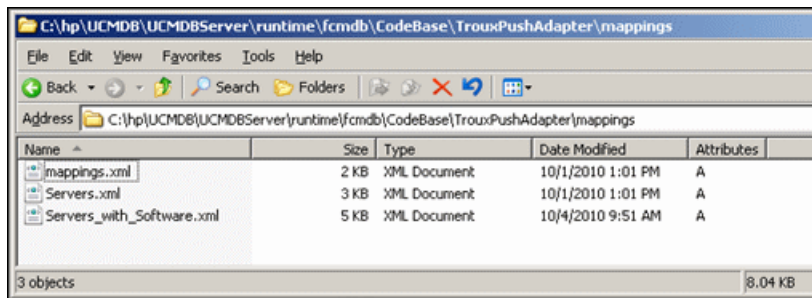
`UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\<adapter>\mappings`

where <adapter> is the name of the adapter.

The example mapping file and query (Servers_with_Software) included with the content package sends Windows computers with installed software to Troux, as expected by Troux. If your environment uses different CIs with Troux, make sure Troux handles those component types.

When you create the mapping file, give it exactly the same name as your query. For details about the mapping file options, see "Prepare the Mapping Files" in the *Developer Reference section of the UCMDB Help*. Use the example mapping files as reference examples for the mapping file creation.

Note: The definitions in the mapping file (<adapter>.xml) must be the direct CITs and relationships to be transferred to Troux. The mapping does not support inheritance of class types. For example, if the query is transferring nt CITs, the mapping file must have definitions for nt CITs, and not for general nodes or computers. That is, the definition must be an exact match for what to transfer.



Example

The image shows an XML configuration for a Troux integration. It includes annotations for defining source/target, source CIT, target CIT, and attributes/conversions.


```
<integration>
  <info>
    <source name="UCMDB" versions="9.0" vendor="HP" />
    <target name="Troux" versions="9.0" vendor="Troux" />
  </info>
  <targetcis>
    <!-- UCMDB computers to Troux Server -->
    <source_ci_type name="nt" namespace="" query="">
      <apioutputseq>1</apioutputseq>
      <target_ci_type name="Server">
        <targetprimarykey>
          <pkey>name</pkey>
        </targetprimarykey>
        <target_attribute name="action" datatype="char" >
          <map type="constant" value="find" />
        </target_attribute>
        <target_attribute name="name" datatype="char" >
          <map type="direct" source_attribute="name" />
        </target_attribute>
        <target_attribute name="External ID" datatype="char" >
          <map type="direct" source_attribute="id" />
        </target_attribute>
        <target_attribute name="description" datatype="char" >
          <map type="direct" source_attribute="discovered_os_name" />
        </target_attribute>
      </target_ci_type>
    </source_ci_type>
    <!-- UCMDB Installed Software to Troux Software Module -->
    <source_ci_type name="installed software" namespace="" query="">
```

Annotations in the image:

- Define Source / Target**: Points to the `<source>` and `<target>` tags in the `<info>` block.
- Define Source CIT**: Points to the `<source_ci_type>` tag.
- Define Target CIT**: Points to the `<target_ci_type>` tag.
- Define attributes and conversions**: Points to the `<target_attribute>` and `<map>` tags.

Create an integration point

This section describes how to create and run the job that replicates the data from UCMDB to Troux.

1. In UCMDB, navigate to **Data Flow Management > Integration Studio**.
2. Click the **New Integration Point** button to open the new integration point Dialog Box.
 - a. Click , select the **Data Push into Troux** adapter and click **OK**.
 - b. Enter the following information:

Name	Description
Integration Name	The name you give to the integration point.
Is Integration Activated	Select this check box to create an active integration point. You clear the check box if you want to deactivate an

Name	Description
	integration, for instance, to set up an integration point without actually connecting to a remote machine.
allowedComponentstodelete	<p>The name of the component which is to be deleted and pushed to Troux. The default is Server.</p> <p>UCMDB is the system of record for servers. The default value of Server is important to note because the Troux system model may not allow deletion of other CITs. If you modify this property, you must verify (a) the mapping file is setup correctly to send the deleted CIT to Troux, and (b) Troux is setup to handle the delete for the CIT you specify. Deletion of Server is the only OOTB CIT that has been verified.</p>
Data Flow Probe	The name of the Data Flow Probe.
TUX path	The location of the TUX output file (created when the integration job is run).

- c. Click **Test connection** to verify the connectivity, and click **OK**. If the connection fails, verify the provided information is correct.

Define an integration job

You use the Integration tab to define a job that uses the integration point that you just defined. For details, see "New Integration Job/Edit Integration Job Dialog Box" in the *Data Flow Management section of the UCMDB Help*.


1. Select the queries that you defined in ["Define queries" on page 293](#).
2. To enable deletion, select the **Allow Deletion** check box. This is a master delete **on/off** flag for the job that turns on and off the capability to send deletes.

Note: Deleted CITs are pushed depending on the changes to them in UCMDB. For example, if UCMDB discovers a computer, and that computer is deleted in UCMDB, this indicates a change of a deleted computer. And if this computer is part of the push query for Troux, with delete enabled, a delete is pushed to Troux the next time the push job is run.

3. Specify the job's schedule.
4. Click **OK**.

5. In the Integration Point pane, click **Save**. A full data push job will run according to schedule.

The Troux output file (TUX) is generated in the path that you specified in the Integration Properties for the job.

Note: To run the job again, click .

How to Run a Troux Population Job

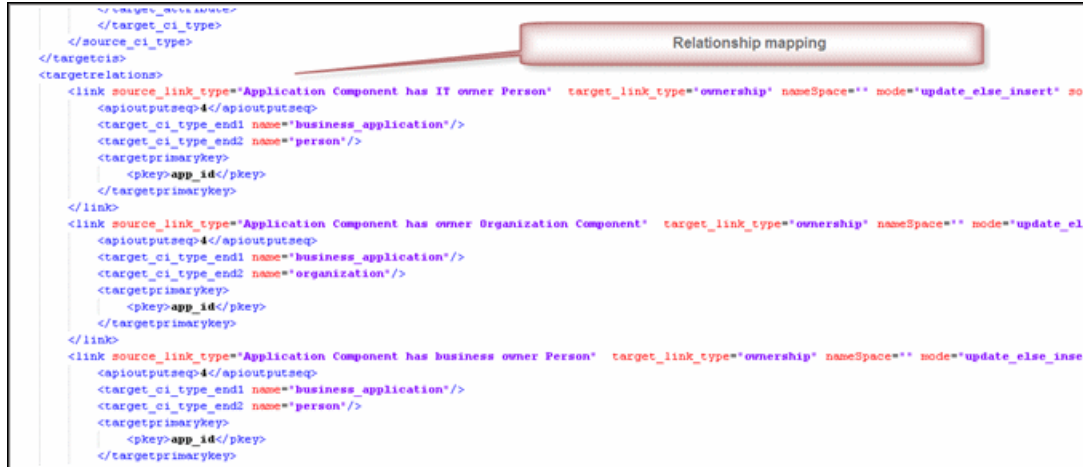
This task has the following steps:

Prerequisite - Create a mapping file

Create a mapping file that enables mapping of Troux components and relationships to UCMDB CITs and relationships.

The top section of the file defines the object or CIT mapping from Troux to UCMDB. The lower section defines the relationship mapping.

The out-of-the-box mapping file, **Troux_to_UCMDB.xml** (located in **\DataFlowProbe\runtime\probeManager\discoveryResources\TQLEXP\Troux\data**) contains the typical definitions for mapping the components and CITs with relationships.



Run the job

Note: For details on running an integration job, see "Integration Studio" in the *Data Flow Management* section of the *UCMDB Help*.

In the Integration Studio, create a new integration point.

1. Provide a name and description for the integration point.
2. Under **Integration Properties > Adapter**, select the **Population from Troux** adapter.
3. Edit the **TUX path** field, if required; this sets the location of the TUX output file.
4. Under **Adapter Properties > Data Flow Probe**, select the Data Flow Probe.
5. Under **Adapter Properties > Trigger CI instance** select:
 - a. **Select Existing CI** (if you have a valid, existing CI). The **Select Existing CI** pane appears. Select the CI or
 - b. **Create New CI** (if you need to create a new CI). The **Topology CI Creation** Wizard appears. Complete the creation of the CI using the Wizard.

Note: For details on the Topology CI Creation Wizard, see "Topology CI Creation Wizard" in the *Data Flow Management section of the UCMDB Help*.

6. Save the integration point.
7. Run the job.

Chapter 21: UCMDB to XML Adapter

This chapter includes:

Overview	302
Integration Mechanism	302
How to Export UCMDB to XML	302
Adapter	303
Troubleshooting and Limitations – UCMDB to XML Adapter	304

Overview

By using the UCMDB to XML adapter, it is possible to export the results (CIs and relationships) of TQL queries and convert these to XML files.

Integration Mechanism

After defining an integration point with the UCMDB to XML adapter, TQL queries can be added to the jobs of that integration point.

The adapter exports the result of the TQL queries into XML format, and creates XML files in the Export Directory (as predefined in the integration point).

How to Export UCMDB to XML

1. Prerequisite - General

- a. Create a directory on the UCMDB data flow probe system to which the adapter will write the exported XML files.
- b. In the Modeling Studio, create integration TQL queries for the data to be exported to XML. Ensure the queries return valid results.

2. Prerequisite - Create an integration point and integration job

In the Integration Studio, create a new integration point.


- a. Provide a name and description for the integration point.
- b. Under **Integration Properties > Adapter**, select **UCMDB to XML** adapter.
- c. Under **Adapter Properties > Export Directory** type an absolute path of the directory on the probe system where the adapter should export the XML files to.
- d. Under **Adapter Properties > Data Flow Probe Name**, select the name of the Data Flow Probe to be used.
- e. Click the **Test Connection** button to ensure the adapter can validate the defined export directory.
- f. Save the integration point.

- g. Under **Integration Jobs** add a new integration job. Edit the job to add integration queries under the job's definition.

For details on integration points, see **Integration Studio > Work with Data Push Jobs > Create an integration point** in the *Data Flow Management section of the UCMDB Help*.

- h. Click **OK** to save the integration job.
- i. Click the **Save** button to save the integration point.

3. Run the job

- a. To run the job ad hoc, select the integration job and click the  button. The job can also be configured to run on a schedule.
- b. Check the defined export directory on the probe system for the exported XML data, to ensure the queries create valid XML files.

Note: The XML files have time stamps in the format **YYMMDDHHMMSSZZZ**. If the integration query returns a large number of CIs, the export is by chunks of 1,000 CIs. Each chunk is a separate XML file, with the file for the last chunk having the string **EOD** (end of data) appended to it.

For details on running an integration job, see "Integration Studio" in the *Data Flow Management section of the UCMDB Help*.

Adapter

This job uses the adapter **UCMDB to XML** (XmlPushAdapter).

Used Script

pushToXml.py

Parameters

Parameter	Description
Export Directory	The absolute path (on the probe system) of the directory where the XML files will be exported to.
probeName	The name of the data flow probe to be used.

Troubleshooting and Limitations – UCMDB to XML Adapter

Problem: UCMDB to XML discovery fails with a timeout exception, with the following error in Data Flow Probe logs:

```
com.mysql.jdbc.PacketTooBigException: Packet for query is too large (14070323 > 10485760). You can change this value on the server by setting the max_allowed_packet' variable.
```

- **Cause of problem:** There is not enough DFM probe resources to export a large amount of CIs/relationships.
- **Solution #1:**

Decrease the **replication.chunk.size** setting for the XmlPushAdapter, as follows:

- a. In UCMDB, go to **Data Flow Management > Adapter Management**.
- b. In the list of resources, go **XMLExportAdapter > Adapters > XmlPushAdapter**.
- c. Right-click on **XmlPushAdapter** and choose **Edit Adapter source** from the pop-up menu.
- d. In the XML Editor that opens, find the adapter setting **replication.chunk.size** and decrease the from the current value. For example, if **replication.chunk.size** is currently 1000, you can decrease it to 100 as follows:

```
<adapter-settings>  
  <adapter-setting name="replication.chunk.size">1000</adapter-setting>  
</adapter-settings>
```

and decrease the chunk size. For example:

```
<adapter-settings>  
  <adapter-setting name="replication.chunk.size">100</adapter-setting>  
</adapter-settings>
```

- e. Click **Save**, and rerun the integration point.

- **Solution #2:**

Instead of one large integration job, create a few smaller jobs. To do this, create a few export TQL queries that will cover all the CIs to export and will be the basis of the smaller integration jobs.

Problem: Several sub-jobs fail when you add the Model Hierarchy query to the integration job.

Solution: The integration does not support TQLs with SubGraphs and TQLs with Full Path Compound links.

Send documentation feedback

If you have comments about this document, you can [contact the documentation team](#) by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

**Feedback on Discovery and Integrations Content Guide - Third Party Integrations
(Configuration Management System (CMS) Content Pack 28.00 (CP28))**

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to cms-doc@microfocus.com.

We appreciate your feedback!