HP Operations Manager Dependency Mapping Automation

For Windows®, UNIX®, and Linux Operating Systems

Software Version: 8.20

Extensibility Guide

Manufacturing Part Number: None

Document Release Date: July 2011

Software Release Date: July 2011

Documentation Updates

This manual's title page contains the following identifying information:

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

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

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

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

Support

You can visit the HP software support web site at:

www.hp.com/managementsoftware/services

HP software online support provides an efficient way to access interactive technical support tools. As a valued support customer, you can benefit by using the support site to:

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

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

To find more information about access levels, go to:

www.hp.com/managementsoftware/access_level

To register for an HP Passport ID, go to:

www.managementsoftware.hp.com/passport-registration.html

Contents

1 Introduction

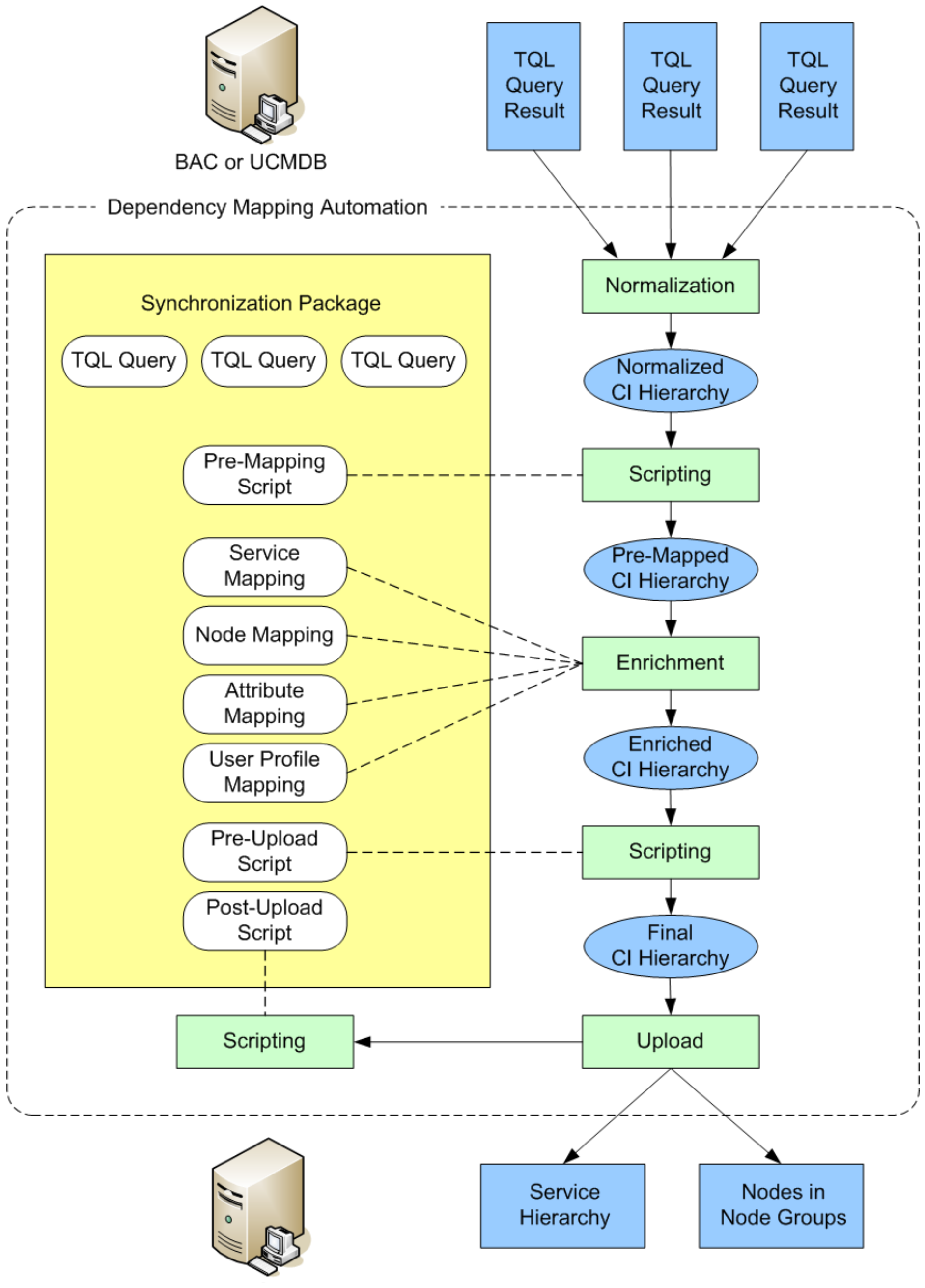
HP Operations Manager (HPOM) Dependency Mapping Automation (DMA) uses Topology Query Language (TQL) queries to retrieve data from Business Availability Center (BAC) or the Universal CMDB (UCMDB) to import nodes and create service hierarchies in HPOM. HPOM DMA includes synchronization packages that enable you to work with node and database information without any need for customizing.

To get your own UCMDB data reflected in HPOM, you extend HPOM DMA by adapting existing synchronization packages. You then create your own synchronization packages, enabling you to:

- Discover your environment as maintained by the UCMDB
- Automatically populate the HPOM Node Bank and Service View

Figure 1 shows how HPOM DMA links the UCMDB or BAC with HPOM.

Figure 1 HPOM DMA Linking HPOM with BAC or the UCMDB



Required Configurations

To integrate each type of application, such as a web server, you need to create the following configurations:

- **TQL Query**

  TQL queries define the node and service CIs in the UCMDB to be synchronized with HPOM.

  To find out how to create TQL queries for the UCMDB or BAC, see the documentation available with these products. For a basic overview, see Chapter 3, TQL Queries.

- **Synchronization Package**

  A synchronization package is a set of mapping rules to transform Configuration Items (CIs) into HPOM services and nodes and are used to synchronize specified services and nodes in HPOM with data from the UCMDB.

  There is a default synchronization package that is always active.

  For details, see Chapter 7, Synchronization Packages.

- **Service Type Definitions**

  The service type definition specifies which calculation and propagation rules are applied, and which icon is to be used to represent a CI from the UCMDB on the HPOM service tree. The subsequent synchronization updates the service instances in HPOM in accordance with the changes made to the service type definition.

  For details, see Chapter 4, Service Type Definition Files.

Installation Locations

The HPOM DMA installation installs the files in the following locations:

- **UNIX and Linux**
  - ***<InstallDir>***

    `/opt/OV/`
  - ***<DataDir>***

    `/var/opt/OV/`
  - ***<SharedDir>***

    `/var/opt/OV/shared`
- **Windows**
  (default location dependent upon HPOM for Windows location)
  - ***<InstallDir>***

    `C:\Program Files\HP\HP BTO Software`
  - ***<DataDir>***

    32-bit: `C:\Documents and Settings\All Users\Application Data\HP\HP BTO Software`

    64-bit: `C:\ProgramData\HP\HP BTO Software`
  - ***<SharedDir>***

    32-bit: `C:\Documents and Settings\All Users\Application Data\HP\HP BTO Software\shared`

    64-bit: `C:\ProgramData\HP\HP BTO Software\shared`

2 Extending HPOM DMA - An Example

HP Operations Manager Dependency Mapping Automation (HPOM DMA) uses TQL queries to retrieve data from BAC or the UCMDB, and import nodes and create service hierarchies in HPOM.

You can adapt and extend the HPOM DMA synchronization between the UCMDB and HP Operations Manager (HPOM). To extend the HPOM DMA synchronization, do the following:

- Specify TQL queries in the UCMDB to collect the content you want to synchronize.
- Adapt existing synchronization packages or create additional synchronization packages.

Using synchronization packages, you take the content collected by the TQL queries and use it to enrich the information in HPOM.

To get your own UCMDB data reflected in HPOM, you extend HPOM DMA by adapting existing synchronization packages. You then create your own synchronization packages. These packages enable you to discover your environment, as maintained by the UCMDB. They also enable you to automatically populate the HPOM Node Bank and Service View.

Getting Started

Before you start changing or extending synchronization packages, it is very important that you have a solid understanding of the following areas:

- The synchronization process used by HPOM DMA. See the *Installation and User's Guide*.
- The out-of-the-box synchronization packages delivered by HPOM DMA, especially the mappings applied by the default package. To view these mappings, go to:

  HPOM DMA Console → Extending DMA → Synchronization Packages
- Viewing data in the UCMDB and creating TQL queries to match the monitoring infrastructure.
- Creating HPOM service type definitions.
- HPOM User Profiles, if you want to automatically assign CIs to users.
- HPOM Nodes.

This guide includes two examples covering two aspects of extensions:

- **My Company**

  You have all the CIs you need, but you want to add a different view of your data. The My Company example shows you a geographical organization view of your company. The example guides you through the following steps:

  — Making Changes in the UCMDB

  — Adapting the HPOM DMA Enrichment Rules

  — Applying Enrichment Rules

  — Activating the My Company Synchronization Package
- **Web Server**

  You want to add your own application CIs that are not currently included in the synchronization. The Internet Information Services (IIS) web server example helps you to extend the synchronization data with nodes hosting IIS web servers and with an IIS service map in Service Navigator. The example guides you through the following steps:

- Prerequisites
- Designing the Web Server Synchronization Package
  - Analyzing the UCMDB Data to be Synchronized
  - Expected Outcome in HPOM
- Designing the Web Server Synchronization Package
  - Developing the Web Server Synchronization Package
  - Developing Your Synchronization Package

This chapter includes information on testing and fine-tuning your synchronization packages. It guides you through the following steps:

- Preparing for Testing
- Running a Test Synchronization
- Iteratively Improving Your Package

This chapter includes information on activating your synchronization packages. It guides you through the following steps:

- UCMDB Package
- HPOM DMA Package

This chapter includes some advanced topics:

- Adding a New Relationship Between CIs to Correctly Model Your Service Map which includes:
  - Adding the Depend Relationship
  - Creating an Enrichment Rule
  - Changing your TQL Query to Reflect the Change
- Combining Multiple Synchronization Packages
- External Services
- Defining STDs
- Scripting

My Company Example - Creating Customized Service Navigator Views

The My Company synchronization package helps you arrange your CIs in Service Navigator quickly and easily. For example, you can arrange your CIs to reflect the geographical organization of your company.

When you extend the synchronization, there are two general parts to consider:

- Specify TQL queries that contain the service view that you want to see in Service Navigator. This is contained in the `hpdmasamples.zip` UCMDB package.

  Make sure that the `hpdmasamples.zip` package is uploaded and deployed to the UCMDB/BAC system. For further information, see the *Installation and User's Guide*.

  For further information on modifying and previewing TQL queries, see Making Changes in the UCMDB.

- Create a synchronization package referring to the TQL associated queries enriching the data or customizing them with scripts. As the CIs are already part of the synchronization in this example, you need to refer to the TQL queries only. For further information, see Activating the My Company Synchronization Package on page 24.

Making Changes in the UCMDB

You can view TQL queries in the UCMDB and preview their results using the View Manager. If the results displayed are not as you expect, modify the TQL query and generate a new preview.

To view a TQL query and see a preview of the results, follow these steps:

1 Open the View Manager:

- **UCMDB 7 and 8:**

  Admin → Modeling → View Manager

- **UCMDB 9:**

  Modeling → Modeling Studio → Resources: Views

- **BAC:**

  Admin → Universal CMDB → Modeling → View Manager

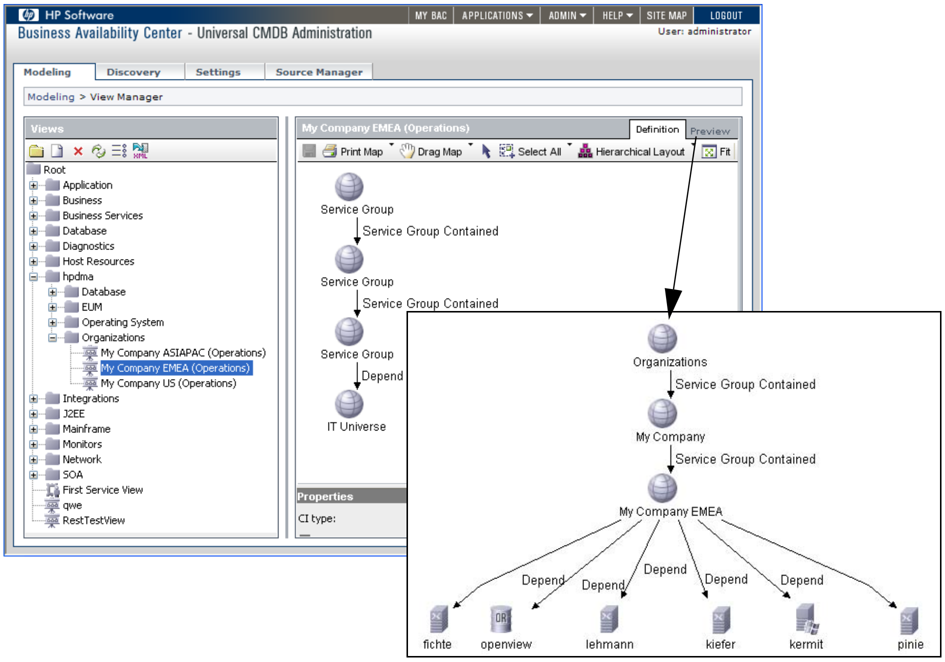
2 Select each of the My Example Company TQL queries in turn from the following:

  hpdma → Organizations

  The preview shows you the result, complete with the expected CIs.

Figure 2 illustrates the My Company EMEA TQL and its preview content.

Figure 2 Example of a My Company TQL Query



Adapting the HPOM DMA Enrichment Rules

HPOM DMA populates your HPOM node banks and service tree. If you are using BAC or UCMDB 7.5x or 8.0x, it is necessary that you adapt the node filters to suit to your specific environment. (The following adaption is not required for UCMDB 9.0x.)

To adapt the HPOM DMA enrichment rules, follow these steps:

1. Open the Enrichment Manager:
   - **UCMDB 7 and 8:**

     Admin → **Modeling** → **Enrichment Manager**
   - **BAC:**

     Admin → **Universal CMDB** → **Modeling** → **Enrichment Manager**
2. Expand the **hpdma** view from the Enrichment Rules list.

   The `MyCompany*Nodes` enrichment rules in the `hpdma` folders include filters in the node types that you must adapt to your environment.
3. For each `MyCompany*Nodes` enrichment rule, make sure that you are in the TQL mode, and select the node type (`Host`, `UNIX` or `Windows`).
4. From the context menu, open **Node Properties**.
5. Change the filter to your environment.

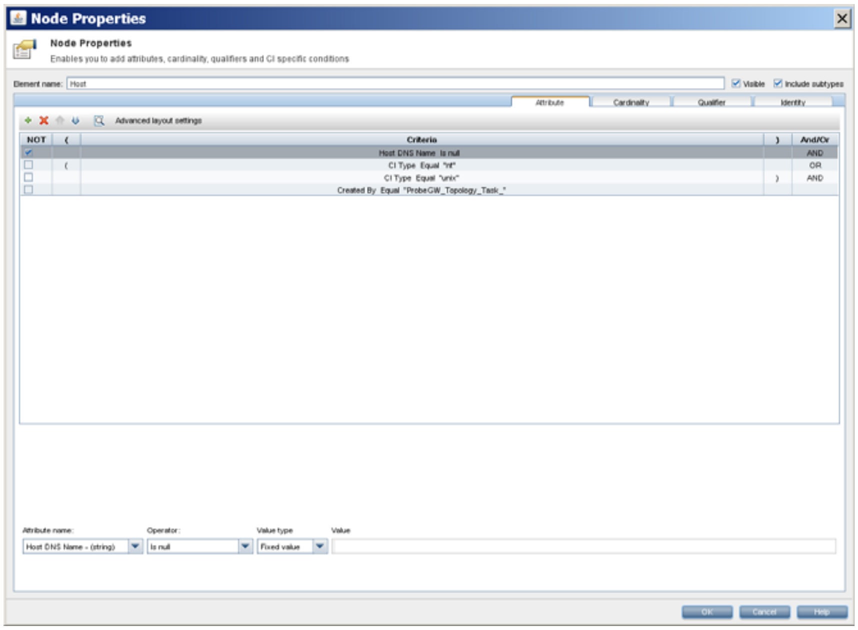
   By default, only nodes that include running UCMDB probes are selected. This is specified with the following node condition:

   ```
   Created By Equal "ProbeGW_Topology_Task_"
   ```

   You can delete this condition and add new ones to match your environment. For example, use the condition restrict on domain and specify domains to limit your search.

   Setting filters for TQL queries in the Node Properties window is shown in Figure 3.

Figure 3 Setting Node Conditions for TQL Queries


Node Properties

Node Properties

Enables you to add attributes, cardinality, qualifiers and CI specific conditions

Element name: Host ✓ Visible ✓ Include subtypes

NOT	(	Criteria	)	And/Or
✓		Host DNS Name Is null		AND
☐	(	CI Type Equal "nt"		OR
☐		CI Type Equal "unix"	)	AND
☐		Created By Equal "ProbeGW_Topology_Task_"		

Attribute name: Host DNS Name - (string)

Operator: Is null

Value type: Fixed value

Value:

After you have assigned nodes to your geography (EMEA, US, ASIAPAC), the databases hosted on those nodes are also assigned to the corresponding geographies.

Do not remove the `HOST DNS Name is null` filter, as it is required by HPOM DMA.

Applying Enrichment Rules

Apply each `MyCompany*` enrichment rule in the `hpdma` group.

➤ UCMDB enrichment rules only add the selected CI and relations to the UCMDB. If you change the enrichment rule, for example, change the list of nodes that should be added to a service group, then the already existing CIs in the group are not automatically removed.

Before you change an enrichment rule, you should remove the already existing CIs.

If you want to remove all currently existing assignments:

1. Select the enrichment rule in the CMDB Enrichment Manager.
2. Click **Remove Enrichment Results.**

However, this might also remove CIs and relations that have been added by other enrichment rules.

To apply enrichment rules:

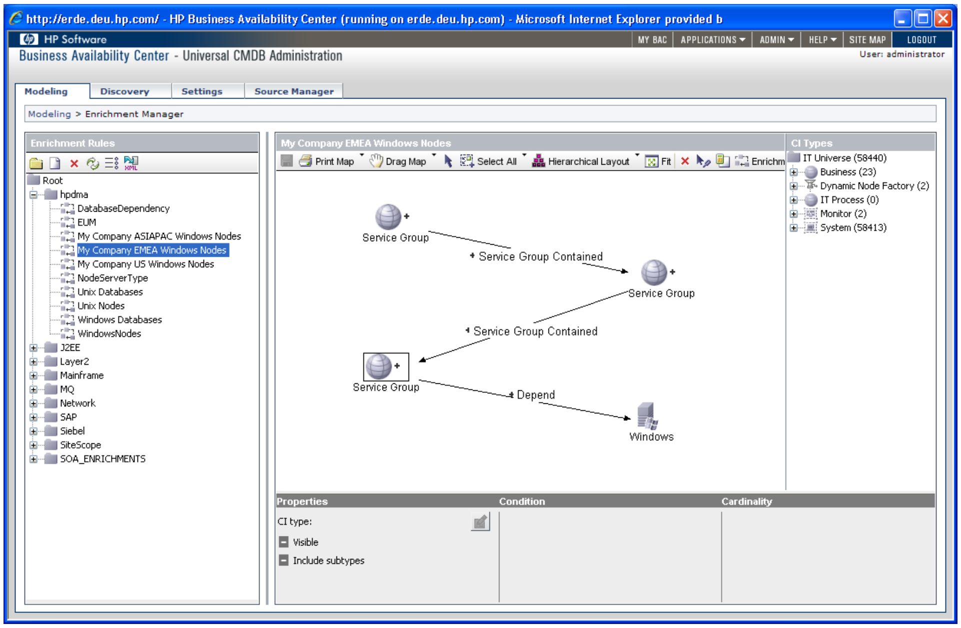
1. Open the Enrichment Manager:
   - **UCMDB 7 and 8:**

     Admin → Modeling → Enrichment Manager
   - **UCMDB 9:**

     Modeling → Enrichment Manager
   - **BAC:**

     Admin → Universal CMDB → Modeling → Enrichment Manager
2. Expand the **hpdma** group from the Enrichment Rules list.
3. Apply enrichment rules (for example, to place Windows nodes in Service groups).

Figure 4   **Example of a My Example Company Enrichment Rule**



You must execute each enrichment rule whenever you discover new instances.

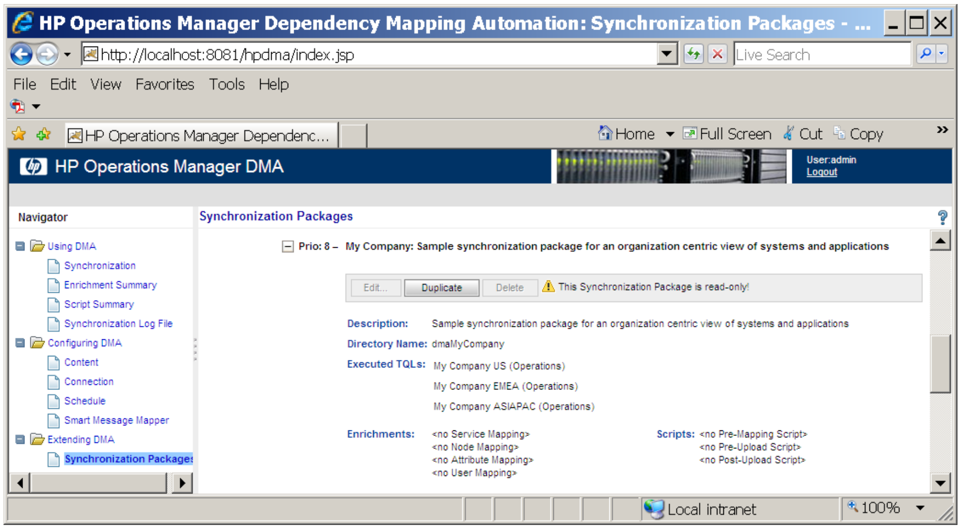
Viewing the My Company Synchronization Package

The synchronization package for the My Company synchronization package is automatically installed during HPOM DMA installation as a standard synchronization package.

To view this synchronization package, go to:

HPOM DMA Console → Extending DMA → Synchronization Packages

Figure 5 My Company synchronization package



Prio: 8 – My Company: Sample synchronization package for an organization centric view of systems and applications

This Synchronization Package is read-only!

Description:	Sample synchronization package for an organization centric view of systems and applications
Directory Name:	dmaMyCompany
Executed TQLs:	My Company US (Operations) My Company EMEA (Operations) My Company ASIAPAC (Operations)
Enrichments:	<no Service Mapping> <no Node Mapping> <no Attribute Mapping> <no User Mapping>
Scripts:	<no Pre-Mapping Script> <no Pre-Upload Script> <no Post-Upload Script>

Activating the My Company Synchronization Package

To activate the My Company Synchronization Package:

1 Open the Synchronization Packages page and check:

My Company: Sample synchronization package for an organization centric view of systems and applications

2 Click **Save**.

3 Start a new synchronization from the Synchronization page.

On completion, the My Company view is synchronized into the HPOM Service Navigator. The CI are arranged under their assigned regions, as well as under System Infrastructure and Applications.

➤ You can activate the My Company synchronization package without the other standard HPOM DMA synchronization packages.

If you do so, you might run into the following restrictions:

- **Oracle Database Packages**

  There is a specific service mapping for Oracle databases in the `dmaOracle` synchronization package. If this is inactive, the Oracle CIs are mapped to the `ucmdb_oracle` STD. This STD is not available by default.

  You must do one of the following:

  - Enable automatic STD creation
  - Create the `ucmdb_oracle` STD
  - Activate the `dmaOracle` synchronization package
  - Copy the service mapping from the `dmaOracle` synchronization package to the `dmaMyCompany` synchronization package

- **Non-Standard Packages**

  For CI types that are not covered in the standard HPOM DMA synchronization packages, the corresponding STDs are also not available by default. Either create the STD or enable automatic STD creation.

- **Inactive Packages**

  For all other mappings used in inactive synchronization packages (for example, node mapping, user mapping, and attribute mapping), the same applies as for service mapping. If you need it, you can copy it to active synchronization packages.

Web Server Example

This section explains how to design and develop a new synchronization package for IIS web servers.

Prerequisites

Make sure that you have the following systems installed, configured and running:

- HPOM management server.
- UCMDB or BAC, including deployed HPOM DMA packages for UCMDB.

  UCMDB should contain some discovery data. In particular, it should contain some hosts and IIS CIs.

Designing the Web Server Synchronization Package

To design the Web Server synchronization package and the associated TQL queries, you must first understand exactly what you want to achieve. The first task is to:

- Analyze the UCMDB data to be synchronized
- Visualize the effect required in HPOM

Analyzing the UCMDB Data to be Synchronized

The first step in creating a new HPOM DMA synchronization package is to investigate what data you want to synchronize and understand why you need that data. HPOM DMA also provides some automation features. Decide which you want to use for what.

So the most important question you should ask yourself is: *What is the expected outcome in HPOM?*

This question leads to the following detailed questions:

- Which nodes in the UCMDB do you want to monitor in HPOM and need to synchronize?
- What applications do you want to monitor with HPOM?

- How do you plan to organize your data in HPOM?
- To which node groups do you want to assign the nodes?
- Do you need a service map in Service Navigator? Which services do you need to create in Service Navigator?
- How do you want to propagate problems in Service Navigator?

These questions are answered in this chapter based on a simple development example where your organization is required to provide a web server service to internal customers. In this scenario, some host systems and web servers running on these hosts, are synchronized from the UCMDB to HPOM.

Expected Outcome in HPOM

To achieve the expected outcome in HPOM, the following steps must be completed:

- You must synchronize all hosts that are hosting your web servers and all hosts on which your web servers depend.
- You must monitor the web server applications. As a result, you must also synchronize the web server CIs.
- You want to organize your services under the application type.

  As a result, you need a node group for the web server application.
- You must create a service map that shows a simple topology map.

Developing the Web Server Synchronization Package

To develop the Web Server synchronization package and the associated TQL queries, you must:

- Create TQL queries in the UCMDB to collect the content that you want to synchronize
- Develop your Web Server synchronization package in HPOM DMA

Developing Your TQL Queries to Collect Content to Synchronize

To create TQL queries that can be used for synchronization, you create views with the View Manager in the UCMDB. Each view you create also creates a corresponding TQL query.

What you define in your view is finally displayed in Service Navigator. As a result, it is essential that you define in this view everything that you want to see in Service Navigator. In addition, you must make sure that you also have the information that later enables you to decide how to assign nodes to node groups.

Creating TQL Queries in the UCMDB

To define your UCMDB view:

1. Create a view named **`Web Server`** that can be used as a query in your synchronization package.
2. Drag the **IIS** CI type from the CI type list to your view.
3. Add the **Host** CI type.
4. Select a dependency type between the Host CI type and the IIS CI type which starts with `contain`.

   By default, this type creates a containment relationship in Service Navigator.
5. For the Host CI, enable the Host DNS Name (`host_dnsname`) attribute for exporting as follows:
   a. Right-click **Host** and select **Node Properties** from the context menu.

      The Node Properties dialog box opens.
   b. Click **Advanced layout settings**.

      The Layout Settings window opens.
   c. Select the following attributes for calculation if not selected:
      — `CI Type` (to distinguish the node type, for example, `nt`, `host`, `unix`)

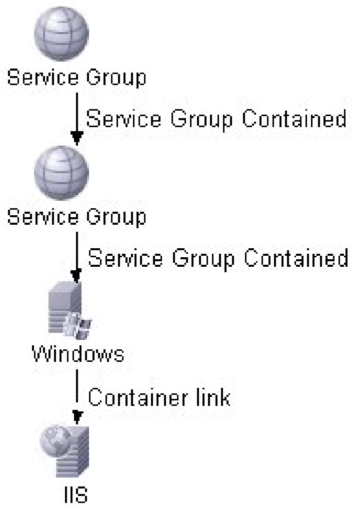
      — `Host DNS Name` (to set the `hosted_on` attribute)
6. Make sure that your Hosts and IIS CIs have a common root, so that it is easy to find your CIs in Service Navigator:

a Add a **Service Group** and link the `Host` CIs using the **Service Group Contained** relationship.

b To make sure that the Service Group is created first, execute the HPOM DMA enrichment rules. They create two Service Group for:

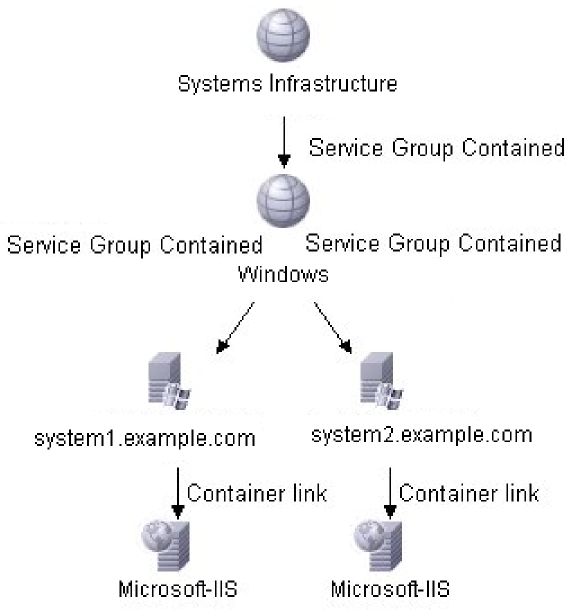
— UNIX hosts

— Windows hosts



7 Preview your newly created view. Verify that it contains what you specified.

Figure 6 Preview of the UCMDB Web Server View



8 When the view is correctly specified, save your view.

Developing Your Synchronization Package

You have defined what you want to synchronize. The next step is to define in HPOM DMA where to find it and what to do with it. To do this, you must create a new synchronization package for your IIS model. HPOM DMA ships with a default package that is always deployed. It handles some essential steps, including:

- Assigning the `hosted_on` attribute for Smart Message Mapping
- Providing a useful default for labeling

You can take advantage of this standard functionality and concentrate on the specific components that you need for your package.

Synchronization packages are containers that can have multiple TQL queries, mapping rules and scripts. Each synchronization package has a priority that determines in which sequence the mapping rules and scripts are executed. The synchronization package concept supports easy extensibility.

A synchronization package can contain the following files:

- `bundle.xml` (essential)
- `attributemapping.xml`
- `servicemapping.xml`
- `nodemapping.xml`
- `usermapping.xml`
- `premapping.groovy`
- `preupload.groovy`
- `postupload.groovy`

The `bundle.xml` file defines the package and is the only file essential to the synchronization package. The file contains a name, a description, a priority and, most importantly, one or more TQLs that define which views are used to read the UCMDB data.

All mapping files are used to enrich the data with HPOM relevant information:

- The attribute mapping file contains rules that enable you to modify the content of a CI. You can define new attributes or you can modify existing ones. Because all mappings required by the example are already available in the default package, an additional `attributemapping.xml` file is not required.
- The service mapping file contains rules that define how a CI type is mapped to a service type definition (STD) in Service Navigator. Service type definitions (STDs) are used to identify how CIs from the UCMDB are to be handled when building services in HPOM on synchronization. For further information, see Chapter 4, Service Type Definition Files.
- The node mapping file contains rules to specify to which node groups a node is assigned.
- The user mapping file enables you to set up rules defining how services are assigned to user profiles. So you can automatically set access permissions to your services.

The script files enable you to execute your own scripts for every synchronization:

- The `premapping.groovy` file is executed before the mapping rules are executed.

- The `preupload.groovy` file is executed when HPOM DMA has retrieved all CIs from UCMDB but before they are uploaded to HPOM.
- The `postupload.groovy` file is executed as a last step.

These files enable you to modify the CIs before mapping, before upload, and after upload.

Creating the Web Server Synchronization Package

To create a synchronization package:

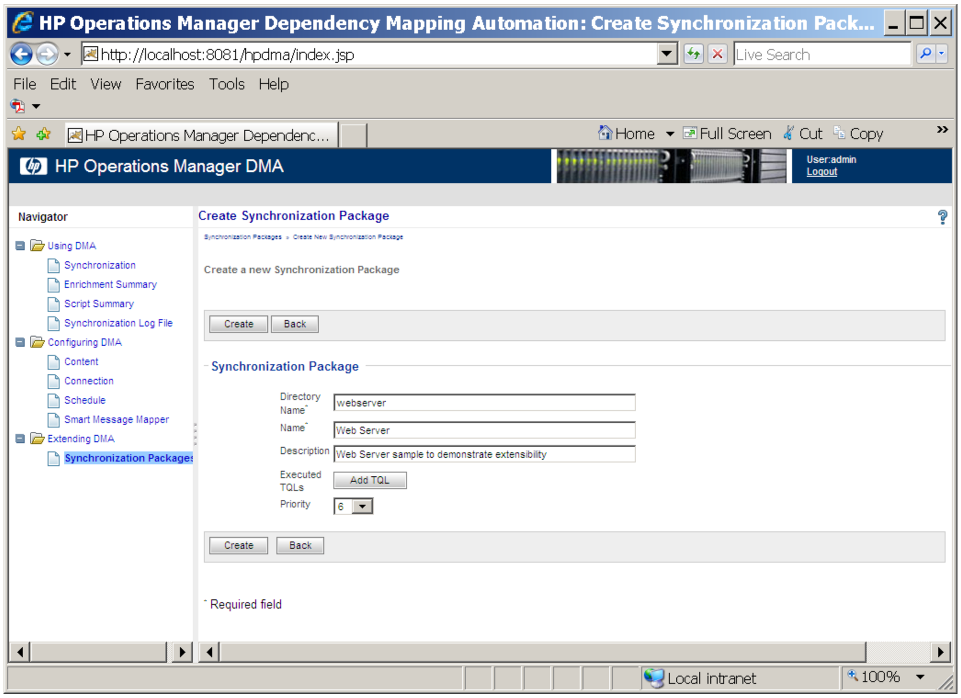
1. Go to Synchronization Packages in the HPOM DMA Console:

   HPOM DMA Console → **Extending DMA** → **Synchronization Packages**

2. Click **Create new**.
3. Enter the following information:

• Directory name:	**`webserver`**
• Name:	**`Web Server`**
• Description:	**`Web Server sample to demonstrate extensibility`**
• TQL:	**`Web Server`**
• Priority:	**6**

Figure 7 The Web Server Synchronization Package



Create Synchronization Package

Synchronization Packages » Create New Synchronization Package

Create a new Synchronization Package

Synchronization Package

Directory Name*	webserver
Name*	Web Server
Description	Web Server sample to demonstrate extensibility
Executed TQLs	Add TQL
Priority	6

* Required field

4 Click **Create**.

This creates a new directory in the synchronization package folder called `webserver` and adds the `bundle.xml` for this synchronization package.

You have created a synchronization package named `Web Server` that is querying the `Web Server TQL` with priority `6`. The priority defines in which sequence the synchronization packages is executed. Priority 6 is comparatively low. Synchronization packages with a higher priority, for example, 3 may overwrite your results. The default package has the lowest priority: 10.

Creating Service Mapping

Service mappings define the relationship between the type of a CI in the UCMDB and the service type definition (STD) of a service in HPOM.

In your view, you have defined three types of CIs:

- Hosts (`nt` or `unix`)
- IIS (`iis`)
- Service Groups (`servicegroups`)

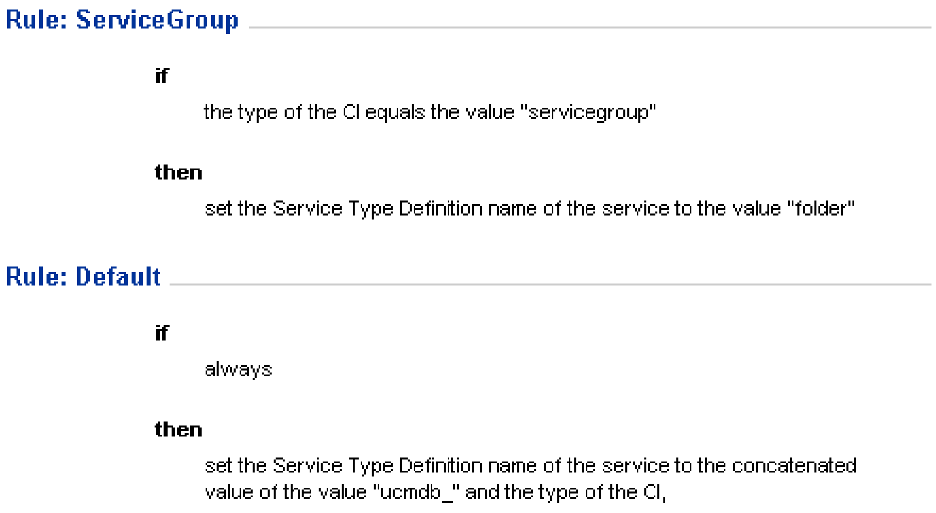
To view the active service mappings, go to:

HPOM DMA Console → Using DMA → Enrichment Summary → Service Mapping

The default mapping rules are active:

- Mapping Service groups to `folder` STD
- Mapping all other CI types to `<ucmdb>_` CIType

Figure 8 Web Server Synchronization Package



Rule: ServiceGroup

if

the type of the CI equals the value "servicegroup"

then

set the Service Type Definition name of the service to the value "folder"

Rule: Default

if

always

then

set the Service Type Definition name of the service to the concatenated value of the value "ucmdb_" and the type of the CI, This means that, for example, all hosts of type `nt` are assigned to the STD `ucmdb_nt`, and all web servers of type `iis` are assigned to STD `ucmdb_iis`. If this does not meet your needs, you must add your own service mapping with a higher priority.

➤ The UCMDB distinguishes between a CI type label and a CI type name. In the mapping files, you must always refer to the type name. You can look up the type name in the CI type manager of the UCMDB by selecting the properties of a type.

In this example, it is assumed that you want to map to the STD `UCMDB_IIS` instead of the default STD `ucmdb_iis`. You must add this mapping as follows:

1 Go to:

   HPOM DMA Console → **Extending DMA** → **Synchronization Packages**

2 Open the **Web Server** synchronization package.

3 Click **Create service mapping**.

4 Enter the following content:

```
<?xml version="1.0" encoding="utf-8"?>
<Mapping>
    <Rules>
        <Rule name="IIS">
            <Condition>
                <Equals ignoreCase="true">
                    <CiType/>
                    <Value>iis</Value>
                </Equals>
            </Condition>
            <MapTo>
                <STD>
                    <Value>UCMDB_IIS</Value>
                </STD>
            </MapTo>
        </Rule>
    </Rules>
</Mapping>
```

The IIS service mapping rule includes two parts:

- The `<Condition>` section defines when the rule is applied. In the example, the rule is applied whenever a CI of type `iis` is found.
- The `<MapTo>` section defines the action. In the example, the rule specifies that the resulting service has an STD called `UCMDB_IIS`.

HPOM DMA includes STDs for the out-of-the-box synchronization packages. It is essential that the STDs used in the service mapping exist in HPOM. If they do not exist, the synchronization fails. HPOM DMA can automatically create STDs, if you have enabled automatic STD creation. However, if you require a custom STD, you can create one.

> If you create your own STDs and you use the UI launch feature, then you must assign the `BAC Service UI Launch` or `UCMDB Service UI Launch` tool groups to those STDs.
>
> This is done in the HPOM for Windows console under **Tools** → **Configure** → **Service Types**.
>
> In HPOM for UNIX or Linux, service type definitions are specified in service type definition files, which must be in XML format and must be of form `DOCTYPE Services`. For information on the documentation type definitions and services, see the *Administrator's Reference*.

Creating Node Mapping

HPOM node groups are used to organize your nodes and to automate the deployment of the monitoring environment of nodes that are added to a node group.

Assuming that you have the Web Server SPI installed on your HPOM server and assign all nodes hosting IIS in your UCMDB to the node group `IIS`, you can configure HPOM DMA to trigger the HPOM deployment mechanism when HPOM DMA synchronizes the nodes. HPOM automatically deploys the Web Server SPI onto your HPOM managed node, when the node is added to the IIS node group.

You must define the node mapping to assign all nodes hosting IIS to the IIS node group.

To define the node mapping, complete the following steps:

1. Go to:

   HPOM DMA Console → **Extending DMA** → **Synchronization Packages**

2. Open the **Web Server** synchronization package.
3. Click **Create node mapping**.
4. Enter the following content:

```
<?xml version="1.0" encoding="utf-8"?>
<Mapping>
    <Rules>
        <Rule name="IIS Node Group">
            <Condition>
                <And>
                    <Equals>
                        <CiType/>
                        <Value>nt</Value>
                    </Equals>
                    <Exists>
                        <XPathResult>descendant::
                         ci[type='iis']</XPathResult>
                    </Exists>
                </And>
            </Condition>
            <MapTo>
                <NodeGroup>
                    <Value>IIS</Value>
                </NodeGroup>
            </MapTo>
        </Rule>
    </Rules>
</Mapping>
```

5 Click **Save**.

The node mapping includes a rule that puts every host of type `nt` that contains an `iis` CI into the IIS node group. An XPathResult expression is used to detect these nodes. The XPath query checks through the children for CIs of type `iis`.

When you synchronize, HPOM DMA can automatically create the corresponding node groups, if you have enabled automatic node group creation.

Testing and Fine-Tuning

After you create the TQL query and synchronization package for the first simple example, you should do a test run.

Preparing for Testing

You need to do some configuration changes in the HPOM web service:

- Enable the automatic service type definition creation
- Enable the automatic node group creation
- Activate the Web Server synchronization package

Enabling Automatic Service Type Definition and Node Group Creation

To enable automatic service type definition and automatic node group creation:

1. Open a console.
2. Enter the following commands:

```
ovconfchg -ovrg server -ns
opc.WebService.ConfigurationItem -set StdCreationEnabled
true
```

```
ovconfchg -ovrg server -ns
opc.WebService.ConfigurationItem -set
NodeGroupCreationEnabled true
```

```
ovc -restart ovtomcatB
```

Activating the Web Server Synchronization Package

To activate your synchronization package:

1. Go to:

   HPOM DMA Console → Configuring DMA → Content

2. *Optional.* To avoid possible side effects, deactivate all the other synchronization packages.

The default synchronization package is always executed during synchronization.

3 Review the enrichment summary under:

HPOM DMA Console → Using DMA → Enrichment Summary

Step through the links and check the specified configurations.

Running a Test Synchronization

To execute the test synchronization:

1 Go to:

HPOM DMA Console → Using DMA → Synchronization

2 Click **Start Synchronization**.

HPOM DMA informs you that synchronization has been started.

3 Click **log file** to switch to the log file view.

4 Follow the synchronization steps.

If everything is working correctly, you get a `Synchronization successfully completed on <date and time>` message.

In HPOM, there is a new node group containing your IIS hosts. In Service Navigator, there should be a service map with all the CIs you have selected in your UCMDB view.

Iteratively Improving Your Package

You can continue to modify your TQL queries and add additional CIs and dependencies. Follow the above steps to add node and service mappings.

You can continue to further improve your model. For your first extension, you could, for example, add Apache web servers to your existing views.

Activating Your Synchronization Package

When you have finished your development work, you must prepare your synchronization package for rollout. For this, you must create two deployment packages that you can deploy to your production servers, one each for:

- UCMDB or BAC
- HPOM DMA

UCMDB Package

To create a UCMDB/BAC package:

1. From BAC or the UCMDB, open the Package Manager.
   - **UCMDB 7 and 8:**

     Admin → **Settings** → **Package Manager**
   - **UCMDB 9:**

     Administration → **Package Manager**
   - **BAC:**

     Admin → **Universal CMDB** → **Settings** → **Package Manager**
2. Click **New** button. The Create Custom Package wizard opens.
3. Enter a name and description for the new package and click **Next**..
4. Select your Web Server view and save your package.

   You can copy and deploy this package to any other UCMDB.

HPOM DMA Package

To create and deploy the HPOM DMA package:

1. Compress the `webserver` directory in the synchronization package folder to zip file, for example, `webserver.zip`:

   `<`*`SharedDir`*`>/server/conf/dma/sync-packages/webserver/`

2 To deploy the HPOM DMA package, extract your package file into the package directory of your production HPOM DMA installation.

➤ This package does not contain any configuration for automatic STD and node group creation. This is a server configuration It is not part of your package.

Advanced Topics

The Web Server Example on page 26 illustrates how to extend the synchronization package. This section covers some advanced topics.

Adding a New Relationship Between CIs to Correctly Model Your Service Map

Looking at the result of your service map, you can see that the IIS servers propagate their status up to the hosts. But the reality is actually the other way around. When the host has a problem (for example, if the disk is full), the IIS server application does not perform.

The UCMDB has not modeled the correct impact for you and you must do this yourself. To do so you have to:

- Add, for example, the `Depend` relationship between the host CI Type and the web server CI Type.
- Create an enrichment rule that adds this `Depend` relationship between all hosts and web servers that have a contained relationship.

Adding the Depend Relationship

To add the Depend relationship between the host CI Type and the web server CI Type:

1 Go to the CI Type Manager:

- **UCMDB 7 and 8:**

  Admin → Modeling → CI Type Manager

- **UCMDB 9:**

  Modeling → CI Type Manager

- **BAC:**

  Admin → Universal CMDB → Modeling → CI Type Manager

2 In the navigation pane open:

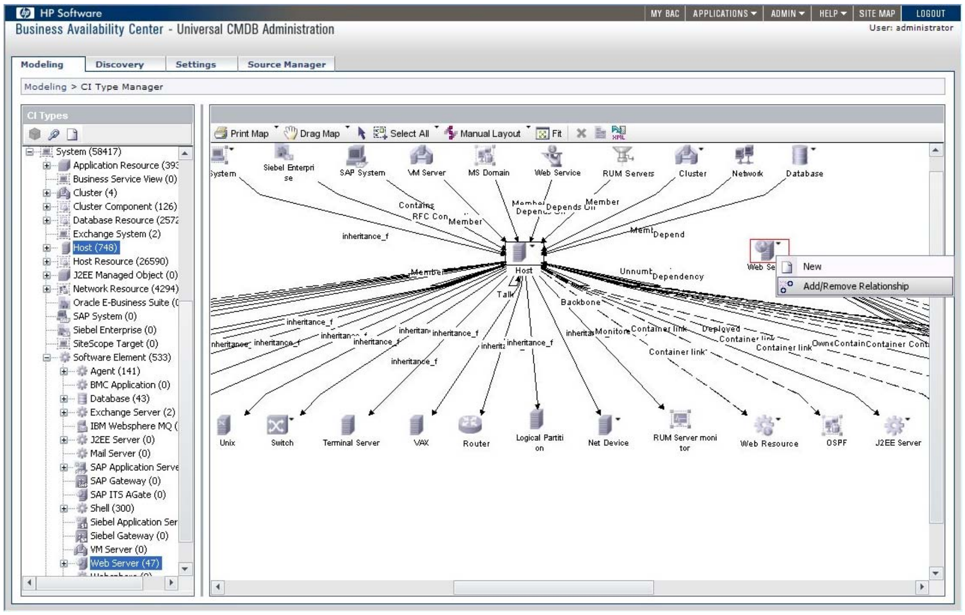
IT Universe → System → Software Element

3 Select **Web Server**.

4 Also in the navigation pane, select the **Host** (with **CTRL** + click).

A combined view opens in the content pane.

Figure 9 Add Depend Relationship

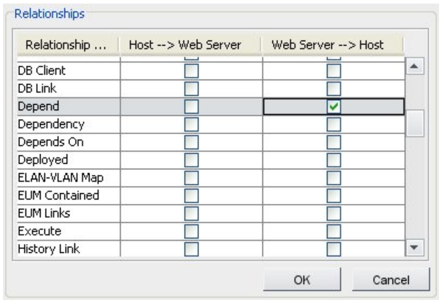


5 Select the `Host` and `Web Server` CI types in the content pane.

6 From the context menu for the `Web Server` CI types, select **Add/ Remove Relationship**.

The Add Depend Relationship dialog box opens, as shown in Figure 10.

Figure 10 Add Depend Relationship Dialog Box



Relationship ...	Host --> Web Server	Web Server --> Host
DB Client	☐	☐
DB Link	☐	☐
Depend	☐	☑
Dependency	☐	☐
Depends On	☐	☐
Deployed	☐	☐
ELAN-VLAN Map	☐	☐
EUM Contained	☐	☐
EUM Links	☐	☐
Execute	☐	☐
History Link	☐	☐

7 For the `Depend` relationship, select the check box for **Web Server** → **Host** and click **OK**.

The Depend relationship between the host CI Type and the web server CI Type is established.

Creating an Enrichment Rule

An enrichment rule consists of two parts:

- **TQL** The TQL is the condition part and you need all `IIS` and `Host` CI Types.
- **Enrichment** The Enrichment part is the action part of the rule and you must add the `Depend` relationship.

To create an enrichment rule:

1 Open the Enrichment Manager:

- **UCMDB 7 and 8:**

  Admin → **Modeling** → **Enrichment Manager**

- **UCMDB 9:**

  Modeling → **Enrichment Manager**

- **BAC:**

  Admin → **Universal CMDB** → **Modeling** → **Enrichment Manager**

2 Click **New** or **CTRL + N**.

The Create New Enrichment Rules dialog box opens.

3 Enter the following information:

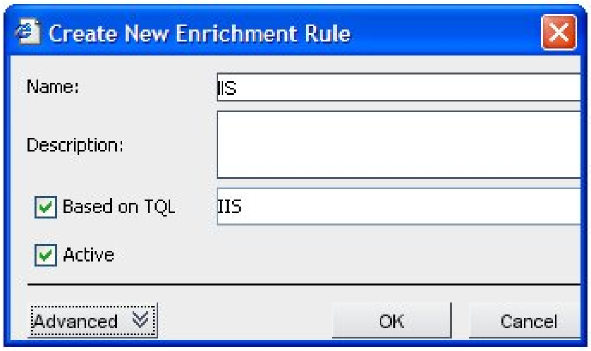
   a Name: `IIS`

   b Based on TQL: `IIS`

   c Check the **Active** check box.

   Figure 11 on page 45 illustrates the Create New Enrichment Rules dialog box entries.

   Figure 11 Create New Enrichment Rule Dialog Box



Create New Enrichment Rule

Name: IIS

Description:

[✓] Based on TQL IIS

[✓] Active

4 Add the TQL (condition) part:

   a Drag the `IIS` and `Host` CI Types from the type selection pane into the view.

   b Add a containment relationship between both CIs.

5 Add the Enrichment part (action) part of the rule. Here you must add the `Depend` relationship.

   a Select the IIS and host CI Type and in the context menu select **Add Relationship**. Direction should be from IIS to host.

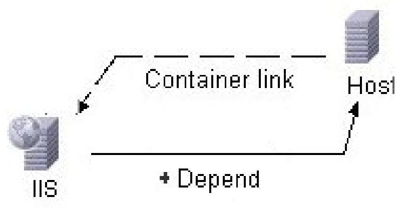
   b Browse to the Depend relationship:

   IT World Links → **System Links** → **Parent** → **Depend**

   c Click **OK**.

   The enrichment rule is shown in Figure 12.

Figure 12 Enrichment Rule



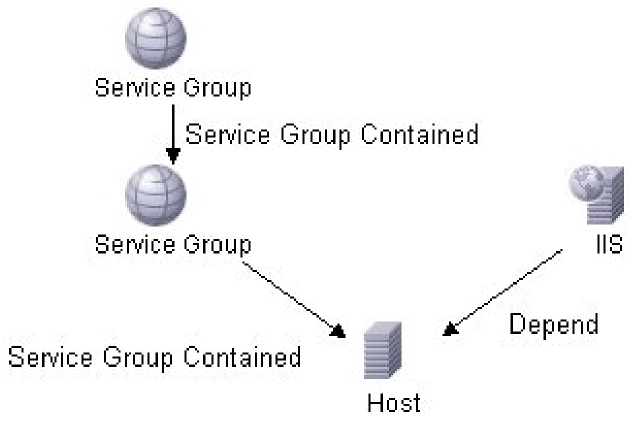
d Save your enrichment rule.

6 Execute the enrichment.

Changing your TQL Query to Reflect the Change

Remove the containment relationship from Host to IIS, and add a depend relationship from IIS to host.

Figure 13 TQL with depend relationship



With this change, the IIS is not contained any longer under host. Although this is the desired behavior, it also makes all IIS web servers into root services under the CMDB root service in your service map.

To display IIS web servers under a more appropriate service tree structure, create the required new service groups.

If you want to add your IIS web servers under:

`Applications` → `Web Servers` → `IIS`

1 Before changing your enrichment rule, remove the old enrichment results.

2 Go back to the Enrichment Manager:

- **UCMDB 7 and 8:**

  Admin → **Modeling** → **Enrichment Manager**

- **UCMDB 9:**

  Modeling → **Enrichment Manager**

- **BAC:**

  Admin → **Universal CMDB** → **Modeling** → **Enrichment Manager**

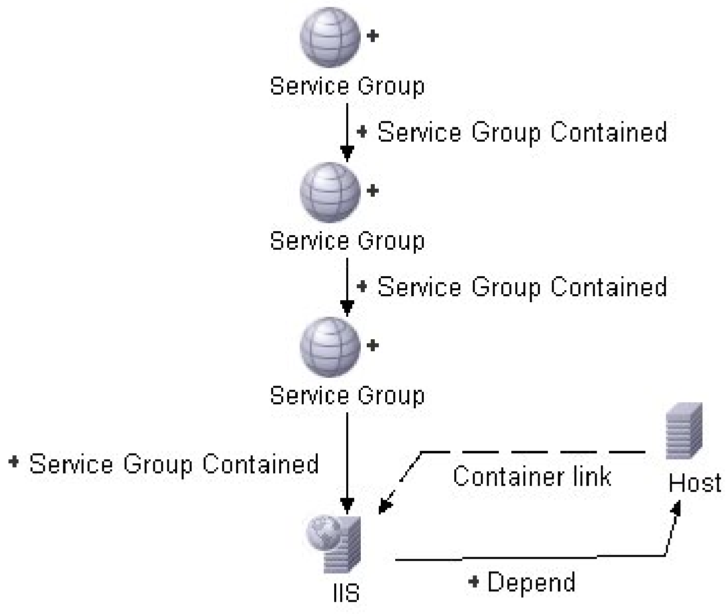
3 Change the Enrichment part of your rule:

a Add the three service groups: **`Applications`**, **`Web Servers`**, and **`IIS`**.

b Link the three service groups with the **Service Group Contained** relationships, as shown in Figure 14.

c In the Node Definition, set the name as selected for the service groups (**`IIS`**, **`Web Servers`**, **`Applications`**).

Figure 14 **Enrichment Rules With Service Groups**

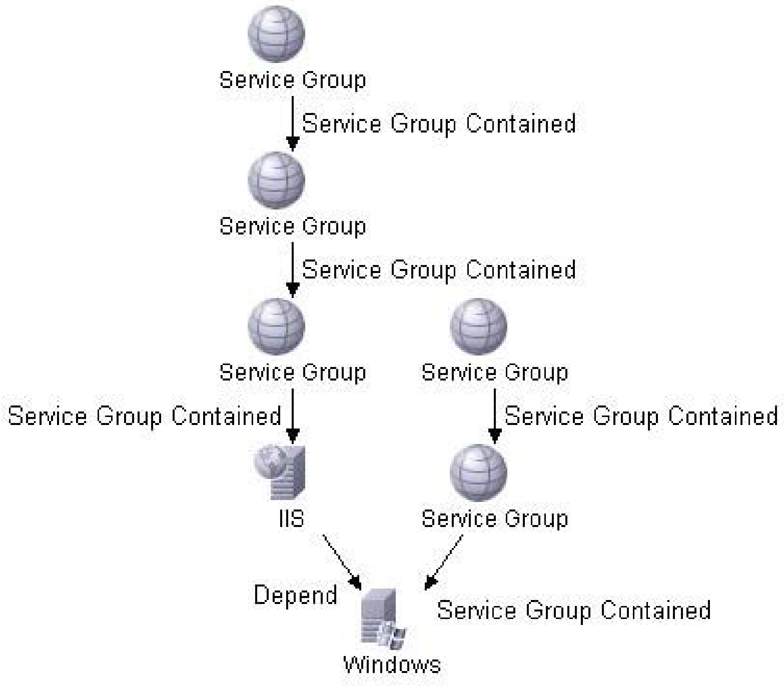

Service Group
+ Service Group Contained
Service Group
+ Service Group Contained
Service Group
+ Service Group Contained
Container link
Host
IIS
+ Depend

d Save your enrichment rule.

4 Execute the enrichment.

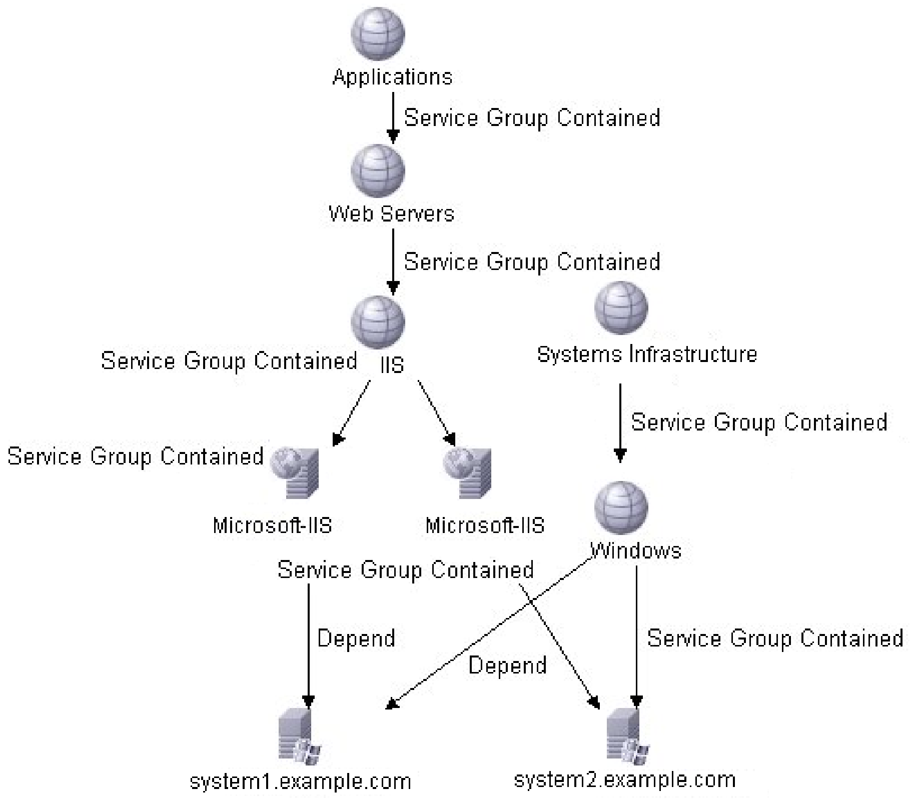
5 Go back to the View Manager to change your view. Add the service groups and link them with the appropriate relationship.

Figure 15 Web Server View with Depend Relationship



6 Selecting preview should display the following:

Figure 16 Preview



7 Save your view.

You have changed your TQL query to reflect the enrichment rules change. You can now test and fine-tune this example.

Combining Multiple Synchronization Packages

Until now, you have synchronized with your synchronization package only. It is now time to consider other synchronization packages, for example, those supplied with HPOM DMA, which you deactivated earlier. Now you want to have your hosts and host resources. You also want your databases in the service map. How can you integrate your synchronization package with the existing data or even add further extensions?

In HPOM DMA, you can select multiple synchronization packages. Each synchronization package can have multiple TQL queries associated with it. All the data retrieved from the UCMDB, based on the TQL queries, are merged by HPOM DMA and uploaded as a single view into Service Navigator. HPOM DMA helps to make this as easy as possible. However, there are some points that you should keep in mind:

- A CI can be synchronized only once. Even though it can be selected in multiple views, it occurs in Service Navigator just once. HPOM DMA automatically merges multiple occurrences of a CI.
- A CI can have only one containment parent. Make sure that there are not multiple containers for a CI in different views. If HPOM DMA finds multiple containment parents for a CI, you are notified with a warning.
- There may be an unlimited number of dependencies for every CI. However, you must make sure that there are no cycles in the dependency graph. Cycles are not supported in Service Navigator. Review your view to make sure that you have not introduced a cycle when HPOM DMA merges multiple views.

For your Web Server example, these issues have been avoided. Go to the HPOM DMA console and activate additional synchronization packages. Then continue with testing and tuning.

External Services

When synchronizing, all services are added under the CMDB root service. If you have your own top-level service structure for web servers, you might want to add links into the CMDB service view. You can do this within service mapping. See Mapping UCMDB Information to Services on page 87 for details.

Defining STDs

This example assumes that new STDs are automatically created. If you want to create your own STDs, see Chapter 4, Service Type Definition Files for details.

Scripting

The enrichment of HPOM DMA is a powerful tool that should meet most of your extension needs. In addition, you can customize your synchronization with scripts. There are the following types:

- Pre-mapping to set CI attributes before mapping.
- Pre-upload to tune your synchronization results before they get uploaded to HPOM.
- Post-upload to trigger additional actions you need when the synchronization is done.

See Chapter 12, Scripting for details.

3 TQL Queries

TQL queries define the node and service configuration items (CI) in the UCMDB to be synchronized with HPOM. All TQL queries that are referenced in the `bundle.xml` file must exist in the UCMDB. You can either use existing TQL queries or write specific TQL queries that match the monitoring infrastructure.

Selecting TQL Attributes

When specifying the necessary TQL queries, make sure that the following attributes are synchronized for each UCMDB CI type:

- `display_label`
- `CiType`

You can select the attributes that are exposed by the UCMDB web service in the View Manager:

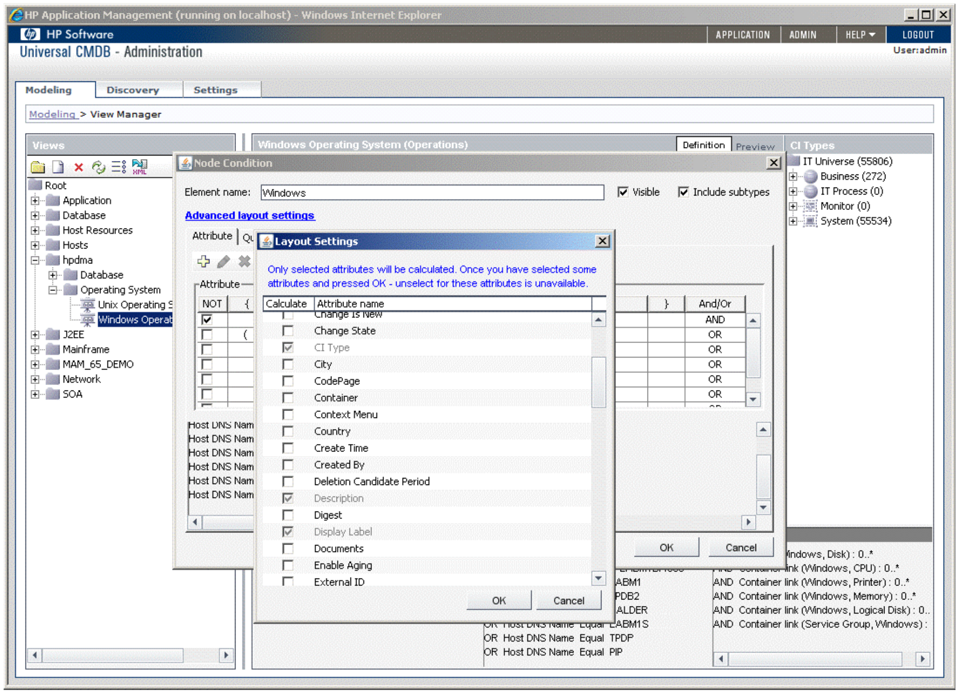
Right-click a node in the TQL definition tree.

From the context menu, select **Node Properties** → **Advanced layout settings** and select the appropriate attributes.

Typically, for each CI type in the UCMDB, one or more attributes are selected to form the key attributes for Smart Message Mapping. For a list of required and recommended attributes, see UCMDB Attributes on page 143.

Make sure that all key attributes that you choose are exposed by the UCMDB web service.

Figure 17 Selecting Attributes Exposed by the UCMDB Web Service



Element name: Windows ☑ Visible ☑ Include subtypes

Only selected attributes will be calculated. Once you have selected some attributes and pressed OK - unselect for these attributes is unavailable.

Calculate	Attribute name
☐	Change Is New
☐	Change State
☑	CI Type
☐	City
☐	CodePage
☐	Container
☐	Context Menu
☐	Country
☐	Create Time
☐	Created By
☐	Deletion Candidate Period
☑	Description
☐	Digest
☑	Display Label
☐	Documents
☐	Enable Aging
☐	External ID

NOT	{	}	And/Or
☑			AND
☐	(		OR
☐			OR
☐			OR
☐			OR
☐			OR

```
Host DNS Nam
Host DNS Nam
Host DNS Nam
Host DNS Nam
Host DNS Nam
Host DNS Nam
```

```
ABM1
PDB2
ALDER
OR  Host DNS Name  Equal  LABM1S
OR  Host DNS Name  Equal  TPDP
OR  Host DNS Name  Equal  PIP
```

```
indows, Disk) : 0..*
ink (Windows, CPU) : 0..*
AND  Container link (Windows, Printer) : 0..*
AND  Container link (Windows, Memory) : 0..*
AND  Container link (Windows, Logical Disk) : 0..
AND  Container link (Service Group, Windows) :
```

Setting Up a TQL Query in a UCMDB Development Environment

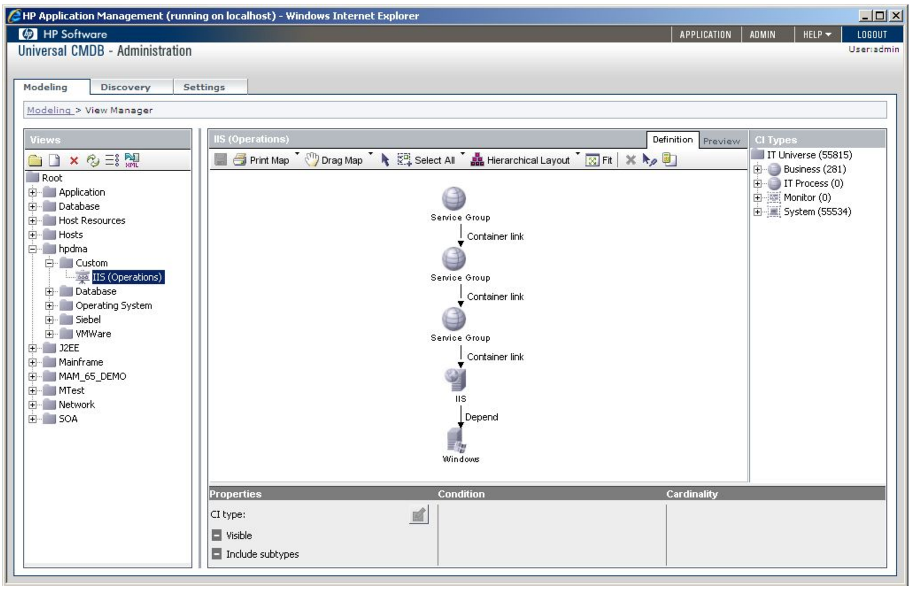
To set up a TQL query in a UCMDB development environment:

1 Create a view in the View Manager. Activate essential attributes, such as CI Type and the ID to be able to launch tools. Attributes are selected to form the key attributes for Smart Message Mapping. For a list of required and recommended attributes, see UCMDB Attributes on page 143.

> If an object is found by more than one TQL query, it is merged to a single object in HPOM, and includes all attributes and all links defined in all of the queries. You can always split a topology into multiple queries, if this is easier to define. This situation also occurs if the TQL queries are defined in different bundles and all of them are activated during synchronization.

2 Make sure the created View has the desired CI result map. Figure 18 is an example of a view.

Figure 18 Example of a View



3. Use the Preview tab on the top of the view area to preview the result map selected by the view TQL.
4. Open the Package Manager:
   - **UCMDB 7 and 8:**

     Admin → **Settings** → **Package Manager**
   - **UCMDB 9:**

     Administration → **Package Manager**
   - **BAC:**

     Admin → **Universal CMDB** → **Settings** → **Package Manager**
5. Create a package in the Package Manager.
   a. Click **New**.

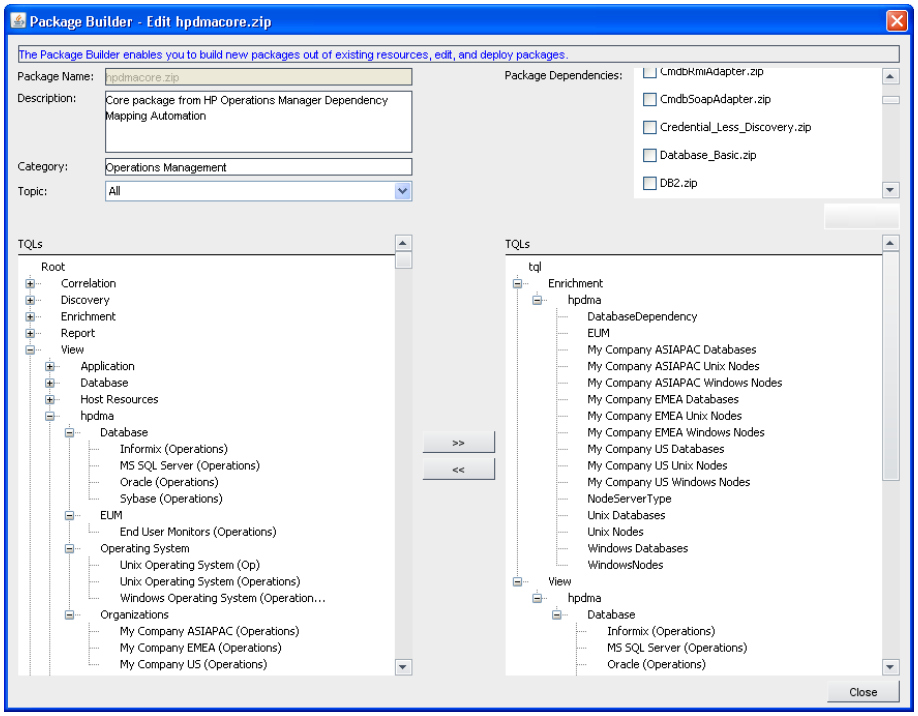
      The Create Custom Package wizard opens.
   b. Enter a name and description for the new package and click **Next**.
   c. Expand **Query** → **TQLs** → **Enrichment** and check *<Package Name>*, where *<Package Name>* is the name of the package containing the TQL query that you want to move.
   d. Expand **Query** → **TQLs** → **View** and check *<Package Name>*.
   e. Expand **Query** → **Views** and check *<Package Name>*
   f. Expand **Query** → **Enrichments** and check *<Package Name>*.
   g. Click **Next**.

      A summary of the selections that you have made is displayed.
   h. Click **Finish**.

Setting Up a TQL Query in a UCMDB Production Environment

Import and deploy package in Package Manager. For an example, see Figure 19.

Figure 19 Importing and Deploying Packages



Package Builder - Edit hpdmacore.zip

The Package Builder enables you to build new packages out of existing resources, edit, and deploy packages.

Package Name: hpdmacore.zip

Description: Core package from HP Operations Manager Dependency Mapping Automation

Category: Operations Management

Topic: All

Package Dependencies:
- [ ] CmdbRmiAdapter.zip
- [ ] CmdbSoapAdapter.zip
- [ ] Credential_Less_Discovery.zip
- [ ] Database_Basic.zip
- [ ] DB2.zip

TQLs

```
Root
  Correlation
  Discovery
  Enrichment
  Report
  View
    Application
    Database
    Host Resources
    hpdma
      Database
        Informix (Operations)
        MS SQL Server (Operations)
        Oracle (Operations)
        Sybase (Operations)
      EUM
        End User Monitors (Operations)
      Operating System
        Unix Operating System (Op)
        Unix Operating System (Operations)
        Windows Operating System (Operation...
      Organizations
        My Company ASIAPAC (Operations)
        My Company EMEA (Operations)
        My Company US (Operations)
```

TQLs

```
tql
  Enrichment
    hpdma
      DatabaseDependency
      EUM
      My Company ASIAPAC Databases
      My Company ASIAPAC Unix Nodes
      My Company ASIAPAC Windows Nodes
      My Company EMEA Databases
      My Company EMEA Unix Nodes
      My Company EMEA Windows Nodes
      My Company US Databases
      My Company US Unix Nodes
      My Company US Windows Nodes
      NodeServerType
      Unix Databases
      Unix Nodes
      Windows Databases
      WindowsNodes
  View
    hpdma
      Database
        Informix (Operations)
        MS SQL Server (Operations)
        Oracle (Operations)
```

4 Service Type Definition Files

Service type definitions (STDs) are used to identify how CIs from the UCMDB are to be handled when building services in HPOM on synchronization. They are a concept in HPOM for Windows to assign tools, propagation rules, and calculation rules to services. HPOM DMA uses this concept to identify how CIs from the UCMDB are handled when constructing services in HPOM. It can also be applied to HPOM for UNIX or Linux.

Service type definition files in Windows use the MOF format. On UNIX and Linux, they are XML files.

STDs in HPOM for Windows

A service type definition file for HPOM for Windows contains four parts:

- **Propagation Rules**

  The status of most services in the service hierarchy are calculated from the messages associated with the service and from the sub-services on which the service is dependent. Status propagation refers to how a service represents its status to its superordinate services.

  The propagation rules section of an STD specifies whether the HPOM propagation rules are used or whether they are modified. A value of zero leaves the HPOM propagation rule unmodified. A positive value increases the severity.

  This is the section titled `instance of OV_PropagationRule` in Example of STDs on Windows on page 60.

- **Calculation Rules**

  The calculation rule lets you indicate the threshold value that is used to determine the status of a service. You can choose either most critical, single threshold, or multiple threshold. All three rules have the same functional principle: the highest-level severity with a rating that crosses a threshold that you specify is the severity of the service.

  The calculation rules section of an STD specifies whether the HPOM calculation rules are used or whether they are modified. A value of zero leaves the HPOM calculation rule unmodified. A positive value increases the severity.

  This is the section titled `instance of OV_CalculationRule` in Example of STDs on Windows on page 60.

- **Service Type Definitions**

  The service type definition specifies which calculation and propagation rules are applied, and which icon is to be used to represent a CI from the UCMDB on the HPOM service tree. The subsequent synchronization updates the service instances in HPOM in accordance with the changes made to the service type definition.

  This is the section titled `instance of OV_ServiceTypeDefinition` in Example of STDs on Windows on page 60.

  Service type definitions file for Windows are specified using the Managed Object Format (MOF).

- **Service Type Component**

  The service type component specifies the parent-child association and the propagation rule to be used for a matching UCMDB CI type.

  This is the section titled `instance of OV_ServiceTypeComponent` in Example of STDs on Windows on page 60.

Example of STDs on Windows

This example illustrates the common parts of the default service type definition file:

```
#pragma namespace("\\\\.\\Root\\HewlettPackard\\OpenView\\Data")

instance of OV_PropagationRule
{
    Caption = "Generic UCMDB Service PR";
    CriticalRule = 0;
    DefaultRule = 0;
    Description = "Generic UCMDB Service propagation.";
    MajorRule = 0;
    MinorRule = 0;
    NormalRule = 0;
    SettingID = "ucmdb_generic_PR";
    WarningRule = 0;
};

instance of OV_CalculationRule
{
    CalculationThresholdType = 0;
    CalculationType = 1;
    Caption = "Generic UCMDB Service CR";
    CriticalSetTo = 0;
    CriticalThreshold = 0;
    Description = "Generic UCMDB Service calculation.";
    MajorSetTo = 0;
    MajorThreshold = 0;
    MinorSetTo = 0;
    MinorThreshold = 0;
    NormalSetTo = 0;
    SettingID = "ucmdb_generic_CR";
    SingleSetTo = 0;
    SingleThreshold = 0;
    WarningSetTo = 0;
    WarningThreshold = 0;

};

instance of OV_ServiceTypeDefinition
{
    CalcRuleId = "ucmdb_generic_CR";
    Caption = "Unix";
    CaptionFormat = "folder";
    Description = "Mapping of corresponding UCMDB type";
    GUID = "ucmdb_unix";
    Icon = "ServiceGrp.ico";
    KeyFormat = "folder";
    MsgPropRuleId = "ucmdb_generic_PR";
};

instance of OV_ServiceTypeComponent
{
    GroupComponent = "OV_ServiceTypeDefinition.GUID=\"folder\"";
    PartComponent = "OV_ServiceTypeDefinition.GUID=\"ucmdb_unix\"";
    PropRuleId = "ucmdb_generic_PR";
};
```

For further information on STDs in HPOM for Windows, see the HPOM for Windows Online Help.

Uploading STDs into HPOM for Windows

The command-line utility for MOF file compilation with WMI is `mofcomp`. This utility can be used to deposit CIM information into the WMI repository or to do a simple syntax check of the MOF file. Windows NT or later operating systems include the `mofcomp` utility. You can execute the utility by entering **`mofcomp`** at the DOS prompt.

To compile a MOF file into the WMI repository, use the command:

```
mofcomp default.mof
```

Make sure that a `#pragma namespace...` statement is included at the start of the `<`*`service type definition`*`>.mof` file.

Example:

```
#pragma namespace("\\\\.\\Root\\HewlettPackard\\OpenView\\Data")
```

If you create your own STDs and you use the UI launch feature, then you must assign the `BAC Service UI Launch` or `UCMDB Service UI Launch` tool groups to those STDs.

This is done in the HPOM for Windows console under **Tools** → **Configure** → **Service Types**.

STDs in HPOM for UNIX or Linux

Propagation and Calculation rules referenced in a service type definition file must exist in the HPOM for UNIX or Linux service engine repository. You can create or update them using the `opcservice` tool. For information on the `opcservice` tool, see the *Administrator's Reference*.

A service type definition file for HPOM for UNIX or Linux contains two parts:

- **Service**

  The service specifies which calculation and propagation rules are applied, and which icon is to be used to represent a CI from the UCMDB on the HPOM service tree. Changing and uploading a service type definition is the main way to adapt the service hierarchy. The subsequent synchronization updates the service instances in HPOM in accordance with the changes made to the service type definition.

Service type definitions are specified using XML files.

The service type definition section is contained within the `<Services>` section. Each service type definition is specified within a section titled `<Service>`:

```
<Services>
  <Service>
    <Name>ucmdb_oracle_nt</Name>
    <Label>ucmdb_oracle_nt</Label>
    <Icon>database.32.gif</Icon>
    <CalcRuleRef>ucmdb_generic_CR</CalcRuleRef>
    <MsgPropRuleRef>ucmdb_generic_PR</MsgPropRuleRef>
  </Service>
  ...
```

- **Association**

  The service type association specifies the propagation rule to be used for matching associations between UCMDB CI types that match the referenced source and target service type definitions.

  The association section is contained within the `<Services>` section. Each association is specified within a section titled `<Association>`:

```
  <Association>
    <Composition/>
    <SourceRef>ucmdb_oracle_nt</SourceRef>
    <TargetRef>folder</TargetRef>
    <PropRuleRef>ucmdb_generic_PR</PropRuleRef>
  <Association>
  ..
```

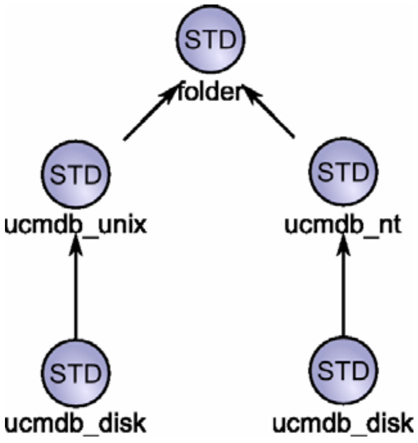
Unlike UCMDB relationships, associations are directed bottom up, that is, from the child to its parent.

Service type definitions are specified in service type definition files, which must be in XML format. The XML files have to be of the form DOCTYPE `Services`. For information on the documentation type definitions and `Services`, see the *Administrator's Reference*.

Example of STDs on UNIX or Linux

This example illustrates an extract from the default service type definition file.

Figure 20 Service Type Definition Arrangement



```
<Services>
  <Service>
    <Name>folder</Name>
    <Label>folder</Label>
    <Icon>folder.32.gif</Icon>
    <CalcRuleRef>ucmdb_generic_CR</CalcRuleRef>
    <MsgPropRuleRef>ucmdb_generic_PR</MsgPropRuleRef>
  </Service>
  <Service>
    <Name>ucmdb_nt</Name>
    <Label>ucmdb_nt</Label>
    <Icon>winnt.32.gif</Icon>
    <CalcRuleRef>ucmdb_generic_CR</CalcRuleRef>
    <MsgPropRuleRef>ucmdb_generic_PR</MsgPropRuleRef>
  </Service>
  <Service>
    <Name>ucmdb_unix</Name>
    <Label>ucmdb_unix</Label>
    <Icon>server.32.gif</Icon>
    <CalcRuleRef>ucmdb_generic_CR</CalcRuleRef>
    <MsgPropRuleRef>ucmdb_generic_PR</MsgPropRuleRef>
  </Service>
  <Service>
    <Name>ucmdb_disk</Name>
    <Label>ucmdb_disk</Label>
    <Icon>hdisk.32.gif</Icon>
    <CalcRuleRef>ucmdb_generic_CR</CalcRuleRef>
    <MsgPropRuleRef>ucmdb_generic_PR</MsgPropRuleRef>
  </Service>
...
  <Association>
    <Composition/>
    <SourceRef>ucmdb_nt</SourceRef>
    <TargetRef>folder</TargetRef>
    <PropRuleRef>ucmdb_generic_PR</PropRuleRef>
  </Association>
  <Association>
    <Composition/>
    <SourceRef>ucmdb_unix</SourceRef>
    <TargetRef>folder</TargetRef>
    <PropRuleRef>ucmdb_generic_PR</PropRuleRef>
  </Association>
  <Association>
    <Composition/>
    <SourceRef>ucmdb_disk</SourceRef>
    <TargetRef>ucmdb_nt</TargetRef>
    <PropRuleRef>ucmdb_generic_PR</PropRuleRef>
  </Association>
  <Association>
    <Composition/>
    <SourceRef>ucmdb_disk</SourceRef>
    <TargetRef>ucmdb_unix</TargetRef>
    <PropRuleRef>ucmdb_generic_PR</PropRuleRef>
  </Association>
...
</Services>
```

Uploading STDs into HPOM for UNIX or Linux

Service type definitions are uploaded into HPOM for UNIX or Linux using the `ServiceTypeDefinitionCLI` utility located in `<`*`InstallDir`*`>/bin`.

To upload service type definitions, enter the following command, specifying the location and name of the service type definition xml file:

`/opt/OV/bin/ServiceTypeDefinitionCLI -o -a <`*`filename`*`>`

ServiceTypeDefinitionCLI Command Parameters

`ServiceTypeDefinitionCLI [-l] [-a FILENAME] [-o] [-d definition]`

`-l`	Lists all STDs and associations that have been uploaded and are currently registered.
`-a` *`<filename>`*	Adds service type definitions and their associations from the specified XML file. Requires an additional parameter containing the file name. Example: `/opt/OV/bin/ServiceTypeDefinitionCLI -a std.xml`
`-o`	Overwrite existing entries. Example: `/opt/OV/bin/ServiceTypeDefinitionCLI -o -a std.xml`
`-d` *`<definition>`*	Deletes the named service type definitions and their associations. Requires an additional parameter containing the name of the service type definition. Example: `/opt/OV/bin/ServiceTypeDefinitionCLI -d ucmdb_nt`

5 Contained Relationship Files

The HPOM Service Navigator distinguishes between two major types of relationships:

- **Containments**

  A containment relationship defines the hierarchy between services. Containments are one-to-many relationships. That is, every service can have only one parent CI, but any number of children.

- **Dependencies**

  A dependency relationship models a cross-hierarchy dependency between arbitrary services. Dependencies are many-to-many relationships, that is, every service can have dependencies pointing to any number of services.

The hierarchy of a service tree is defined by a TQL query. HPOM DMA must convert these UCMDB relationships to containment relationships to create a valid service tree. This can be accomplished by classifying certain UCMB relationship types as containment relationships for Service Navigator. This classification is done by the containment relationship mapping configuration file.

Configuration File

The configuration file is an XML file that holds a list of regular expressions. All UCMDB relationship type names that match this regular expression become containment relationships, when the relationship is synchronized into Service Navigator using HPOM DMA.

The file is located in the following directory:

`<InstallDir>/newconfig/DataDir/conf/dma/containmentrelations.xml`

Syntax of the configuration file:

```
<containmentrelations>
  <match>[RegularExpression]</match>
  [<match>[RegularExpression]</match>]
  ...
</containmentrelations>>
```

➤ Regular expressions are matched against the relation types name and not the display name.

An XML schema for validating and easier editing of this file is available on the system at:

`<InstallDir>/misc/dma/schemas/containmentrelations.xsd`

It is also available at the following location:

http://<*DMA_Host*>:8081/schemas/containmentrelations.xsd

Default Configuration

The default configuration maps all UCMDB relationships that start with the string container-to-containment relationships:

```
<containmentrelations>
   <match>^container.*</match>
   <match>contained.*</match>
</containmentrelations>
```

➤ If a configuration item (CI) has more than one containment relationship, there is a clash and one relationship is discarded.

6 Node Types

The node type list defines which UCMDB CI types are imported as nodes into the HPOM node groups.

The node types available in the UCMDB and required by HPOM must be specified in the `nodetypes.xml` file:

`<SharedDir>/server/conf/dma/nodetypes.xml`

Format:

```
<nodetypes>
  <type>[UCMDB Node Type]</type>
  ... more node types
</nodetypes>
```

Example:

```
<nodetypes>
  <type>unix</type>
  <type>nt</type>
  <type>host</type>
</nodetypes>
```

7 Synchronization Packages

Synchronization packages are sets of configuration files. They are used to:

- Transform CIs into HPOM services and nodes
- Synchronize specified services and nodes in HPOM with data from the UCMDB

A synchronization package can be created and delivered by SPI or application developers, and deployed on an HPOM DMA installation.

A synchronization package must include the synchronization package descriptor file (`bundle.xml`) to define the synchronization package.

Optional files include:

- Service Mappings (`servicemapping.xml`)
- Node Mappings (`nodemapping.xml`)
- Attribute Mapping (`attributemapping.xml`)
- User Profile Mapping (`usermapping.xml`)
- Pre-mapping script File (`premapping.groovy`)
- Pre-synchronization script File (`preupload.groovy`)
- Post-synchronization script File (`postupload.groovy`)

HPOM DMA includes the synchronization packages for nodes and selected databases. For example, if web servers are required, an appropriate additional mapping is required. Integrators can create these synchronization packages to enable UCMDB to HPOM synchronization to suit the needs of the IT environment.

For basic information on mapping, see Appendix A, Mapping Syntax.

Synchronization Package Descriptor File

The `bundle.xml` file specifies the synchronization package name, its priority, and the associated TQL queries.

The highest priority is represented by `1`. The default synchronization package is assigned the lowest priority of `10`. A higher priority rule result overwrites a result from a lower priority rule.

➤ There may be more than one synchronization package with the same priority. The order of execution of the rules between synchronization packages with the same priority is not specified.

Users are then able to activate these synchronization packages in the HPOM DMA console under `Configuration`.

The following is an example of a `bundle.xml` file:

```
<Bundle>
  <Name>MS SQL</Name>
  <Description>Out of the box synchronization package for
   Microsoft SQL Server databases</Description>
  <Priority>7</Priority>
  <TQLs>
    <TQL>MS SQL Server (Operations)</TQL>
  </TQLs>
</Bundle>
```

The `MS SQL Server (Operations)` TQL query is queried and the results are synchronized.

Mappings

HPOM DMA supports the following types of mapping files.

Service Mappings

Service mappings define the relationships between the type of a CI in the UCMDB and the service type definition of a service in HPOM.

For details, see Chapter 8, Service Mapping.

Node Mappings

The nodes selected from the UCMDB are mapped to the HPOM node groups as defined by the node mapping file (`nodemapping.xml`) from the associated synchronization package.

For details, see Chapter 9, Node Mapping.

Attribute Mapping

Attribute mapping enables you to change CI attributes and add new attributes to better describe a service and create a more detailed view of the environment.

For details, see Chapter 10, Attribute Mapping.

User Profile Mapping

User profile mapping enables you to map a service to an HPOM user profile. After synchronization, the service is assigned to all operators that are included in any specified HPOM user profile.

For details, see Chapter 11, User Profile Mapping.

Customizing Synchronization and Scripting

Scripting enables you to perform additional processing and customization during the synchronization process before the mapping and before and after the upload of nodes and services to HPOM. One pre-mapping, one pre-upload, and one post-upload script can be associated with each synchronization package.

For details, see Chapter 12, Scripting.

Synchronization Package Locations

The `sync-packages` directory contains dedicated subdirectories for each synchronization package. It is recommended but not essential that you use directory names that match the synchronization package name.

Synchronization packages are deployed by placing them into the following directory:

`<SharedDir>/server/conf/dma/sync-packages/<SyncPackageName>`

The default location is:

- **Windows**

  32-bit: `C:\Documents and Settings\All Users\Application Data\HP\HP BTO Software\shared\server\conf\dma\sync-packages\<SyncPackageName>`

  64-bit: `C:\ProgramData\HP\HP BTO Software\shared\server\conf\dma\sync-packages\<SyncPackageName>`

- **UNIX and Linux**

  `/var/opt/OV/shared/server/conf/dma/sync-packages/<SyncPackageName>`

The synchronization package is then available in the HPOM DMA console. You can select and activate it.

8 Service Mapping

Service mappings define the relationships between the type of a configuration item (CI) in the UCMDB and the service type definition of a service in HPOM. For example, if the UCMDB CI type `oracle` is a child of CI type `unix`, they are mapped to the HPOM service type definition `ucmdb_oracle_unix`.

The type of incoming objects from the UCMDB must be mapped to a service type definition. This is handled by the Mapping Engine. For generating services in HPOM, it is necessary to map a CI type (Source Type) in the UCMDB or BAC to a Target Type in HPOM.

Service Mapping File

Service mappings are specified in the `servicemapping.xml` file of the associated synchronization package. Based on the condition, a service is assigned to an STD. For information on STDs, see Chapter 4, Service Type Definition Files.

➤ If you map to service type definitions and definitions for associations, they must exist in HPOM and fulfill the constraints of the STD associations.

The default service mapping maps UCMDB CI types to service type definitions using the following relationship:

`<citype>` in the UCMDB is mapped to the STD `ucmdb_<`*`citype`*`>`

Example:

If the CI type is `disk`, the STD maps this as `ucmdb_disk`.

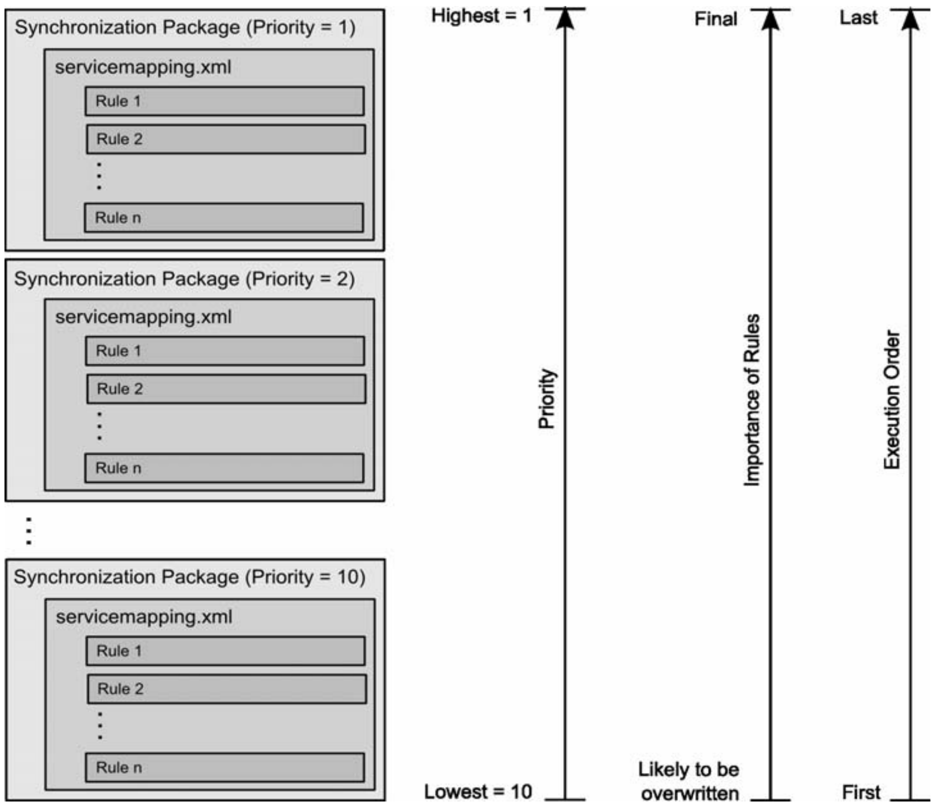
The default STD mapping is defined in the `servicemapping.xml` file of the default bundle. The contents of this synchronization package should not be changed. If you require additional service mapping, it is recommended that you create a new synchronization package with additional service mappings.

For mapping syntax information, see Appendix A, Mapping Syntax.

If the default patterns meet your needs, there is no need to write a dedicated service mapping file for your synchronization package. For example, a specific service mapping is required for the standard SQL Server database package, to fetch different versions of SQL Servers. Example of a servicemapping.xml File on page 78 illustrates how this service mapping is specified.

Rules are applied in reverse order to the CI to make sure that the top-most and higher prioritized rules overwrite values set by lower prioritized rules. Service mapping is applied by all matching rules and are executed according to the priority of the synchronization package and the position within the rules declarations. A service that is set by a rule that is contained in a higher prioritized bundle and is above other rules in the same mapping file overrides values set by lower prioritized rules or set by rules that are located below the current rule.

Figure 21 Synchronization Package Priority



In Example of a servicemapping.xml File, the service mapping for SQL Servers contained in the SQL Server synchronization package contains two rules:

```
<Rule name="SqlServer 6.5">
```

```
<Rule name="SqlServer">
```

The rule `SqlServer` is executed first. This sets the STD to `ucmdb_sqlserver` for all CIs that are of the type `sqlserver`. The rule `SqlServer 6.5` is checked next. If the CI contains the attribute `database_dbversion` and this is set to 6.5, the previously set STD `ucmdb_sqlserver` is overwritten to the new value `ucmdb_sqlserver65`.

Example of a servicemapping.xml File

The following example show the contents of a `servicemapping.xml` file in a synchronization package.

In this example, all discovered services with the UCMDB CI type `sqlserver` and where `database_dbversion` attribute value is `6.5` are mapped to the service type definition `ucmdb_sqlserver65`. All other CIs of type `sqlserver` are mapped to service type definition `ucmdb_sqlserver`.

➤ The SqlServer rule (`<Rule name="SqlServer">`) is not strictly necessary as the service mapping in the default synchronization package achieve the same result.

➤ All elements are described in detail in Mapping Syntax on page 125.

```
<?xml version="1.0" encoding="utf-8"?>
<Mapping>
  <Rules>
    <Rule name="SqlServer 6.5">
      <Condition>
        <And>
          <Equals ignoreCase="true">
            <CiType />
            <Value>sqlserver</Value>
          </Equals>
          <Contains>
            <Attribute>database_dbversion</Attribute>
            <Value>6.5</Value>
          </Contains>
        </And>
      </Condition>
      <MapTo>
        <STD>
          <Value>ucmdb_sqlserver65</Value>
        </STD>
      </MapTo>
    </Rule>
    <Rule name="SqlServer">
      <Condition>
        <Equals ignoreCase="true">
          <CiType />
          <Value>sqlserver</Value>
        </Equals>
      </Condition>
      <MapTo>
        <STD>
          <CiType />
          <Value>ucmdb_sqlserver</Value>
        </STD>
      </MapTo>
    </Rule>
  </Rules>
</Mapping>
```

Overriding STD Properties

Using these additional mapping actions, you can override default values of an STD assigned to matched CIs. All actions can be specified independently of each other and independently of the STD mapping action. As a result, you can override only selected values. For example, you can change only the icon of a specified CI that is mapped to the fallback default STD.

`ParentPropagationRule` and `ParentWeight` determine the propagation to the parent of the service.

`MessagePropagationRule` and `MessageWeight` determine the propagation of the messages to the service.

➤ Calculation and Propagation Rules must exist prior to synchronization.

For more information about Calculation Rules, Propagation Rules and the Weight, see the HPOM documentation.

Simple Example of STD-Properties Overrides

This example shows the content of a `servicemapping.xml` file in a synchronization package with Service Type Definition Properties overrides.

As in Example of a servicemapping.xml File on page 78, the CI is mapped to `ucmdb_apache_std` by the first rule. However, the propagation rule with the name `least_critical` is used as the Apache server in this example is part of a load balancing cluster. The second rule sets the icon names for all CIs to `<CI Type>_icon` unless the icon is already specified in a previous rule, for example, the Apache web server CI to `apache_icon`.

```
<?xml version="1.0" encoding="utf-8"?>
<Mapping>
    <Rules>
        <Rule name="Apache Server">
            <Condition>
                <And>
                    <Equals ignoreCase="true">
                        <CiType />
                        <Value>apache</Value>
                    </Equals>
                    <Equals>
                        <Attribute>cluster</Attribute>
                        <Value>true</Value>
                    </Equals>
                </And>
            </Condition>


            <MapTo>
                <STD>
                    <Value>ucmdb_</Value>
                    <CiType />
                    <Value>_std</Value>
                </STD>
                <MessagePropagationRule>
                    <Value>least_critical</Value>
                </MessagePropagationRule>
            </MapTo>
        </Rule>
        <Rule name="Set Icons">
            <Condition>
                <True />
            </Condition>
            <MapTo>
                <Icon>
                    <CiType />
                    <Value>_icon</Value>
                </Icon>
            </MapTo>
        </Rule>
    </Rules>
</Mapping>
```

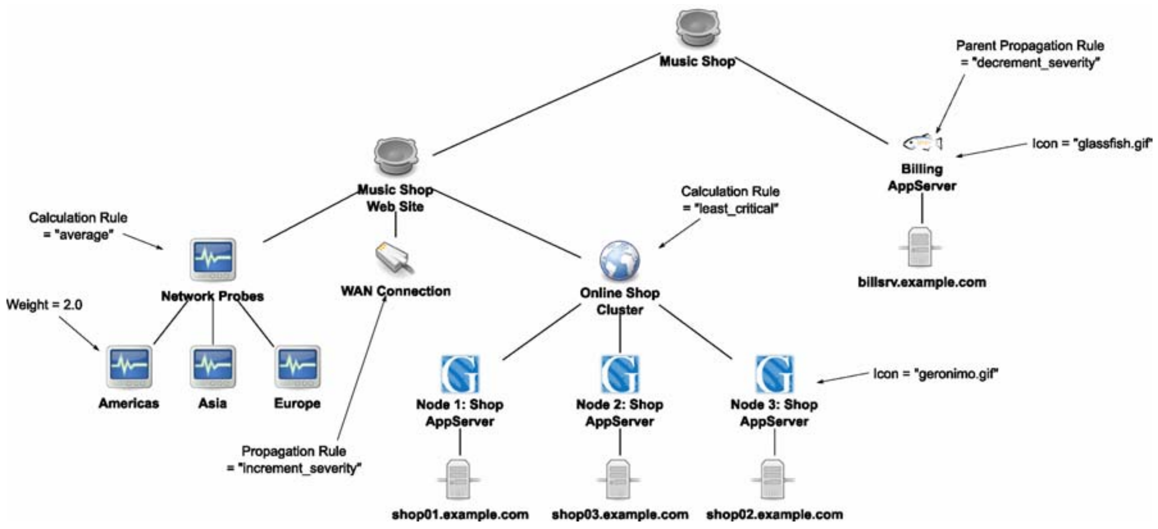
Complex Example of STD-Properties Overrides

This examples describes the environment of a imaginary online music shop. The desired service tree is shown in Figure 22 on page 82.

Names of CMDB types, calculation rules, propagation rules, icons and STDs are not part of the out of the box data. If you wish to recreate a similar example in a production environment, you must make sure that these rules, icons and STDs exist. You may need to create some manually. For more information, see the UCMDB and HPOM documentations.

The online shop is powered by an application server cluster composed of three Apache Geronimo application servers. This cluster is monitored by network probes that are installed in three different locations according to the markets in Americas, Asia, and Europe with the primary market being the Americas. Another application server powered by a Sun Glassfish application server provides an internal billing system.

Figure 22 Service Tree for the Online Music Shop



There are three main steps to consider:

- Setting Rules for Application Servers on page 82
- Setting Rules for Application Servers on page 82
- Setting Rules for Network Probes on page 85

Setting Rules for Application Servers

First we create the rules in a `servicemapping.xml` for the application servers. There are two different application servers. To be able to differentiate these servers at a glance, different icons are required. The rule `Set STD and icons for Application Servers` sets the STD and provides different icons depending on the application server type.

Because the application servers for the shop are arranged in a cluster, the failure of up to two nodes does not result in a failure of the cluster and the application remains available. The cluster should only reflect the state of the least critical message of any cluster node. To get the cluster to reflect the state of the least critical message, use the calculation rule `least_critical`, which is set by the rule `Set calculation rule for cluster`.

Although any outage of the online shop may result in the loss of business, the billing system is less critical. If customers receive bills a few days late, they are not unduly upset. The status of the billing system is not as critical as that of the shop web site. To reduce the severity, we set the parent propagation rule of the billing application server using the rule `Set parent propagation rule for billing below Music Shop` to `decrement_severity`. We do this on the parent propagation rule because we do not want to change the general propagation rule of the billing application server, as it might be critical in a different context. Only in the context of this tree, it is less critical.

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://localhost:8081/hpdma/schemas/
mapping.xsd">
    <Rules>
        <Rule name="Set parent propagation rule for billing below Music
         Shop">
            <Condition>
                <And>
                    <Equals>
                        <CiCaption/>
                        <Value>Billing AppServer</Value>
                    </Equals>
                    <Equals>
                        <XPathResult>/caption</XPathResult>
                        <Value>Music Shop</Value>
                    </Equals>
                </And>
            </Condition>
            <MapTo>
                <ParentPropagationRule>
                    <Value>decrement_severity</Value>
                </ParentPropagationRule>
            </MapTo>
        </Rule>
        <Rule name="Set calculation rule for cluster">
            <Condition>
                <Equals>
                    <CiCaption/>
                    <Value>Online Shop Cluster</Value>
                </Equals>
            </Condition>
            <MapTo>
                <CalculationRule>
                    <Value>least_critical</Value>
                </CalculationRule>
            </MapTo>
        </Rule>
        <Rule name="Set STD and icons for Application Servers">
            <Condition>
                <Or>
                    <Equals>
                        <CiType />
                        <Value>glassfish</Value>
                    </Equals>
                    <Equals>
                        <CiType />
                        <Value>jboss</Value>
                    </Equals>
                    <Equals>
                        <CiType />
                        <Value>geronimo</Value>
                    </Equals>
                </Or>
            </Condition>
            <MapTo>
                <STD>
                    <Value>app_server</Value>
                </STD>
                <Icon>
                    <CiType/>
                    <Value>.gif</Value>
                </Icon>
        </Rule>
    </Rules>
</Mapping>
```

Setting Rules for WAN Connections

After creating the rules for the application servers, we consider the WAN connection of our server to the Internet. Even a warning here may result in slow response times. So this is very critical to the overall availability. In contrast to the billing application server, this is true in every context, so we set the message propagation rule of the CI itself to the calculation rule `increment_severity`, which increments the severity of each message by one.

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://localhost:8081/hpdma/schemas/
mapping.xsd">
    <Rules>
        <Rule name="Set propagation rule for WAN connection">
            <Condition>
                <And>
                    <Equals>
                        <CiType/>
                        <Value>networklink</Value>
                    </Equals>
                    <StartsWith>
                        <CiCaption/>
                        <Value>WAN</Value>
                    </StartsWith>
                </And>
            </Condition>
            <MapTo>
                <MessagePropagationRule>
                    <Value>increment_severity</Value>
                </MessagePropagationRule>
            </MapTo>
        </Rule>
    </Rules>
</Mapping>
```

Setting Rules for Network Probes

After creating the WAN connection of our server to the Internet, we take care of the network probes that monitor the external availability of the shop. There are three different network probes located in the regions of the target markets. These are collected under the Network Probes CI, which calculate the overall network probe status using the average of these three locations. The rule `Set calculation rule for top level network probe CI` sets the calculation rule average for this purpose.

Because most customers are located in the Americas region, availability is most important here. By setting the message weight to 2.0, the status of the Americas network probe is propagated with twice the importance. So the calculation of the Network Probes CI is biased towards the result of the Americas network probe.

All other STDs properties of the network probe CIs may be derived by DMA from the default STD that is mapped by the default synchronization package. We want an alternative icon here, which is set through the rule `Set icon for network probes. Use the DMA supplied default STD`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://localhost:8081/hpdma/schemas/
mapping.xsd">
    <Rules>
       <Rule name="Set calculation rule for top level network probe CI">
            <Condition>
                <Equals>
                    <CiCaption/>
                    <Value>Network Probes</Value>
                </Equals>
            </Condition>
            <MapTo>
                <CalculationRule>
                    <Value>average</Value>
                </CalculationRule>
```

```
        </MapTo>
    </Rule>
    <Rule name="Set a higher weight for probes in key markets">
        <Condition>
            <And>
                <Equals>
                    <CiType/>
                    <Value>networkprobe</Value>
                </Equals>
                <Equals>
                    <Attribute>location</Attribute>
                    <Value>americas</Value>
                </Equals>
            </And>
        </Condition>
        <MapTo>
            <MessageWeight>2.0</MessageWeight>
        </MapTo>
    </Rule>
    <Rule name="Set icon for network probes. Use DMA supplied
     default STD.">
        <Condition>
            <Equals>
                <CiType/>
                <Value>networkprobe</Value>
            </Equals>
        </Condition>
        <MapTo>
            <Icon>
                <CiType/>
                <Value>probe.gif</Value>
            </Icon>
        </MapTo>
    </Rule>
</Rules>
</Mapping>
```

Mapping UCMDB Information to Services

HPOM DMA imports data from the UCMDB into HPOM. The automation process can synchronize this data on a regular basis. You may have already defined services directly in HPOM, and want to incorporate the data imported from the UCMDB with the data already present in Service Navigator.

The term `ExternalCi` refers to CIs that exist in HPOM but do not originate from the UCMDB and have not been imported using HPOM DMA. External service CIs are service CIs which are not synchronized during the synchronization task.

Dependency relations of services are "owned" by the target service (source and target is defined by the direction of the status propagation). Only those dependencies where a UCMDB service is the target can be completely controlled by the synchronization task (create, update, delete). Dependencies where a UCMDB service is the source can only be created or updated by the synchronization task, if necessary they need to be deleted manually.

External dependencies can be achieved using Service mapping.

Creating Dependencies from External CIs to Internal CIs

External CI dependency creation associates service CIs to services that are already present in Service Navigator, and that are not part of the HPOM DMA synchronization. The following syntax is used:

```
<DependencyFromExternalCi>
  <ExternalCiId>
       [Operand]
       ...
  </ExternalCiId>
  <DependencyType>
       [Operand]
       ...
  </DependencyType>
  <PropagationRuleName>
       [Operand]
       ...
  </PropagationRuleName>
  <PropagationWeight>[Double]</PropagationWeight>
</DependencyFromExternalCi>
```

This service mapping creates a dependency from the external service CI identified by the CI referenced in the `<ExternalCiId>` tag. The type of the dependency is specified in the value of the `<DependencyType>` tag. Both values can be concatenated by any operands.

Optionally, the propagation rule and propagation weight of the dependency can be specified in the `<PropagationRuleName>` and `<PropagationWeight>` tags.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="schemas/mapping.xsd">
  <Rules>
    <Rule name="Create dependency from external CI">
      <Condition>
        <Contains>
          <CiType />
          <Value>unix</Value>
        </Contains>
      </Condition>
      <MapTo>
        <DependencyFromExternalCi>
          <ExternalCiId>
            <Value>SVCDISC</Value>
            <Value>:</Value>
            <Value>Cluster</Value>
          </ExternalCiId>
          <DependencyType>
            <Value>Dependency</Value>
          </DependencyType>
          <PropagationRuleName>
            <Value>test_PR</Value>
          </PropagationRuleName>
          <PropagationWeight>2.0</PropagationWeight>
        </DependencyFromExternalCi>
      </MapTo>
    </Rule>
  </Rules>
</Mapping>
```

Creating Dependencies from Internal CIs to External CIs

Internal CI dependency creation associates service CIs to services that are already present in the HPOM DMA synchronization, and that are not part of Service Navigator. The following syntax is used:

```
<DependencyToExternalCi>
  <ExternalCiId>
       [Operand]
       ...
  </ExternalCiId>
  <DependencyType>
       [Operand]
       ...
  </DependencyType>
  [<PropagationRuleName>
       [Operand]
       ...
  </PropagationRuleName>
  <PropagationWeight>[Double]</PropagationWeight>]
</DependencyToExternalCi>
```

This service mapping creates a dependency to the external service CI identified by the CI referenced in the `<ExternalCiId>` tag. The type of the dependency is specified in the value of the `<DependencyType>` tag. Both values can be concatenated by any operands.

Optionally the propagation rule and propagation weight of the dependency can be specified in the `<PropagationRuleName>` and `<PropagationWeight>` tags.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="schemas/mapping.xsd">
  <Rules>
    <Rule name="Create dependency to external CI">
      <Condition>
        <Contains>
          <CiType />
          <Value>nt</Value>
        </Contains>
      </Condition>
      <MapTo>
        <DependencyToExternalCi>
```

```
          <ExternalCiId>
            <Value>SVCDISC</Value>
            <Value>:</Value>
            <Value>Applications</Value>
          </ExternalCiId>
          <DependencyType>
            <Value>Dependency</Value>
          </DependencyType>
          <PropagationRuleName>
            <Value>test_PR</Value>
          </PropagationRuleName>
          <PropagationWeight>3.0</PropagationWeight>
        </DependencyToExternalCi>
      </MapTo>
    </Rule>
  </Rules>
</Mapping>
```

9 Node Mapping

HPOM node groups are used to organize your nodes and to automate the deployment of the monitoring environment of nodes that are added to a node group. Node mapping is used by HPOM DMA to specify the rules required to assign nodes imported from the UCMDB to node groups in HPOM.

Node Mapping File

The nodes selected from the UCMDB are mapped to the HPOM node groups as defined by the node mapping files (`nodemapping.xml`) from all activated synchronization packages.

The default node group is CMDB and all synchronized nodes are sent to this node group in addition to any node groups specified in the dedicated `nodemapping.xml` files.

The node type list defines which UCMDB CI types are imported as nodes into the HPOM node groups. For further information, see Node Types on page 69.

Example of Node Mapping

```
<?xml version="1.0" encoding="utf-8"?>
<Mapping>
    <Rules>
        <Rule name="Unix Oracle Nodes">
            <Condition>
                <And>
                    <Equals>
                        <CiType/>
                        <Value>unix</Value>
                    </Equals>
                    <Equals>
                        <Attribute>host_servertype</Attribute>
                        <Value>oracle</Value>
                    </Equals>
                </And>
            </Condition>
            <MapTo>
                <NodeGroup>
                    <Value>DBSPI_Oracle_Unix_Nodes</Value>
                </NodeGroup>
            </MapTo>
        </Rule>
        <Rule name="Oracle Windows Nodes">
            <Condition>
                <And>
                    <Equals>
                        <CiType/>
                        <Value>nt</Value>
                    </Equals>
                    <Equals>
                        <Attribute>host_servertype</Attribute>
                        <Value>oracle</Value>
                    </Equals>
                </And>
            </Condition>
            <MapTo>
                <NodeGroup>
                    <Value>DBSPI_Oracle_Windows_Nodes</Value>
                </NodeGroup>
            </MapTo>
        </Rule>
     </Rules>
</Mapping>
```

In this example:

- The rule `Unix Oracle Nodes` searches for nodes of the CI type `unix` where the attribute `host_servertype` is set to `oracle` and places these nodes into the node group `DBSPI_Oracle_Unix_Nodes`.
- The rule `Oracle Windows Nodes` searches for nodes of the CI type `nt` where the attribute `host_servertype` is set to `oracle` and places these nodes into the node group `DBSPI_Oracle_Windows_Nodes`.

10 Attribute Mapping

Attribute mapping enables you to do the following:

- Change CI attributes and add new attributes to:
  - Better describe a service and create a more detailed view of the environment.
  - Enable Smart Message Mapping to enrich HPOM messages with custom message attributes.
- Set the `hosted_on` attribute to the fully qualified hostname so that Smart Message Mapping can work correctly.
- Customize the default service ID that HPOM DMA assigns to services synchronized from UCMDB.

Attribute mapping is applied by all matching rules and is executed according to the priority of the bundle and the position within the rules declarations in a a similar way to STD mapping. For details, see Service Type Definition Files on page 59.

An attribute that is set by a rule that is contained in a higher prioritized bundle and is above other rules in the same mapping file overrides attributes set by lower prioritized rules or set by rules that are located below the current rule.

Attribute Mapping File

Attribute mappings are specified in the `attributemapping.xml` file from the associated synchronization package.

In the default `attributemapping.xml` file, the following attributes are mapped:

- Attribute required for Smart Message Mapping:

The `hosted_on` attribute is required by Smart Message Mapping.

- Attributes required for BAC and UCMDB for UI Launch:
  - `CmdbCiCaption`
  - `CmdbCiType`
  - `CmdbObjectID`
- Attributes for nodes:
  - `system_type`
  - `os_type`
  - `version`

These attributes can be used as parameters to call tools in HPOM.

Syntax: `$OPC_SERVICE_VALUE[<attribute>]`

Example: `$OPC_SERVICE_VALUE[hosted_on]`

To view a working example, see the standard UI Launch tools.

Example of Attribute Mapping

The example attribute mapping in this section illustrates how to set specific attributes for a selected CI type with particular attributes. For all CIs that are of the CI type `unix` and that have the following attributes:

- `host_os` set to `HP-UX`
- `host_model` set to `ia64`
- `data_description` containing `B.11.23`

The following attributes are set:

- `ovo_osType` to `HP-UX_64`
- `ovo_systemType` to `Itanium Compatible`
- `ovo_osVersion` to `B.11.23`

For further information about mapping syntax, see Mapping Syntax on page 125.

```
<?xml version="1.0" encoding="utf-8"?>
<Mapping>
   <Rules>
      <Rule name="Set node attributes for HP-UX B11.23 Itanium">
         <Condition>
            <And>
               <Equals>
                  <CiType />
                  <Value>unix</Value>
               </Equals>
               <Equals>
                  <Attribute>host_os</Attribute>
                  <Value>HP-UX</Value>
               </Equals>
               <Equals>
                  <Attribute>host_model</Attribute>
                  <Value>ia64</Value>
               </Equals>
               <Contains>
                  <Attribute>data_description</Attribute>
                  <Value>B.11.23</Value>
               </Contains>
            </And>
         </Condition>
         <MapTo>
            <Attribute>
               <Name>ovo_osType</Name>
               <SetValue>
                  <Value>HP-UX_64</Value>
               </SetValue>
            </Attribute>
            <Attribute>
               <Name>ovo_systemType</Name>
               <SetValue>
                  <Value>Itanium Compatible</Value>
               </SetValue>
            </Attribute>
            <Attribute>
               <Name>ovo_osVersion</Name>
               <SetValue>
                  <Value>B.11.23</Value>
               </SetValue>
            </Attribute>
         </MapTo>
      </Rule>
   </Rules>
</Mapping>
```

11 User Profile Mapping

User profile mapping enables you to map a service to an HPOM user profile. After synchronization, the service is assigned to all operators that are included in any specified HPOM user profile.

For all services, the mapping starts with the rules of the highest priority and executes all rules with progressively lower priority until the rules with the lowest priority have been executed.

For user profile mapping, the order of rule execution is not important. CIs are assigned to all user profiles that match and matching rules do not conflict with each other.

User Profile Mapping File

User profile mappings are specified in the `usermapping.xml` file of the associated synchronization package:

```
<SharedDir>/server/conf/dma/sync-packages/<syncPackage>/
usermapping.xml
```

Example of User Profile Mapping

In this example, Apache web server CIs are assigned to the global operators responsible for web server operations as well as to the local web server operations users, for example, `London_web_op` for Apache web servers that are located in London:

```
<?xml version="1.0" encoding="utf-8"?>
<Mapping>
    <Rules>
        <Rule name="Assign Apache Server users">
            <Condition>
                <Equals ignoreCase="true">
                    <CiType/>
                    <Value>apache</Value>
                </Equals>
            </Condition>
            <MapTo>
                <UserProfile>
                    <Value>global_web_op</Value>
                </UserProfile>
                <UserProfile>
                    <Attribute>location</Attribute>
                    <Value>_web_op</Value>
                </UserProfile>
            </MapTo>
        </Rule>
    </Rules>
</Mapping>
```

For further information about mapping syntax, see Mapping Syntax on page 125.

12 Scripting

Scripting enables you to perform additional processing and customization during the synchronization process:

- Pre-mapping scripts are executed before the mapping rules are applied.
- Pre-upload scripts are executed before the upload of nodes and services to HPOM.
- Post-upload scripts are executed after the upload of nodes and services to HPOM.

One script of each type can be associated with each synchronization package. These optional script files are located in the associated synchronization package directory. For details of synchronization package locations, see Synchronization Packages on page 71.

Associating script files with synchronization packages simplifies the distribution of scripts and enables script development to be handled independently of the working environment. The execution of synchronization scripts follows the settings of the synchronization packages:

- Only scripts in active synchronization packages are executed.
- Scripts are executed in the order of the priority of the synchronization packages.

⚠ Script execution is potentially insecure. In particular, the use of `DMA.exec(...)` commands can cause damage to an installation. To enhance security, script access for editing is allowed on the file system level only. This makes sure that only users with log-on credentials to the HPOM DMA host can edit scripts. This protects the scripts by the log-on security of the HPOM DMA host.

Script Syntax

Groovy is an object-oriented programming language for the Java platform. It can be used as a scripting language for the Java platform. HPOM DMA supports Groovy for its scripting capabilities.

Groovy uses a Java-like curly bracket syntax that is dynamically compiled to JVM bytecodes, and that works well with other Java code and libraries. For more information about Groovy and documentation describing the Groovy language, visit: **http://groovy.codehaus.org**

Script Files

Script files are identified using fixed names within synchronization package directories:

- `preMapping.groovy`— script to be executed before mapping
- `preUpload.groovy` — script to be executed before upload
- `postUpload.groovy` — script to be executed after upload

The upload is performed between execution of the active `preUpload.groovy` scripts and the `postUpload.groovy` scripts. The Script Summary page of the HPOM DMA console displays the execution order and the data upload.

Script Variables

Each script has two predefined variables:

DMA

Object Type:	`com.hp.ov.om.dma.scripting.IDmaScriptingInterface`
Description:	Enables access to DMA function calls to manipulate synchronization data and control the synchronization.

syncData

Object Type:	`com.hp.ov.om.dma.common.data.sync.ISyncData`

Description: Provides access to the data that is synchronized.

For more information about the object types, see the online API documentation, which can be accessed on every DMA installation at:

http://<*hostname*>:8081/hpdma/apidoc/index.html

Scripts within a synchronization package share the same variable scope. That means variables assigned in `preMapping.groovy` can be later used in the corresponding `preUpload.groovy` and `postUpload.groovy`. Scripts from different synchronization packages do not share variables with the same name, which avoids name clashes and undesired side effects.

Handling Errors

Errors in scripts result in the generation of exceptions. The error handling is around each script invocation. By default, an exception in a script aborts the synchronization. This behavior can be changed by calling the command:

```
DMA.setAbortSyncOnError(boolean)
```

When set to false, you can enforce a script failure using the method `DMA.abortSync("...")`. For example, your script checks conditions, and because of these forced failures, a synchronization cannot be completed.

Table 1 Relation to Synchronization Status

Status	Script behavior
Synchronization OK	Scripting completed without errors and without forced synchronization interruption within a script. Scripting completed without errors even if an exception is thrown, and `AbortSyncOnError` is set to false.
Synchronization failed	A script execution caused an exception or the script forced a failure because of a scripting condition using the `DMA.abortSync(String)` command.

Enabling and Disabling Scripts

To help identify the cause synchronization failure, scripting can be disabled. If there is an error in a script, disabling scripting should enable successful synchronization.

To disable synchronization package script execution, clear the **Enable Script Execution** check box from the Content page of the HPOM DMA console and click **Save**.

➤ If script execution is disabled, a warning is displayed on the Script Summary page of the HPOM DMA console.

Sample Scripts

The following examples are also available on your HPOM DMA host system after installation at:

`<InstallDir>/misc/dma/samples`

Creating Dependency Associations to External Services

This example iterates through all the services that are synchronized from the UCMDB.

For all services of the cmdb CI type `nt`, the script creates a dependency association so that the `DNS_Server` service propagates its status to these services.

For all services of the cmdb CI type `unix`, the script creates a dependency association so that the `Unix_Domain` service propagates its status to these services.

```
import com.hp.ov.om.dma.common.data.ci.ICi
import com.hp.ov.om.dma.common.data.ci.ICiRelation

// create two external Cis to use as reference.
ICi toCi = DMA.getExternalCiStub("DNS_Server")
ICi fromCi = DMA.getExternalCiStub("Unix_Domain")
// iterate over all Cis
for (ICi ci : syncData.getConfigurationItems()) {
  if (ci.getAttributeValue("CmdbCiType") == "nt") {
    // if type is nt, add a dependency from this ci to the DNS_Server.
    DMA.createRelation(ci, toCi, "Dependency")
  }
  if (ci.getAttributeValue("CmdbCiType") == "unix"){
    // if type is unix, add a dependency from Unix_Domain to this ci.
    DMA.createRelation(fromCi, ci, "Dependency")
  }
}
```

Modifying Dependency Associations

This sample modifies the dependency association from `test1` to `test2` (`test2` propagates its status to `test1`) and sets the propagation rule of the dependency association to `test_PR` and its weight to `2`.

```
import com.hp.ov.om.dma.common.data.ci.ICi
import com.hp.ov.om.dma.common.data.ci.ICiRelation

// iterate over all cis
for (ICi ci : syncData.getConfigurationItems()) {
  // iterate over all the dependency associations
  for (ICiRelation rel : ci.getDependencyRelations()) {
    if (rel.getFrom().getId() == "test1" && rel.getTo().getId() ==
    "test2") {
      // if the dependency is from test1 to test2, set the properties
      rel.setPropagationRuleName("test_PR")
      rel.setPropagationWeight(new Double(2))
    }
  }
}
```

Submitting an opcmsg Message

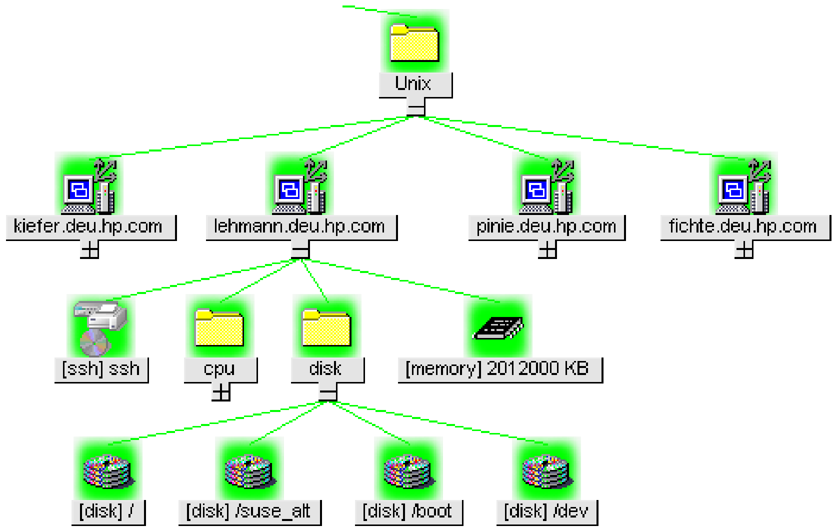
This sample script executes `opcmsg` after a synchronization submitting its outcome.

```
/* (C) Copyright 2008 Hewlett-Packard Development Company, L.P. */

/*
 * This sample script executes opcmsg after a sync submitting its
 * outcome.
 */

import com.hp.ov.om.dma.common.data.ci.CiMessageSeverity

def command = "opcmsg"

def application = "application=hpdma"
def targetObject = "object=HPDMA"

def severitySyncSucceeded = "severity=normal"
def severitySyncFailed = "severity=major"

def messageSuccess = "msg_text=The sync finished successfully."
def messageFailure = "msg_text=The sync failed."

def message = []

if ( syncData.hasConfigurationItemsWithMessages(
CiMessageSeverity.SEVERE ) ) {
  message = [ "$command", "$severitySyncFailed", "$application",
"$targetObject", "$messageFailure" ]
} else {
  message = [ "$command", "$severitySyncSucceeded", "$application",
"$targetObject", "$messageSuccess" ]
}
// message is a ArrayList in the Groovy context
// so it has be casted to a String[]
DMA.exec( (String[]) message )
```

Grouping Multiple Host Resource Types

If a node has multiple host resources of the same type, the resources are grouped under a new subgroup. Figure 23 illustrates the grouping of disks under a `disk` subgroup.

Figure 23 Grouping Multiple Disks Under a New Subgroup



This example is also available as an out-of-the-box synchronization package.

```
/* (C) Copyright 2008 Hewlett-Packard Development Company, L.P. */

/*
 * This sample iterates through all the nodes and checks for multiple
 * occurrences of host resources of the same type. Host resource types that
 * occur more than once will then be grouped under a new virtual service.
 */

import com.hp.ov.om.dma.common.data.ci.ICi
import com.hp.ov.om.dma.common.data.ci.INode
import com.hp.ov.om.dma.common.data.ci.ICiRelation

// Grouping host resources is nice-to-have, but not critical.
DMA.setAbortSyncOnError(false)

// create two external Cis to use as reference.
for (INode node in syncData.getNodes()) {
    def childrenTypes = node.getNavigator().getValues("./children/ci/type")
    def multipleTypes = [:]
    childrenTypes.each {
        def typeCount = childrenTypes.count(it)
        if (typeCount > 1 && ! multipleTypes.containsKey(it) ) {
            multipleTypes[it] = typeCount
        }
    }

    for (String resourceType in multipleTypes.keySet() ) {
        DMA.logInfo("Node $node has multiple host resources of the type
        $resourceType. Grouping host resources.");
        // Create virtual service for the host resource group
        ICi virtualService = DMA.createCi(resourceType + "@@" +
        UUID.randomUUID());
        virtualService.setCaption(resourceType);
        virtualService.setType("servicegroup");
        DMA.performEnrichment(virtualService);

        // Add the virtual service to the node
        DMA.createRelation(node, virtualService, "container")

        // Move the host resources below the virtual service
        def hostResources = node.getNavigator().getCiValues("./children/
        ci[type='$resourceType']");
        for (ICi hostResource in hostResources) {
            node.removeRelation(hostResource.getParentRelation())
            DMA.createRelation(virtualService, hostResource, "container")
        }
    }
}
```

Grouping Synchronized Nodes Alphabetically

This sample creates an alphabetical sub-grouping of the synchronized nodes in Service Navigator.

This script creates new STDs. You must either create these STDs manually or enable the automatic STD creation. For details, see the *Installation and Configuration Guide*.

```
/* (C) Copyright 2008 Hewlett-Packard Development Company, L.P. */

/*
 * This sample script creates an alphabetical hierarchy
 * of the synchronized nodes.
 */

import com.hp.ov.om.dma.common.data.ci.ICi
import com.hp.ov.om.dma.common.data.ci.INode

// instance of IDmaScriptingInterface --> DMA
// instance of ISyncData --> syncData

// A map that will map letters to corresponding new CIs
def letterCIs = [:]

for (INode node : syncData.getNodes()) {
  // Checks all synced UCMDB/BAC nodes and caches their caption's first letter
  String captionStart = node.getCaption().toUpperCase().charAt(0)
  String nodeCiType = node.getAttributeValue("CmdbCiType")
  String mapEntry = "${captionStart}_${nodeCiType}"

  // Create a CI for each letter corresponding to the first letter of each synced
  //node
  letterCIs["$mapEntry"] = DMA.createCi("$mapEntry")
  letterCIs["$mapEntry"].setCaption("$captionStart")
  letterCIs["$mapEntry"].setType("$nodeCiType")

  // Perform enrichment on the created CI for the letter
  DMA.performEnrichment(letterCIs["$mapEntry"])

  // Automatic STD creation has to be enabled for setServiceTypeDefinitionID to
  //succeed
  letterCIs["$mapEntry"].setServiceTypeDefinitionID("$nodeCiType")
  letterCIs["$mapEntry"].setIsService(true)
  letterCIs["$mapEntry"].addAttribute("CmdbCiType", "${nodeCiType}")
  DMA.logInfo( "Created letter CI for $mapEntry")
}

// Iterate over all nodes to create containment relations to the letter CIs
for (INode node : syncData.getNodes()) {
    String captionStart = node.getCaption().toUpperCase().charAt(0)
    String nodeCiType = node.getAttributeValue("CmdbCiType")
    String mapEntry = "${captionStart}_${nodeCiType}"

    String nodeId = node.getId()

    DMA.logInfo( "Altering relations between CI $nodeId and letter CI
$captionStart")
```

```
    // Create appropriate dependencies
    parentCI = node.getParentCi()

    if ( parentCI != null) {

    String parentCiId = parentCI.getId()

      DMA.logInfo( "Got parent CI $parentCiId")

      // Save original relation type for replacement
      String relationType = node.getParentRelation().getType()

      // Remove original containment association
      if ( parentCI.removeRelation( node.getParentRelation() ) ) {
        DMA.logInfo( "Successfully removed parent CI ($parentCiId) relation from
$nodeId")
      }

      // Assign new relations with the same relation type as before
      letterCIs["$mapEntry"].addRelation(
DMA.createRelation(letterCIs["$mapEntry"], node, "$relationType") )
      parentCI.addRelation (DMA.createRelation(parentCI, letterCIs["$mapEntry"],
"$relationType") )
    }
}
```

Organizing Nodes in Layout Groups

This example script moves all Windows nodes into the `managed/windows` node layout group and all UNIX nodes to the `managed/unix` node layout group.

```
/* (C) Copyright 2008 Hewlett-Packard Development Company, L.P.
 *
 * This sample script moves all Windows nodes into the node layout
 * group 'managed/windows' and all unix nodes to 'managed/unix'.
 */

import com.hp.ov.om.dma.common.data.ci.INode
import com.hp.ov.om.dma.common.data.ci.CiMessageSeverity

String windowsNodes = ""
String unixNodes = ""

// Iterate over all nodes
for (INode node : syncData.getNodes()) {
  if (!node.hasMessages(CiMessageSeverity.SEVERE)){
    if (node.getType() == "nt"){
      // if the CI-Type is nt, add the DnsName to the list of Windows nodes
      if (windowsNodes.length() == 0){
        windowsNodes = node.getDnsName()
      } else {
        windowsNodes += " " + node.getDnsName()
      }
    }
    if (node.getType() == "unix"){
      // if the CI-Type is unix, add the DnsName to the list of UNIX nodes
      if (unixNodes.length() == 0){
        unixNodes = node.getDnsName()
      } else {
        unixNodes += " " + node.getDnsName()
      }
    }
  }
}

if (windowsNodes.length() > 0){
  // move the nodes to the layout group
  def cmd=["/opt/OV/bin/OpC/utils/opcnode", "-move_nodes", "node_list=\"" +
  windowsNodes + "\"", "layout_group=managed/windows"]
  DMA.exec((String[])cmd)
}
if (unixNodes.length() > 0){
  // move the nodes to the layout group
  def cmd=["/opt/OV/bin/OpC/utils/opcnode", "-move_nodes", "node_list=\"" +
  unixNodes + "\"", "layout_group=managed/unix"]
  DMA.exec((String[])cmd)
}
```

13 Testing and Deployment

This chapter contains information on:

Validating XML Configuration Files

You can use the supplied XML schema definitions to validate the correctness of XML configuration files. You can also use the supplied XML schema definition files to make writing new configuration files easier when using an XML editor such as Eclipse.

XML Schema Definition (XSD) is a standard from World Wide Web Consortium (W3C) for describing and validating the contents of XML files. HPOM DMA provides XSD files for all XML configuration files.

For more information, see the XML Schema documentation by W3C available from the **http://www.w3.org/XML/Schema** web site.

HPOM DMA XSD Files

The HPOM DMA schema files are stored in the following directory:

`<InstallDir>/misc/dma/schemas`

They are also available from the following location:

http://<*HPOM DMA Host*>:8081/hpdma/schemas/<*Schema File Name*>

The files are:

bundle.xsd	Validates the `bundle.xml` file in each synchronization package.
containmentrelations.xsd	Validates the `containmentrelations.xml` file.
datadump.xsd	Validates synchronization data files that are created through enabling data dumps or used as input for the enrichment simulator.
mapping.xsd	Validates the following mapping files contained in the synchronization packages: • Service mapping - `servicemapping.xml` • Node mapping - `nodemapping.xml` • Attribute mapping - `attributemapping.xml` • User profile mapping - `usermapping.xml`
nodetypes.xsd	Validates the node type mapping file `nodetypes.xml`.
schedule.xsd	Validates the scheduler configuration file `schedule.xml`.

Validating Files Automatically

Each configuration file is automatically validated against the associated XSD file whenever it is read by HPOM DMA. If a file cannot be validated, an error message is written to the error log that describes the location of the error in the validated file.

Validating Files Manually

Modern XML editors, such as Eclipse, enable you to validate a file against a schema. Eclipse, for example, can validate an XML file against a schema, if the top level element of the document contains a reference to an XSD file. To enable validation, add the following attributes to the top level element of an XML file:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="<path or URL to schema file>"
```

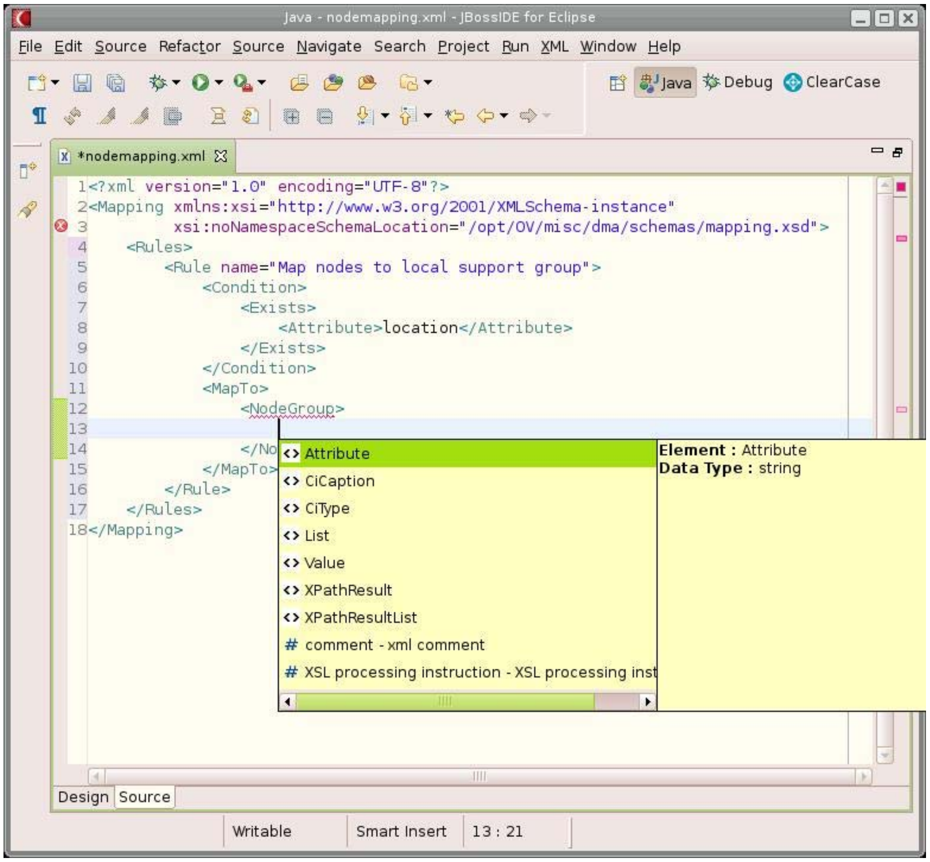
Replace *<path or URL to schema file>* with the respective path or URL to the schema file against which you want to validate. For example, for a mapping rules file add the following on UNIX installations:

```
<?xml version="1.0" encoding="UTF-8"?>>
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://localhost:8081/hpdma/
schemas/mapping.xsd">
...
</Mapping>
```

When you create or edit a mapping file using the HPOM DMA console, the schema definition is added automatically. In this case, you only have to copy and paste the XML code from the console into your editor.

After you have added the reference, the Eclipse editor validates the file and suggests valid elements when pressing **CTRL+SPACE** during editing. See Figure 24 for an example.

Figure 24 Example of Validating a File


Java - nodemapping.xml - JBossIDE for Eclipse
File Edit Source Refactor Source Navigate Search Project Run XML Window Help
Java
Debug
ClearCase
*nodemapping.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="/opt/OV/misc/dma/schemas/mapping.xsd">
    <Rules>
        <Rule name="Map nodes to local support group">
            <Condition>
                <Exists>
                    <Attribute>location</Attribute>
                </Exists>
            </Condition>
            <MapTo>
                <NodeGroup>

                </No
            </MapTo>
        </Rule>
    </Rules>
</Mapping>
```

You may have to reopen the XML file after you have added the XSD reference to the XML file before Eclipse starts to validate it and provides suggestions.

Dumping Synchronization Data

You can use a dump of the synchronization data to:

- Troubleshoot mapping rules to discover incorrect mappings.
- Compare the data sent by the UCMDB and the data changed and added during the enrichment.
- Create a dump file that you can use in the enrichment simulator or to check XPath expressions of rules. See Testing Mapping Rules on page 118.

Creating a Synchronization Data Dump

A synchronization data dump contains the synchronized CIs in XML files using the data format as exposed to the XPath Expression matching in the mapping rules.

There are two separate dumps:

- The first is recorded following CI data normalization.
- The second is recorded following the processing of the mapping rules.

To activate the creation of synchronization data dumps:

1. Open the following file for editing:

   `<SharedDir>/server/conf/dma/DefaultSyncTask.settings`

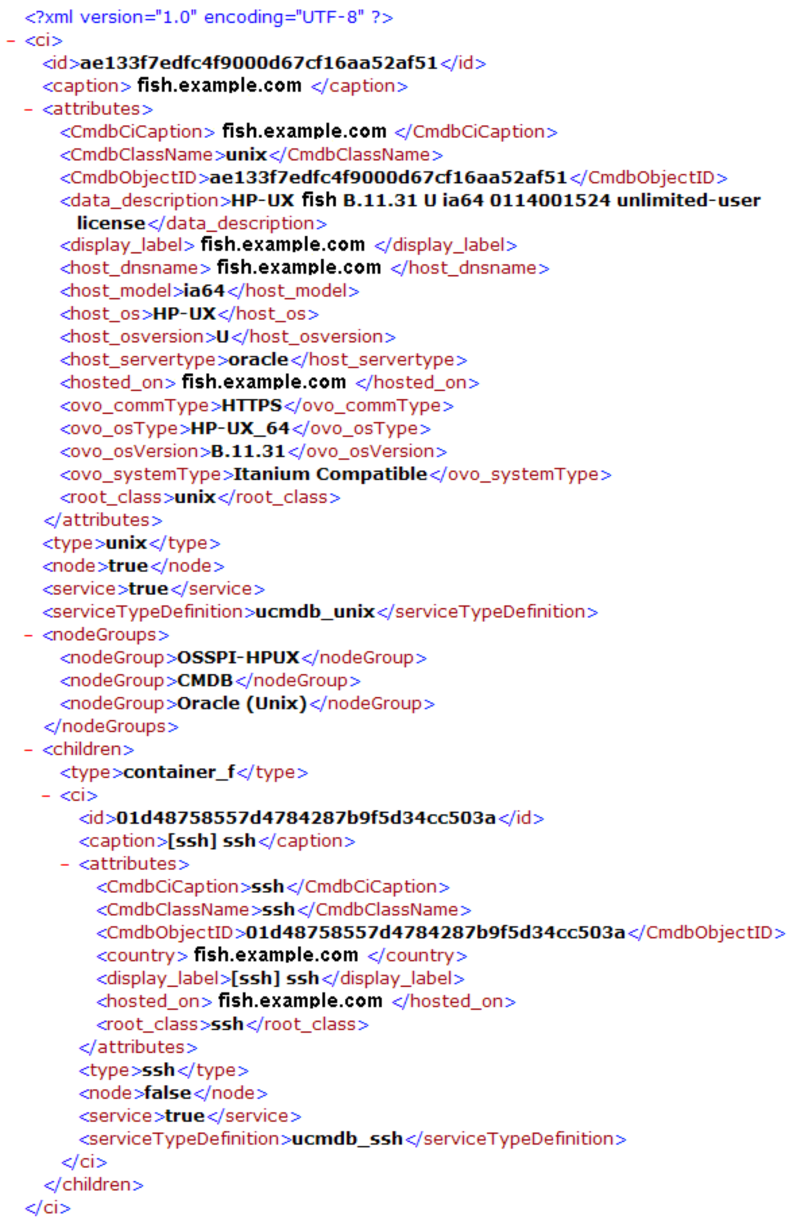
2. Change the following line:

   `sync.dumpData=true`

3. Start a synchronization using the HPOM DMA console.

Figure 25 is a example of a data dump after enrichment has been performed.

Figure 25 Synchronization Data Dump



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <ci>
    <id>ae133f7edfc4f9000d67cf16aa52af51</id>
    <caption> fish.example.com </caption>
  - <attributes>
      <CmdbCiCaption> fish.example.com </CmdbCiCaption>
      <CmdbClassName>unix</CmdbClassName>
      <CmdbObjectID>ae133f7edfc4f9000d67cf16aa52af51</CmdbObjectID>
      <data_description>HP-UX fish B.11.31 U ia64 0114001524 unlimited-user
        license</data_description>
      <display_label> fish.example.com </display_label>
      <host_dnsname> fish.example.com </host_dnsname>
      <host_model>ia64</host_model>
      <host_os>HP-UX</host_os>
      <host_osversion>U</host_osversion>
      <host_servertype>oracle</host_servertype>
      <hosted_on> fish.example.com </hosted_on>
      <ovo_commType>HTTPS</ovo_commType>
      <ovo_osType>HP-UX_64</ovo_osType>
      <ovo_osVersion>B.11.31</ovo_osVersion>
      <ovo_systemType>Itanium Compatible</ovo_systemType>
      <root_class>unix</root_class>
    </attributes>
    <type>unix</type>
    <node>true</node>
    <service>true</service>
    <serviceTypeDefinition>ucmdb_unix</serviceTypeDefinition>
  - <nodeGroups>
      <nodeGroup>OSSPI-HPUX</nodeGroup>
      <nodeGroup>CMDB</nodeGroup>
      <nodeGroup>Oracle (Unix)</nodeGroup>
    </nodeGroups>
  - <children>
      <type>container_f</type>
    - <ci>
        <id>01d48758557d4784287b9f5d34cc503a</id>
        <caption>[ssh] ssh</caption>
      - <attributes>
          <CmdbCiCaption>ssh</CmdbCiCaption>
          <CmdbClassName>ssh</CmdbClassName>
          <CmdbObjectID>01d48758557d4784287b9f5d34cc503a</CmdbObjectID>
          <country> fish.example.com </country>
          <display_label>[ssh] ssh</display_label>
          <hosted_on> fish.example.com </hosted_on>
          <root_class>ssh</root_class>
        </attributes>
        <type>ssh</type>
        <node>false</node>
        <service>true</service>
        <serviceTypeDefinition>ucmdb_ssh</serviceTypeDefinition>
      </ci>
    </children>
  </ci>
```

Viewing a Synchronization Data Dump

To view synchronization data dumps, navigate to the directory:

`<SharedDir>/server/log/dma/`

The directory contains two subdirectories:

- **`normalized`**

  Contains the synchronization data after the CI data structure has been normalized. The data reflects what has been submitted by the UCMDB.

- **`enriched`**

  Contains the synchronization data after the mapping rules have been executed on the normalized data.

Each directory contains one file per CI that is the root of a sub-hierarchy placed under the top-level root service. This is the root service configured on the Configuration page (default is CMDB).

The file names follow the schema:

`<rootCiCaption>_<rootCiId>.xml`

Validating Mapping Rules

To validate mapping rules, complete the following steps:

- **Compare File Differences**

  Using a file comparison tool of your choice you can easily see what has been changed during enrichment.

- **Validate XPath Expressions**

  You can validate XPath Expressions that are used in mapping rules by loading the normalized synchronization data dumps into an XML editor that supports XPath queries.

➤ An XML document must have a single root element (`<ci>`) in the data dumps. When running XPath queries in the mapping rules, this root element does not exist. For testing with dump files, when you create absolute expressions, prepend the expression `/ci` to your test expression.

Testing Mapping Rules

You can test mapping rules using the HPOM DMA Enrichment Simulator command-line tool.

Enrichment Simulator

The enrichment simulator enables you to execute a "dry run" of the enrichment process. This applies the mapping rules for service mapping, node mapping, and attribute mapping. Data is acquired from the UCMDB or BAC, or read from an XML file that has been written manually or that has been created through a data dump during a synchronization. The result is dumped to an XML file and placed in the specified output directory.

For more information on data dumps, see Creating a Synchronization Data Dump on page 115.

The start script is located in the following directory:

- **Windows**

  `<InstallDir>\support\dmaenrichsim.bat`

- **UNIX and Linux**

  `<InstallDir>/support/dmaenrichsim.sh`

dmaenrichsim Command

The `dmaenrichsim` command incorporates the following options:

`-c,-cmdb`	Reads the input data from the web service endpoint currently configured.
`-h,-help`	Shows a detailed help message.
`-i,-input` ***`<input file>`***	Reads the input data from the specified file. Cannot be applied in combination with the `-c` option.
`-o,-output` ***`<output dir>`***	Writes the resulting XML output files into the specified directory.
`-version`	Prints the version.

Examples

```
dmaenrichsim.sh -i /tmp/input.xml -o /tmp/output
```

Reads the input data from the `/tmp/input.xml` file and writes the resulting XML output files into the `/tmp/output` directory.

```
dmaenrichsim.sh -c -o /tmp/output
```

Reads the input data from the web service endpoint currently configured and writes the resulting XML output files into the `/tmp/output` directory.

Writing Rules

This section contains a set of guidelines for writing rules.

Simplifying Rule Development

You can ease the writing of rules by selecting an XML editor that can validate and suggest elements according to an XML schema. See Validating XML Configuration Files on page 111 for more information.

After you create rules, you can test these rules easily using the Enrichment Simulator. See Testing Mapping Rules on page 118 for more information.

Avoiding Complex XPath Queries

Avoid complex XPath queries, especially in general conditions, where such queries must be applied to every CI. If you cannot avoid a complex XPath query, try to narrow the condition using operators such as `CiType`, combined using the `And` operator. Make sure that the simpler, non-XPath conditions are checked first (hint: `And` is an exclusive operator).

Matching Against Existing Attributes Only

Accessing attributes that do not exist for all CI types is very performance intensive in combination with a relative expression depending on the complexity of the CI hierarchy.

Avoiding Broad XPath Expressions

Certain complex XPath expressions can result in excessive processing loads. For example, XPath expressions that include the following characteristics:

- Apply to multiple nodes, such as expressions that contain `//` or `descendants:*/`
- Do not match nodes or match only on nodes that are very distant from the current node

The same applies to the `XPathResultList` operator that returns all matched values. The time required for such operations grows approximately quadratically with the size of a hierarchy. Avoid such expressions where possible.

When using the descendants operator, do not use the star symbol `(*)` as node test, but specify the name of the node of interest. For example, do not use `descendants:*/caption` but use `descendants:ci/caption`.

If you cannot avoid such an XPath expression within a condition, try to limit its execution by using the exclusive `And` operator and perform simple tests before the `XPathResult` operand is being used. For example, you could first check for the CI type.

Exporting TQL Queries

Exporting TQL queries, requires you to:

- Create a new package containing the existing UCMDB resources (views and enrichments), for example, from a development environment.
- Copy the package to another BAC or UCMDB installation, for example, to a production system.
- Upload and deploy the TQL query package.

Creating a TQL Query Package

To create a TQL query package containing queries and enrichments, follow these steps:

1 Open the Package Manager:

- **UCMDB 7 and 8:**

  Admin → **Settings** → **Package Manager**

- **UCMDB 9:**

  Administration → **Package Manager**

- **BAC:**

  Admin → **Universal CMDB** → **Settings** → **Package Manager**

2 Click **New**.

The Create Custom Package Wizard opens.

3 Enter a name and description for the new package and click **Next**.

4 Expand **Query** → **TQLs** → **Enrichment** and check `hpdma`.

5 Expand **Query** → **TQLs** → **View** and check `hpdma`.

6 Expand **Query** → **Views** and check `hpdma`.

7 Expand **Query** → **Enrichments** and check `hpdma`.

8 Click **Next**.

A summary of your selections is displayed.

9 Click **Finish**.

Deploying and Registering UCMDB Packages

To upload and deploy HPOM DMA packages to your UCMDB or BAC installation, complete the following steps:

1 Copy the HPOM DMA packages into a temporary directory on the BAC or UCMDB system.

The HPOM DMA packages reside in the following directory on the HPOM DMA system:

Version 7.5x

`<InstallDir>/misc/dma/ucmdb/7`

Version 8.0x

`<InstallDir>/misc/dma/ucmdb/8`

Version 9.0x

`<InstallDir>/misc/dma/ucmdb/9`

2 Open the Package Manager:

- *UCMDB 7 and 8*

  Select **Admin** → **Settings** → **Package Manager.**

- *UCMDB 9*

  Select **Administration** → **Package Manager.**

- *BAC*

  Select **Admin** → **Universal CMDB** → **Settings** → **Package Manager.**

3 Click **Deploy Packages to Server (from local disk)**.

4 In the Deploy Packages to Server dialog box, click **Add**.

5 Browse to the temporary directory on the BAC or UCMDB system where you stored the HPOM DMA packages.

6 Select the `hpdmacore.zip` file and click **Open**.

This package must be deployed before any of the other HPOM DMA packages.

7 In the Deploy Packages to Server dialog box, click **OK**.

8 Deploy any of the following packages that you want to use:

- `hpdmaos.zip`

  Used for operating systems.

- `hpdmadb.zip`

  Used for databases.

- `hpdmasamples.zip`

  Used for the My Company Sample synchronization package.

- `hpdmaEUM.zip`

  HP BAC only: Used for End User Monitors

For each required package, repeat step 3 to step 7.

A Mapping Syntax

Mapping is the mechanism used to map CIs from the UCMDB to services, attributes, or nodes within HPOM. The file format, mapping syntax, and XPath query language used to navigate through the CI data structure is described in the following sections:

- Common Mapping File Format on page 125
- Mapping File Syntax on page 126
- XPath Navigation on page 138

Common Mapping File Format

➤ The rule name must be unique for all rules in the current file.

This example illustrates the common parts of the mapping file:

```
<?xml version="1.0" encoding="utf-8"?>
<Mapping>
    <Rules>
        <Rule name="Apache Server">
            <Condition>
                <!-- ... Boolean operators ... -->
            </Condition>
            <MapTo>
                <!-- ... Target Mappings ... -->
            </MapTo>
        </Rule>
        <!-- ... More Rules ... -->
    </Rules>
</Mapping>
```

The components of the mapping files are described in Mapping File Syntax on page 126.

Mapping File Syntax

The following subsections describe the valid syntax used in HPOM DMA mapping files.

Rule Conditions

The `<Condition>` element of a rule contains a Boolean operator that specifies how the individual conditions relate to each other.

Each operator can implement an operation against operands. For example, attribute `hosted_on` has a value ending with `.europe.example.com`. (attribute `hosted_on` and `.europe.example.com` are operands) or an operation against one or a set of other nested operators like `<And>`, `<Or>` or `<Not>`.

Operator Elements

True

```
<True/>
```

This operator always returns true when all nested operators return true. It is useful for declaring default (fall-back) rules. In a mapping engine that is using the early-out mode, make sure that this operator is only used at the end of the synchronization package with the lowest priority.

False

```
<False/>
```

Always returns false. You can use the `False` element to temporarily disable rules.

And

```
<And>
    <!-- Operator -->
    <!-- Operator -->
    [... more operators ...]
</And>
```

Returns true when all nested operators return true.

The `<And>` operator is exclusive. This means that if the result of the first operator is false, the next operator is not evaluated. You should use this operator to implement rules with higher performance by placing the simplest condition first and the most complex condition at the end.

Or

```
<Or>
    <!-- Operator -->
    <!-- Operator -->
    [... more operators ...]
</Or>
```

Returns true if at least one of the operators returns true.

Not

```
<Not>
    <!-- Operator -->
</Not>
```

Returns true if the operator does not return true.

The `<Not>` operator is exclusive. This means that evaluation stops as soon as a sub-operator returns true.

Exists

```
<Exists>
    <!-- Operand -->
<Exists>
```

The value of the operand may not be null.

Is Node

```
<IsNode/>
```

True if the CI is imported as a node, which is the case if the CI type is listed in the `nodetypes.xml` file. For further information, see Node Types on page 69

Equals

```
<Equals>
    <!-- Operand -->
    <!-- Operand -->
    <!-- ... -->
</Equals>

<Equals ignoreCase="[true|false]">
    <!-- Operand -->
    <!-- Operand -->
    <!-- ... -->
</Equals>
```

The values of the operands must be equal. If there are more than two operands, all operands must be equal to each other. Using the optional attribute `ignoreCase`, you can also compare the string values of the operands independent of capitalization. By default the equals operator does not ignore case.

Starts With

```
<StartsWith>
    <!-- Operand -->
    <!-- Operand -->
</StartsWith>
```

The string value of the first operand must start with the value of the second operand.

Ends With

```
<EndsWith>
    <!-- Operand -->
    <!-- Operand -->
</EndsWith>
```

The string value of the first operand must end with the value of the second operand.

Contains

```
<Contains>
    <!-- Operand -->
    <!-- Operand -->
<Contains>
```

The value returned by the first operand must contain the value of the second operand. If the operand's return type is a list, the list must contain at least one element that is equal to the second operand. If the operand's return type is a string, the value of the second operand must be a substring of the first operand.

Matches

```
<Matches>
    <!-- Operand -->
    <!-- Operand -->
</Matches>
```

The string value of the first operand must match the regular expression of the second operand.

Example:

```
<Matches>
    <Attribute>host_dnsname</Attribute>
    <Value>.*\.example\.com</Value>
</Matches>
```

For more information on applicable regular expressions, see:

http://download.oracle.com/javase/1.5.0/docs/api/java/util/regex/Pattern.html

http://download.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html

Operand Elements

CI Type

```
<CiType/>
```

Return type: String

Returns the CI type string if the CI exists in the UCMDB or BAC.

➤ This is not the display label of the CI Type.

CI Caption

The CI Caption has the same value as the attribute `display_label`.

```
<CiCaption/>
```

Return type: String

Returns the caption string of the CI in the UCMDB or BAC.

Attribute

```
<Attribute>[Name]</Attribute>
```

Return type: String

Returns the value of the UCMDB CI attribute with the given name.

This is not the display label of the CI Type.

Service Name

```
<ServiceName/>
```

Return type: String

Returns the service ID string of the CI in the UCMDB or BAC.

Replace

```
<Replace [regExp="true|false"]>
    <In>
        <!-- 1st. Operand -->
    </In>
    <For>
        <!-- 2nd. Operand -->
    </For>
    <By>
        <!-- 3rd. Operand -->
    </By>
</Replace>
```

Return type: String

Replaces the sub-strings in the return value of the first operand for all occurrences of the return value of the second operator by the return value of the third operand. For example, to replace all occurrences of a backslash in the CI caption by an underscore, you must declare the following:

```
<Replace>
    <In>
        <CiCaption/>
    </In>
    <For>
        <Value>\</Value>
    </For>
    <By>
        <Value>_</Value>
    </By>
</Replace>
```

Optionally, you can use regular expressions for the second operand. You can also use back references in the third operand.

For more information on applicable regular expressions, see:

http://download.oracle.com/javase/1.5.0/docs/api/java/util/regex/Pattern.html

http://download.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html

This example uses regular expressions to extract part of a domain name:

```
<Replace regExp="true">
    <In>
        <Attribute>host_dnsname</Attribute>
    </In>
    <For>
        <Value>^[^.]*\.([^.]*).*</Value>
    </For>
    <By>
        <Value>$1</Value>
    </By>
</Replace>
```

If the attribute `host_dnsname` contains the value `server.rio.example.com`, the result of the `Replace` operand is `rio`.

XPath Result

```
<XPathResult>[XPath]</XPathResult>
```

Return type: String

Returns the value of the XPath expression, which enables you to access data of any CI that is contained in the same hierarchy. The XPath expression must select a string value, if there are multiple matches an arbitrary element is returned.

For more information on XPath, see XPath Navigation on page 138.

XPath Result List

```
<XPathResultList>[XPath]</XPathResultList>
```

Return type: List

Returns a list of all matched values.

For more information on XPath, see XPath Navigation on page 138.

Value

```
<Value>[String]</Value>
```

Return type: String

Return the constant value.

List

```
<List>
    <!--Operand-->
    <!--Operand-->
    <!--...-->
</List>
```

Return type: List

The list operand is designed for use with operators that accept lists as input parameters, such as the `contains` operator. The list operand contains a list of other operands, the values of which are to be added to the returned list.

Condition Examples

Check if the type of the current CI is `unix`.

```
<Condition>
    <Equals>
        <Type/>
        <Value>unix</Value>
    </Equals>
</Condition>
```

Check if the CI is related to a node that is located in the `europe.example.com` domain.

```
<Condition>
    <EndsWith>
        <XPathResult>//.[node='true']/attributes/
            host_dnsName<XPathResult>
        <Value>.europe.example.com</Value>
    </EndsWith>
</Condition>
```

Mapping Elements

`<MapTo>` defines the mappings. Each concrete implementation of an engine adds its own XML elements for its individual mappings here.

Service Mapping Syntax

Valid element for the `<MapTo>` section:

```
<STD>
    [Operand]
    ...
</STD>
```

Maps the CI to the specified service type definition that is the concatenated string of the results of the operators. There must not be more than one `<STD>` element in the `<MapTo>` section of a `servicemapping.xml` file. For more information about operands, see Operand Elements on page 129.

Override STD Properties

You can override certain attributes of an STD, as described in Service Type Definition Files on page 59, for selected CIs:

Override a Calculation Rule for a CI:

```
<CalculationRule>
    [Operand]
    ...
</CalculationRule>
```

Override a Message Propagation Rule for a CI:

```
<MessagePropagationRule>
    [Operand]
    ...
</MessagePropagationRule>
```

Override a Parent Propagation Rule for a CI:

```
<ParentPropagationRule>
       [Operand]
    ...
</ParentPropagationRule>
```

Override an Icon for a CI:

```
<Icon>
    [Operand]
    ...
</Icon>
```

Override the Message Weight for a CI:

```
<MessageWeight>[Double]</MessageWeight>
```

Override the Parent Weight for a CI:

```
<ParentWeight>[Double]</ParentWeight>
```

Creating Dependencies from External CIs to Internal CIs

External CI dependency creation associates service CIs to services that are already present in Service Navigator and which are not part of the HPOM DMA synchronization.

```
<DependencyFromExternalCi>
  <ExternalCiId>
       [Operand]
       ...
  </ExternalCiId>
  <DependencyType>
       [Operand]
       ...
  </DependencyType>
  <PropagationRuleName>
       [Operand]
       ...
  </PropagationRuleName>
  <PropagationWeight>[Double]</PropagationWeight>
</DependencyFromExternalCi>
```

Creating Dependencies from Internal CIs to External CIs

Internal CI dependency creation associates service CIs to services that are already present in the HPOM DMA synchronization, and that are not part of Service Navigator.

```
<DependencyToExternalCi>
  <ExternalCiId>
       [Operand]
       ...
  </ExternalCiId>
  <DependencyType>
       [Operand]
       ...
  </DependencyType>
  [<PropagationRuleName>
       [Operand]
       ...
  </PropagationRuleName>
  <PropagationWeight>[Double]</PropagationWeight>]
</DependencyToExternalCi>
```

Node Mapping Syntax

Node CIs with attributes that match a particular node mapping are assigned to the associated node group. For each rule that matches, the node is also added to all node groups defined in the rule.

Map the node with the matched conditions to the specified node group, which is the concatenated value of the values of the operands.

Valid element for the `<MapTo>` section:

```
<NodeGroup>
  [Operand]
  ...
</NodeGroup>
```

Multiple node groups are supported and the node is added to each node group.

For more information about operands, see Operand Elements on page 129.

Attribute Mapping Syntax

Attribute mappings are specified using the following syntax:

```
<Attribute>
    <Name>[Attribute Name]</Name>
    <SetValue>
        [Operands]
    </SetValue>
</Attribute>
```

Sets the value of the attribute of the given name to the returned value of the given operand. If more than one operand is given, the values are concatenated. For more information about operands, see Operand Elements on page 129.

User Profile Mapping Syntax

User profile mappings are specified using the following syntax:

```
<UserProfile>
    [Operands]
    ...
</UserProfile>
```

Assigns the current CI to the user profile that is the returned value of the given operand. If more than one operand is given, the values are concatenated. You may simultaneously assign a CI to multiple user profiles by specifying more than one `UserProfile` mapping action in the same way as specifying the NodeGroup mapping action. For more information about operands, see Operand Elements on page 129.

XPath Navigation

XPath is used in the mapping engines to navigate through the CI data structure.

If you are not familiar with the XPath query language, an XPath tutorial can be found at the following web site:

http://www.w3schools.com/xpath/

Data Structure

The data structure that is exposed to the XPath expression matching used in mapping rules is shown in Figure 26.

CI Data Structure

Attributes	Contains a map of all original UCMDB CI attributes. The key of this map is the name of the UCMDB CI attribute that references the UCMDB value of the UCMDB CI attribute.
Caption	Represents the name of the CI to be displayed in Service Navigator. Caption has the same value as the UCMDB CI attribute `display_label`.
Children	References a list of relations to CIs that have a containment relationship from the current CI to other CIs. Using this field, you can create complex XPath queries to retrieve values of children as well as parents using the ". ." XPath selector.
Dependencies	References a list of relations to dependent CIs. Similar to `Children`. However, referenced objects are contained in a different hierarchy.
id	Unique ID of a CI.
Node	Boolean value that indicates, whether this CI is imported as a service only or also as a node into the HPOM Node Bank.
Type	Contains the type ID string of a CI.

This is not the display label of the CI Type.

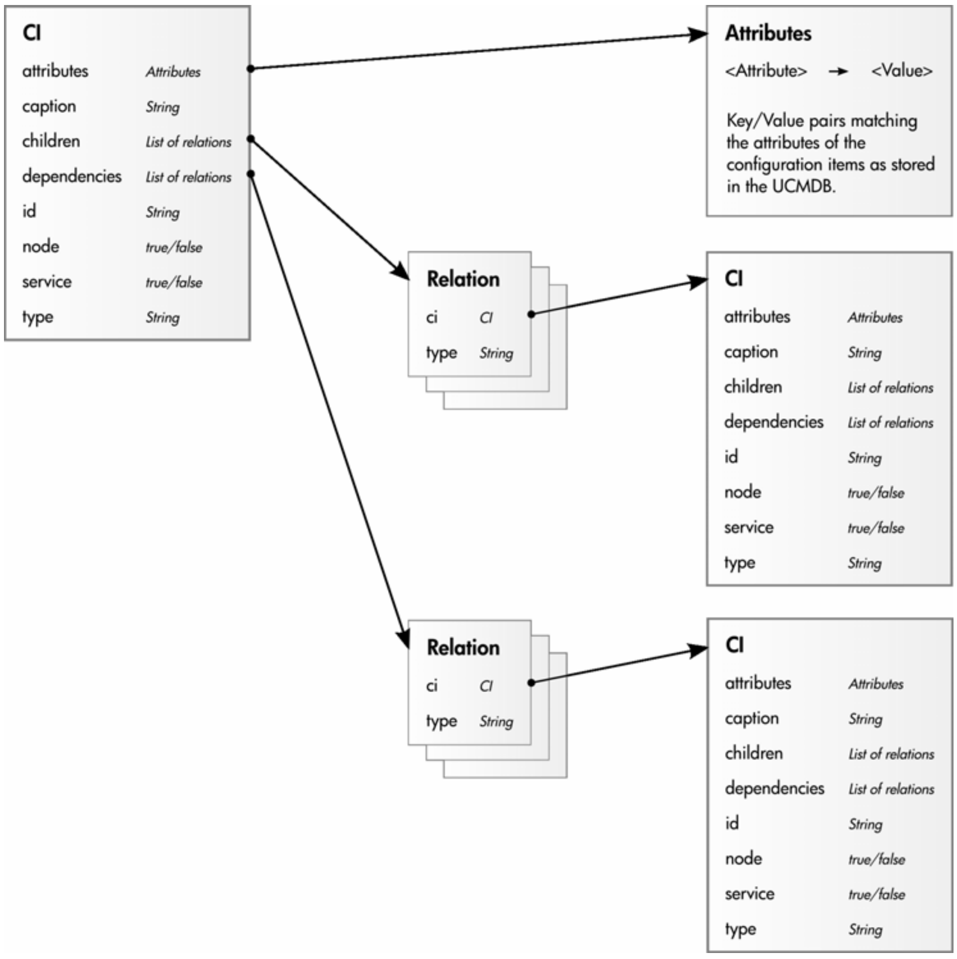
Relationship Data Structure

CI	Contains the reference to the CI to which the current CI is related.
Type	Relationship type as stored in the UCMDB.

➤ This is not the display label of the CI Type.

Figure 26 illustrates the data structure exposed to the navigation.

Figure 26 Data Structure Exposed to the Navigation

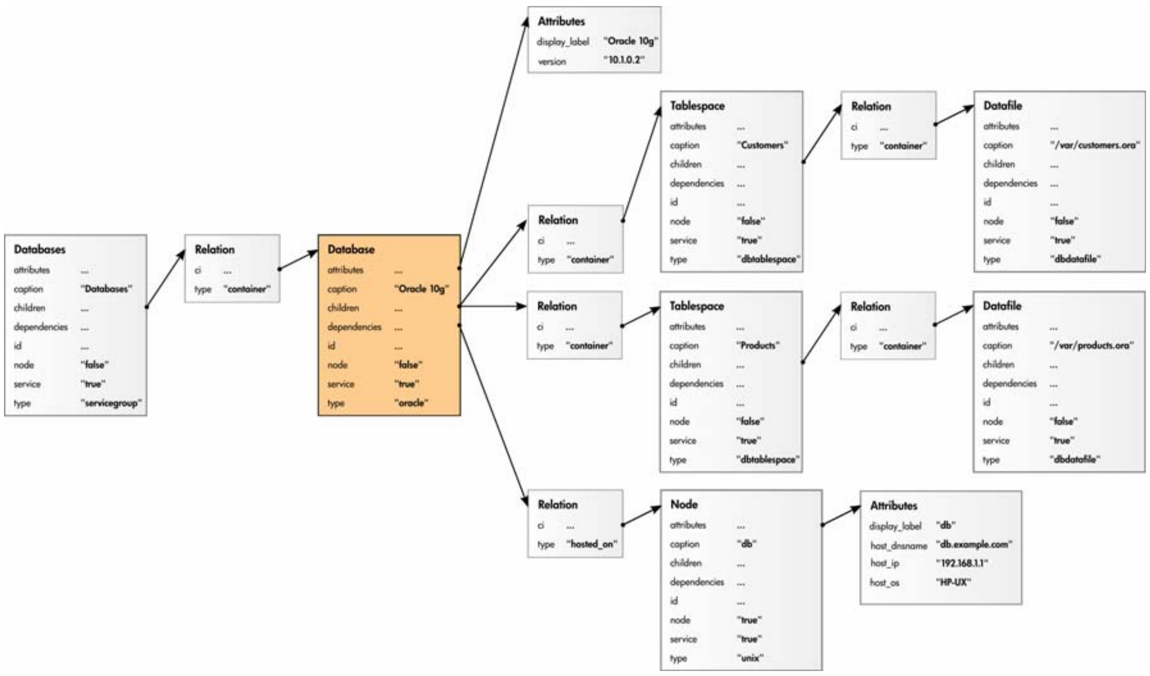


Example of an XPath-Navigated Data Structure

An example of an XPath-navigated data structure is shown in Figure 27. The host is a UNIX system that has an Oracle application running on the HP-UX operating system. The starting point or context for the navigation is the CI that represents the Oracle application (orange background).

Figure 27 illustrates some XPath examples.

Figure 27 XPath Examples



XPath Expressions and Example Values

The following table lists typical XPath expressions and provides an example for each expression.

`caption`	**`Oracle 10g`**
`./caption`	**`Oracle 10g`**
`/caption`	**`Databases`**
`../caption`	**`server.example.com`**
`../type`	**`container`**
`../../type`	**`servicegroup`**
`/type`	**`servicegroup`**
`//.[type='dbtablespace']/caption`	**`Customers`** **`Products`**
`//dependencies[type='hosted_on']/ci/caption`	**`db`**
`//.[type='unix']/attributes/host_dnsname`	**`db.example.com`**
`//host_dnsname`	**`db.example.com`**

`//children/ci/caption`	**`Oracle 10g`**
	`Customers`
	`/var/customers.ora`
	`Products`
	`/var/products.ora`

`//dependencies/ci/caption`	**`db`**

If the Xpath expression selects a node below the starting database node, the "`..`" reads back one step. The following expression reads down to the node `db` and then links back to the starting database node.

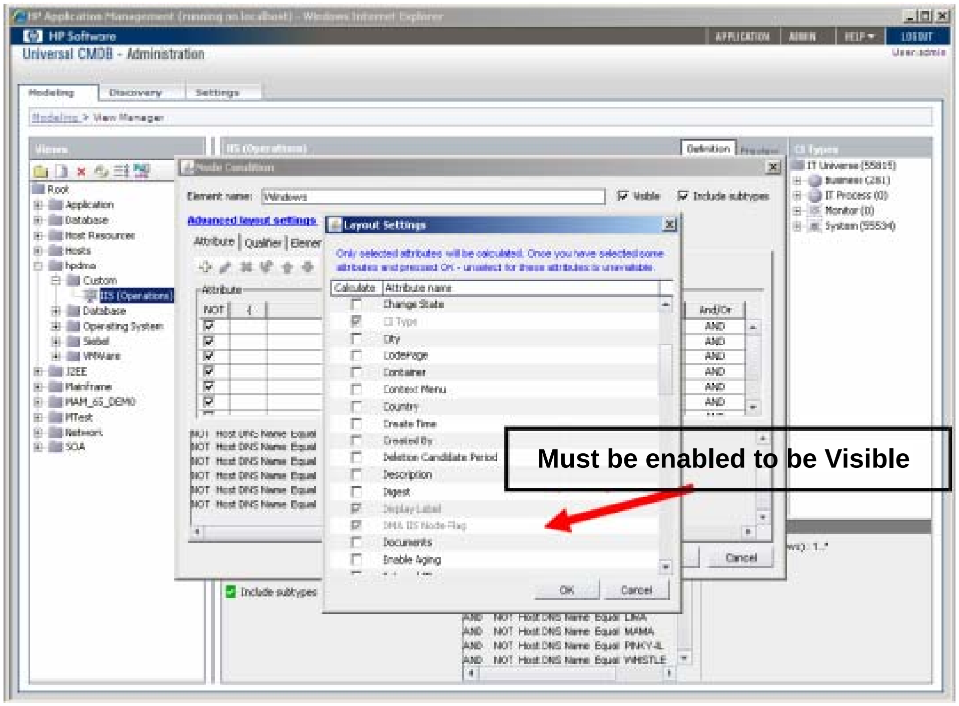
```
//dependencies[type='hosted_on']/ci/../..
```

However, if the node `db` is the starting node, the expression `../..` follows the containment links of the node `db`, which is not the dependency relation that is shown in this example. The result depends on the parent container of the node, which is a different hierarchy.

B UCMDB Attributes

When you are developing TQL queries, make sure that certain necessary attributes are set in the UCMDB and enabled on the Advanced Layout Settings page of the Node Properties.

Figure 28 Enable Attributes in Layout Settings



Layout Settings

Only selected attributes will be calculated. Once you have selected some attributes and pressed OK - unselect for these attributes is unavailable.

Calculate	Attribute name
☐	Change State
✓	CI Type
☐	City
☐	CodePage
☐	Container
☐	Context Menu
☐	Country
☐	Create Time
☐	Created By
☐	Deletion Candidate Period
☐	Description
☐	Digest
✓	Display Label
✓	DMA IIS Node Flag
☐	Documents
☐	Enable Aging

Required Attributes

The following attributes are required for proper operation of DMA, when the mentioned synchronization packages are selected. Note that the Default synchronization package is always enabled.

Table 2 UCMDB Attributes Required by HPOM DMA

CI Types	Attribute Name	Required For	Synchronization Package
All	`display_label`	Setting caption in Service Navigator	Default
All node types	`host_dnsname` [a]	Setting hosted_on attribute for Smart Message Mapping	Default
`database`	`database_dbtype`	Determining database vendor	Informix
All node types	`host_servertype`	Determining the database running on a node	Informix, Oracle, MSSQLServer, Sybase
`sqlserver`	`database_dbversion`	Determining the version of the SQL Server	MSSQLServer
All node types	`host_os`	OS type detection	UNIX operating systems, Windows operating systems

a. With BAC or UCMDB 8.0x and higher, the `host_dnsname` attribute is deprecated. Because HPOM DMA 8.20 requires this attribute, the script `preMapping.groovy` sets the `host_dnsname` attribute based on other attributes.

Recommended Attributes

DMA can work without the following attributes. However there may be drawbacks such as a decrease of synchronization performance. Note that the Default synchronization package is always enabled.

Table 3 UCMDB Attributes Recommended for HPOM DMA

CI Types	Attribute Name	Required For	Synchronization Package
All node types	`host_os`	OS type detection	Default
All node types	`host_osversion`	OS type detection	Default
All node types	`host_model`	OS type detection	Default
All node types	`data_description`	OS type detection	Default

Index