HP Operations Manager Dependency Mapping Automation

For Windows® and UNIX® Operating Systems

Software Version: 8.00

Developer's Guide

Manufacturing Part Number: B7491-90095
Document Release Date: 29 November 2007
Software Release Date: November 2007

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

Trademark Notices

Intel® and Itanium® are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java™ and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

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

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

UNIX® is a registered trademark of The Open Group.

Documentation Updates

This 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

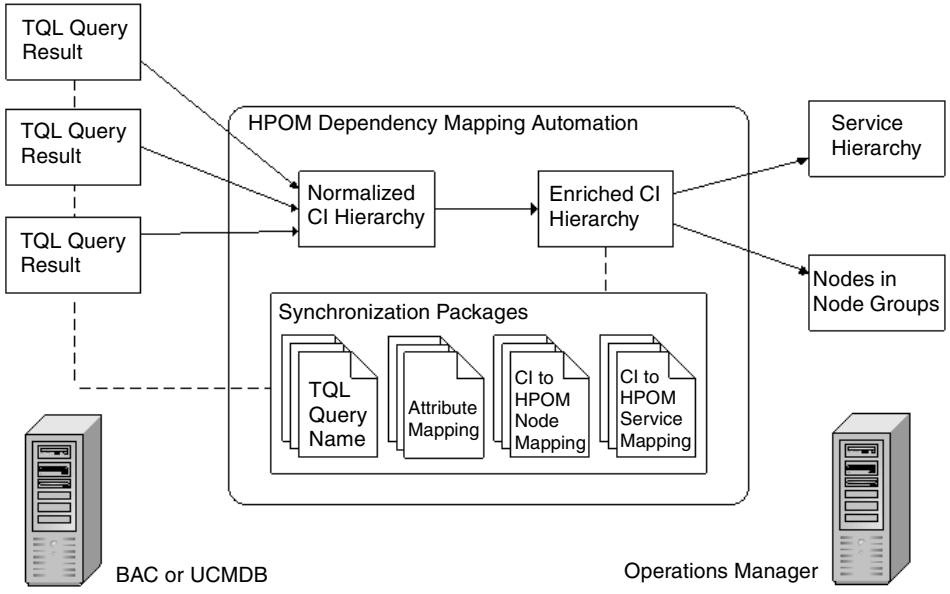
Contents

1 Introduction

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

With the help of this guide, you can learn how to create your own synchronization packages to get your own Universal CMDB (UCMDB) data reflected in HPOM as required. Creating your own synchronization packages allows you to discover your environment as maintained by the UCMDB and 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



Installation Locations

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

- UNIX
- Windows

Locations on UNIX

- ***<InstallDir>***

  `/opt/OV/`

- ***<DataDir>***

  `/var/opt/OV/`

- ***<SharedDir>***

  `/var/opt/OV/shared/server`

Locations on Windows

- ***<InstallDir>***

  Default:
  `C:\Program Files\HP\HP BTO Software`

- ***<DataDir>***

  Default:
  `C:\Documents and Settings\All Users\Application Data\HP\HP BTO Software`

- ***<SharedDir>***

  Default:
  `C:\Documents and Settings\All Users\Application Data\HP\HP BTO Software\shared\server`

Components Required for a New Integration

To integrate each type of application, such as web servers, you need the following:

- **TQL Query**

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

  For information on how to create TQL queries for the UCMDB or BAC, refer to the documentation available with these products. A basic overview is given in Chapter 2, TQL Queries.

- **Synchronization Package**

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

  For details, see Chapter 5, Synchronization Packages.

- **Service Type Definitions**

  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 details, see Chapter 3, Service Type Definition Files.

Synchronization Package Locations

Synchronization packages must be located in a specific location in the file system:

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

Each of these synchronization package directories must contain a `bundle.xml` file. The other files, `servicemapping.xml`, `nodemapping.xml`, `attributemapping.xml`, are optional. The default synchronization package specifies the default node group `CMDB` and the default service type definitions.

```
<SharedDir>/conf/dma/sync-packages/default
      attributemapping.xml
      bundle.xml
      servicemapping.xml

<SharedDir>/conf/dma/sync-packages/<SyncPackageName>
      attributemapping.xml
      bundle.xml
      nodemapping.xml
      servicemapping.xml
```

2 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.

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.

Select **Node Condition** → **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.

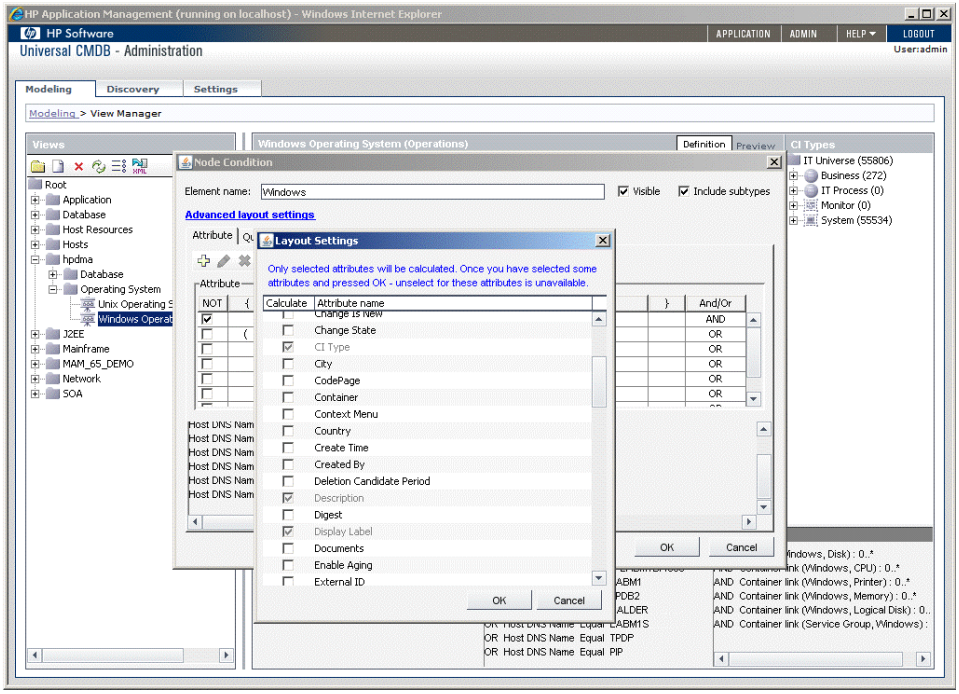
Key attributes must not include the label or the CMDB ID.

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

For example, in the standard attribute mapping, the label is mapped to `[CPU] CPU3` and does not have any effect on message mapping.

In the TQL query, you can choose the attributes of the CI instances that are exposed in the web service. An example is the CI type `disk` where the key attribute is the `name`.

Figure 2 Selecting Attributes Exposed by the UCMDB Web Service



Setting Up a TQL Query

To set up a TQL query, follow these steps:

In the UCMDB Development Environment:

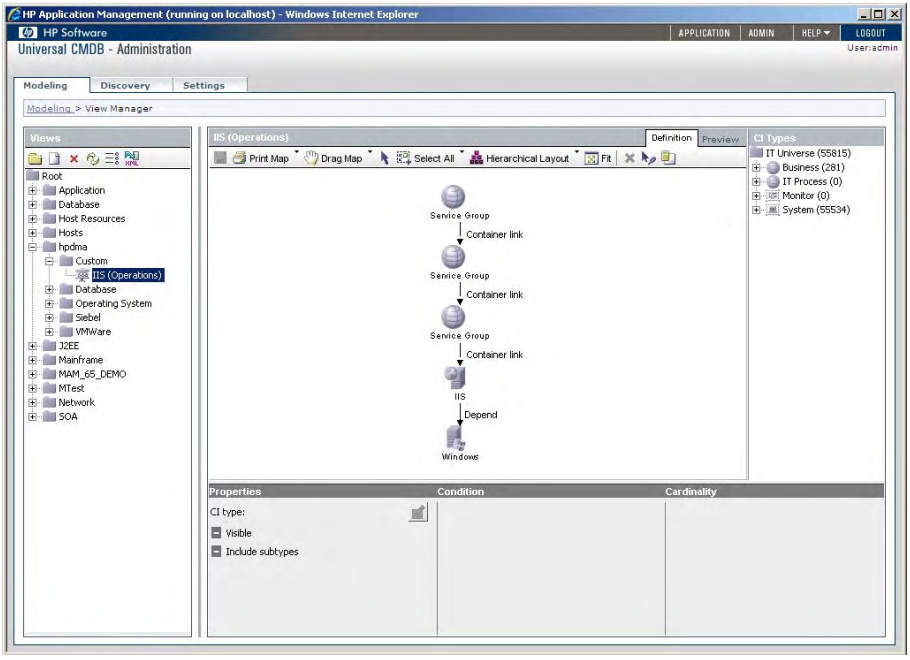
1 Create a view in the View Manager. Activate essential attributes, such as CI Type and the ID, to be able to launch tools and applications.

> If an object is found by more than one multiple 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 3 is an example of a view.

3 Create a package in the Package Manager and export it.

Figure 3 Example of a View

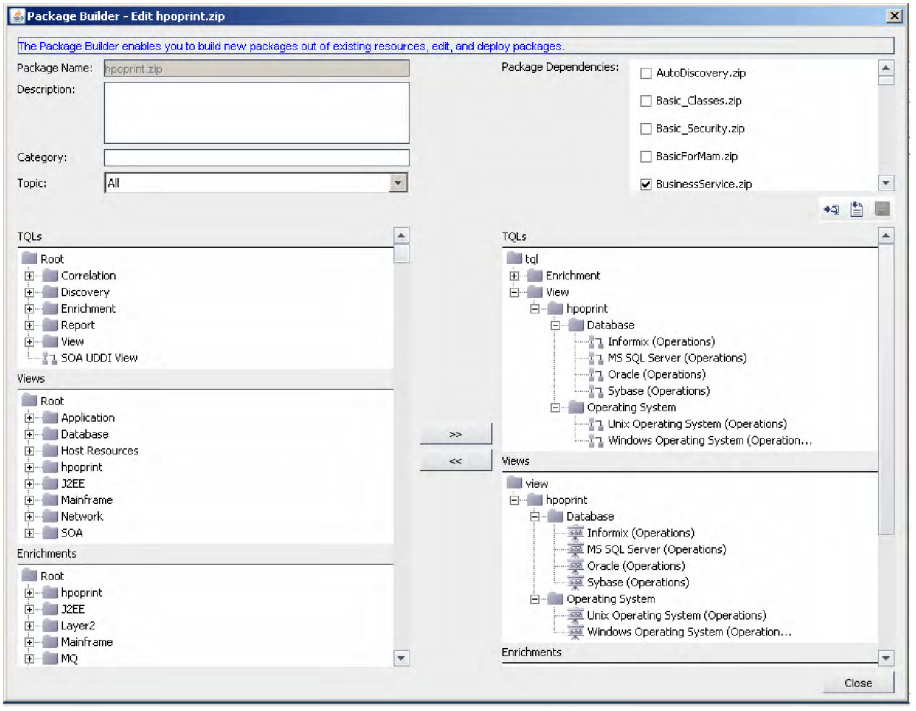


4 Use the Preview tab on the top of the view area to preview the result map selected by the view TQL

In the UCMDB Production Environment:

5 Import and deploy package in Package Manager. See Figure 4 for an example.

Figure 4 Importing and Deploying Packages



3 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. Propagation rules are also assigned to associations between services based on the association type (Composition or Dependency) and the service types of parent and child 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.

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

STDs in HPOM for Windows

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

- **Propagation Rules**

  This section 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 STD Example on Windows on page 18.

- **Calculation Rules**

  This section 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 STD Example on Windows on page 18.

- **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. Changing and uploading a service type definition is the only 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.

  This is the section titled `instance of OV_ServiceTypeDefinition` in STD Example on Windows on page 18.

  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 STD Example on Windows on page 18.

STD Example 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";
};
```

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")

STDs in HPOM for UNIX

Propagation and Calculation rules referenced in a service type definition file must exist in the HPOM for UNIX service engine repository and can be created/updated using the `opcservice` tool. For information on the `opcservice` tool, refer to the *HPOM for UNIX Service Navigator Concepts and Configuration Guide*.

A service type definition file for HPOM for UNIX 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 only 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 for UNIX 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 which 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>
..
```

Service type definitions are specified in service type definition files, which must be in XML format for UNIX. The XML files have to be of document type `Services`. For information on the documentation type `Services`, refer to the *HPOM for UNIX Service Navigator Concepts and Configuration Guide*.

STD Example on UNIX

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

```
<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

Service type definitions are uploaded into HPOM for UNIX 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 service type definitions and their associations.

Example:

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

`-a <filename>` Adds service type definitions and their associations. Requires an additional parameter containing the file name.

Example:

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

`-o` Overwrite existing entries.

Example:

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

`-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 <definition>
```

4 Node Type List

The node type list defines which UCMDB CI types are imported as nodes into the HPOM node groups. For each rule that matches, the node is also added to all the node groups defined in the rule.

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

The node type list configuration is specified in the following file:

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

Format:

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

5 Synchronization Packages

A synchronization package is a set of mapping rules to transform CIs into HPOM services and nodes and are used to synchronize specified services and nodes in HPOM with data from the UCMDB. They can be created and delivered by SPI or application developers, and deployed on an HPOM DMA installation.

A synchronization package includes the following components:

- Synchronization Package (`bundle.xml`) file to define the synchronization package
- Service Mappings (`servicemapping.xml`) optional
- Node Mappings (`nodemapping.xml`) optional
- Attribute Mapping (`attributemapping.xml`) optional

HPOM DMA includes the mappings 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 (bundle.xml) 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`.

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>
```

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 6, 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 7, Node Mapping.

Attribute Mapping

Attribute mapping allows 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 8, Attribute Mapping.

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>`/conf/dma/sync-packages/`*<SyncPackageName>*

The default location is:

- **Windows**

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

- **UNIX**

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

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

6 Service Mapping

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 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 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 3, Service Type Definition Files.

If you map to service type definitions and definitions for associations, they must exist in HPOM.

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` of the default bundle and can be changed there.

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. The Example of a servicemapping.xml File on page 33 illustrates how this service mapping is specified.

Service Mappings are executed according to the priority of the bundle and the position within the rule declarations. A rule that is contained in a higher prioritized bundle and is above other rules in the same mapping file has precedence over these lower-placed rules. If a rule has matched for a certain CI, lower prioritized rules are not executed.

A UCMDB CI or object can only be mapped to one service type definition. From all activated synchronization packages, the service mappings are used and ordered according to their specified priority. The first condition that is true is taken to determine the STD. All further conditions are then ignored for that UCMDB CI or object.

In the Example of a servicemapping.xml File on page 33, 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 selecting the 6.5 version is checked first. If this does not find a match, the second rule is checked. If this also does not find a match, the default rules are checked as the SQL Server synchronization package has a higher priority than the default synchronization package.

Service Mapping Syntax

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

Maps the CI to the specified service type definition that is the concatenated string of the results of the operators. There may not be more than one `<STD>` element in the `<MapTo>` section.

The rules are processed for each CI in the order defined in the mapping file and as defined by the priority of the synchronization package that contains the mapping file. As soon as one condition is matched, no further rules are processed.

Example of a servicemapping.xml File

This example shows the content of a `servicemapping.xml` file in a synchronization package:

```
<?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>
```

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`

7 Node Mapping

The node type list defines which UCMDB CI types are imported as nodes into the HPOM node groups. For further information, see Node Type List on page 25.

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.

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

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.

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

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

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` looks 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` looks 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`.

8 Attribute Mapping

Attribute mapping allows you to change CI attributes and add new attributes to better describe a service and create a more detailed view of the environment. This also helps to make Smart Message Mapping more effective.

Attribute mapping is applied by all matching rules and are executed according to the priority of the bundle and the position within the rules declarations. 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 mappings are specified in the `attributemapping.xml` file from the associated synchronization package.

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

- `hosted_on`
- `display label`
- `CmdbCIcaption`
- `CmdbCiType`
- `CmdbObjectID`

If you encounter additional attributes, you can map them in this file based on CI attributes.

For nodes, the attributes `system_type`, `os_type`, and `version` are also set using the `attributemapping.xml` file. This is handled in the default `attributemapping.xml` file. You should not need to handle node attribute mappings in your synchronization package.

These attributes can be used as parameters to call tools in HPOM for Windows and applications in HPOM for UNIX.

Syntax: `$OPC_SERVICE_VALUE[<attribute>]`

Example: `$OPC_SERVICE_VALUE[hosted_on]`

To view a working example, take a look at the standard UI Launch tools.

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 55.

Example of Attribute Mapping

In this example, 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`

Set the following attributes:

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

```
<?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>
```

9 Testing and Deployment

This chapter contains information on:

- How to test rules and files
- Hints for writing rules
- Deployment and registration of UCMDB packages

Testing Rules and Files

This section contains information on:

- Validating XML configuration files
- Dumping synchronization data
- Testing mapping rules with the enrichment simulator

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 a XML editor such as Eclipse.

XML Schema Definition (XSD) is a standard from 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`

The files are:

bundle.xsd — Validates the `bundle.xml` file in each synchronization package.

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`)

nodetypes.xsd — Validates the node type mapping file `nodetypes.xml`.

schedule.xsd — Validates the scheduler configuration file `schedule.xml`.

Validation in the DMA Application

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

Validating Files Manually

Modern XML editors, such as Eclipse, allow 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:

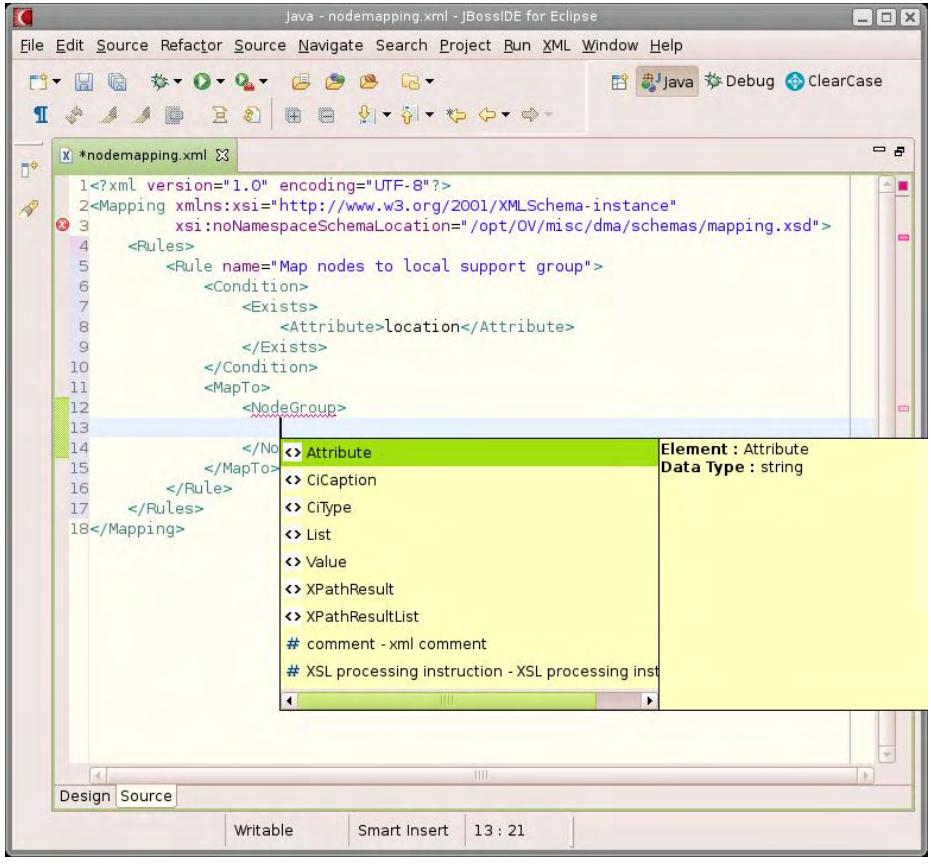
```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="<path to schema file>"
```

Replace *<path to schema file>* with the respective path 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="/opt/OV/misc/dma/schemas/
mapping.xsd">
...
</Mapping>
```

After you have added the reference, press **Ctrl+SPACE** and the Eclipse editor validates the file and suggests valid elements. See Figure 5 for an example.

Figure 5 Example of Validating a File



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 simulator or to check XPath expressions of rules

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>/conf/dma/DefaultSyncTask.settings`

2 Add the following line:

`sync.dumpData=true`

3 Start a synchronization using the HPOM DMA UI.

Figure 6 is a example of a data dump after enrichment has been performed.

Figure 6 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>/log/dma/`

Each 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 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

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 allows 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 with the Enrichment Simulator

The enrichment simulator allows 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 how to enable data dumps, see Creating a Synchronization Data Dump on page 44.

The start script is located in the following directory:

- **Windows**

  `<InstallDir>\support\dmaenrichsim.bat`

- **UNIX**

  `<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.

Hints for Writing Rules

The following is a set of guidelines for writing rules.

Easier 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 41 for more information.

When you have created rules, you can test these rules easily using the Enrichment Simulator. See Testing Mapping Rules with the Enrichment Simulator on page 47 for more information.

Avoid Complex XPath Queries

Try to avoid complex XPath queries, especially in general conditions, where such a query 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).

In Relative Expressions Only Match Against Existing Attributes

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.

Non-Matching XPath Expressions and Result Lists are Expensive

Especially XPath expressions that apply to multiple nodes, like expressions that contain `//` or `descendants:*/` but do not match or match only on nodes that are very distant from the current node are very performance expensive. 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.

Deployment and Registration of UCMDB Packages

Deploy and register HPOM DMA TQL queries on the system hosting the UCMDB or BAC.

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

1. Open the Package Manager:
   - **UCMDB:**

     Admin → Settings → Package Manager
   - **Business Availability Center:**

     Administration → Universal CMDB → Settings → Package Manager
2. Click **Upload package to server packages directory.**
3. Browse to the location of your TQL query package:
4. Select the *<TQLName>*`.zip` file from the Package list and click **Deploy.**

A Mapping Syntax

Mapping

Mapping is the mechanism used to map CIs from the UCMDB to services, attributes or nodes within HPOM as described in the following chapters.

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 the following section Mapping File Syntax on page 52.

Mapping File Syntax

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 and is therefore 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 condition is false, the next condition 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.

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.

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.

Matches

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

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

For example:

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

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.

Operand Elements

CI Type

```
<CiType/>
```

Return type: String

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

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.

Replace

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

Return type: String

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

Attribute

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

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 have to declare the following:

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

Optionally, you can use regular expressions. You can also use back references in the third operand. For more information on applicable regular expressions, see:

http://java.sun.com/j2se/1.5.0/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. 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 59.

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 59.

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']/dnsName<XPathResult>
        <Value>.europe.example.com</Value>
    </EndsWith>
</Condition>
```

Check if the current CI is imported as a node and that it is of the CI type 'unix' and that it has an HP-UX operating system installed on it.

```
<Condition>
  <And>
    <isNode/>
    <Equals>
        <Type/>
        <Value>unix</Value>
    </Equals
    <Contains>
       <XPathResults>/children[type='os']/caption</XPathResults>
        <value>HP-UX</value>
    </Contains>
  <And>

</Condition>
```

Mapping Elements

`<MapTo>` defines the mappings. Each concrete implementation of an engine adds its own XML elements for its individual mappings here. See Chapter 6, Service Mapping, Chapter 7, Node Mapping and Chapter 8, Attribute Mapping for some mapping element examples.

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 illustrated in Figure 7.

CI Data Structure

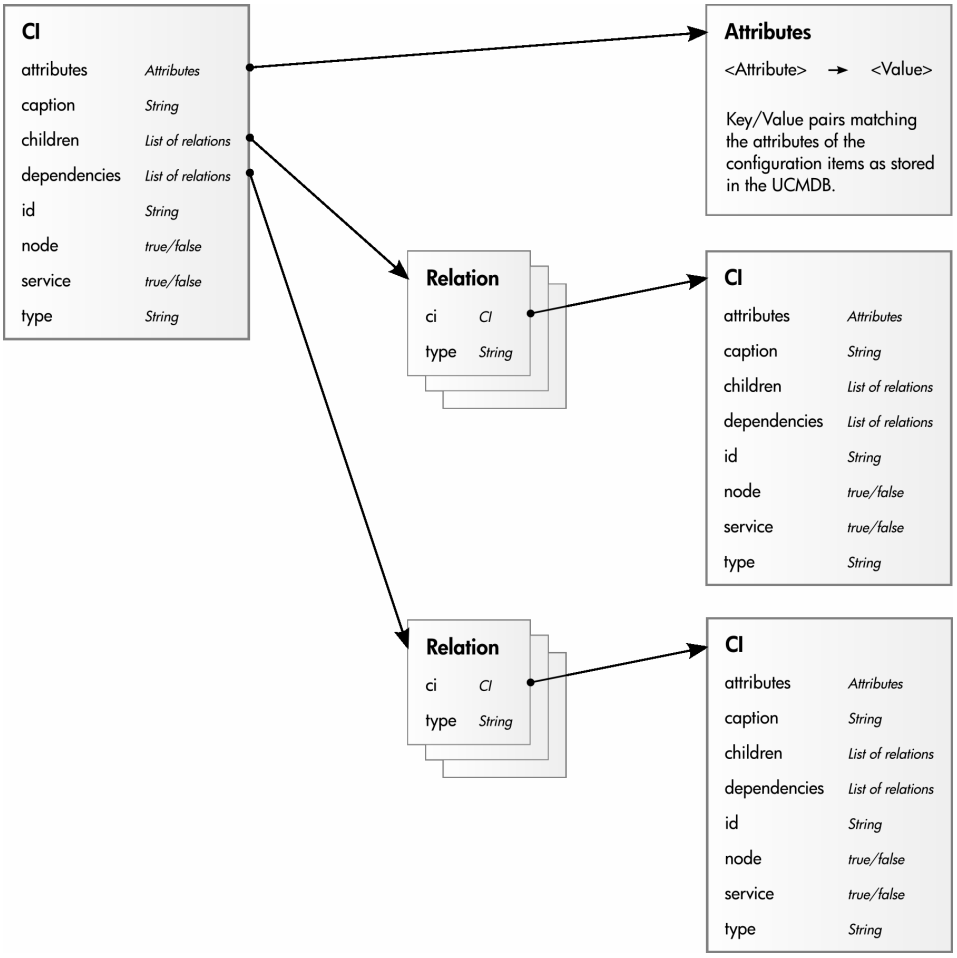
Attributes	Contains a map of all original UCMDB CI attributes. The key of this map is the name of the UCMDB CI attribute that references the UCMDB value of the UCMDB CI attribute.
Caption	Represents the name of the CI to be displayed in the service navigator. Caption has the same value as the UCMDB CI 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`.
id	The 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.

Relation Data Structure

CI Contains the reference to the CI to which the current CI is related.

Type Relation type as stored in the UCMDB.

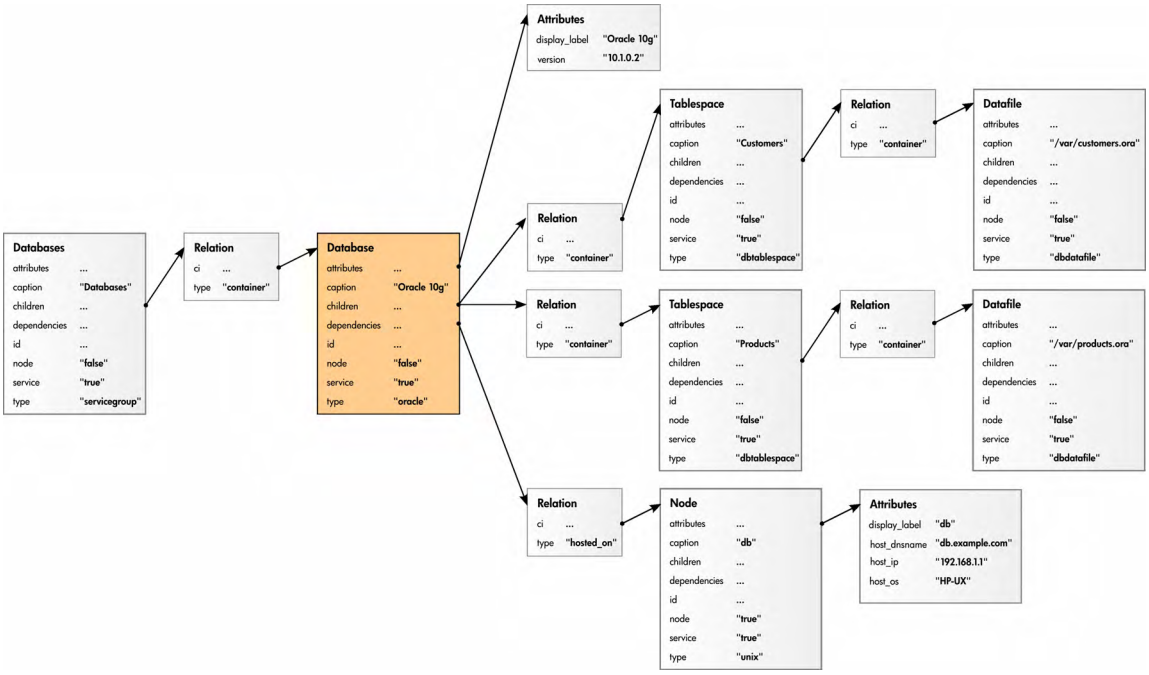
Figure 7 Data Structure Exposed to the Navigation



Example of an XPath-Navigated Data Structure

An example of an XPath-navigated data structure is illustrated in Figure 8. The host is a UNIX system, that has an Oracle application running on the HP-UX operating system. Starting point or context for the navigation is the CI that represents the Oracle application (orange background).

Figure 8 XPath Examples



XPath Expressions and Example Values

`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**
`//dependencies[type='hosted_on']/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 and point to the containing folder `Unix Nodes`.

Index

D

E

F

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X


invent


Java™
COMPATIBLE