HP Operations Manager

For the HP-UX, Red Hat Enterprise Linux, and Solaris operating systems

Software Version: 9.10

Service Discovery and Topology Synchronization Guide

Document Release Date: March 2012

Software Release Date: March 2012

Legal Notices

Support

Visit the HP Software Support Online web site at:

http://www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

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

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

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

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

To find more information about access levels, go to:

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

Contents

Chapter 1

Introduction

This document provides information on how to configure the following features of HPOM for UNIX and HPOM on Linux:

- Service auto-discovery

  This feature enables you to discover services in your environment and automatically populate your service hierarchy. Custom service auto-discovery enables you to discover services that are specific to your organization, and that you would otherwise need to add to your service hierarchy manually. For details, see "Service Auto-Discovery" (on page 6).

- Topology synchronization

  This feature enables you to automatically exchange and update node and service configurations between multiple HPOM management servers. Topology synchronization enables you to make sure that topology data is always up-to-date on all management servers in your environment without the need to maintain the data on each server manually. For details, see "Topology Synchronization" (on page 32).

Prerequisites

Custom service auto-discovery and topology synchronization are available after you install one of the following management server patches:

- HP Operations Manager for UNIX or Linux:
  - On the HP-UX operating system:

    PHSS_42736 IA-64 consolidated patch 09.10.220
  - On the Red Hat Enterprise Linux operating system:

    OML_00050 consolidated patch 09.10.220
  - On the Solaris operating system:

    ITOSOL_00772 consolidated patch 09.10.220

  Note: The patch installation only places a configuration file for custom service auto-discovery on the management server. You must manually upload the file after the patch installation is finished. For details, see "Upload the custom discovery module" (on page 13).

- HP Operations Manager for Windows:
  - On the Windows 32-bit operating system:

    OMW_00121 or higher
  - On the Windows 64-bit operating system:

    OMW_00122 or higher

Chapter 2

Service Auto-Discovery

A service hierarchy in HPOM is a logical organization of the services you provide. Services are the building blocks of your service hierarchy. A service may be anything from a low-level hardware component to a high-level software application. The services in a service hierarchy are dependent on each other. Rules determine the severity of a service based on the states of the contributing services. For more information about services in HPOM, see the chapter on HPOM Service Navigator in the *Administrator's Reference*.

You can add services to HPOM and create a basic service hierarchy using the following methods:

- Service configuration files

  Service configuration files are XML files that define individual services, the relationships between services, and the rules that determine a service's severity. You can create service configuration files manually or generate them automatically using scripts or programs. Before your services appear in the HPOM Java GUI, you must manually activate your service configuration using the opcservice command line tool. For more information about service configuration files and the opcservice tool, see the chapter on HPOM Service Navigator in the *Administrator's Reference*.

- Service auto-discovery policies

  Instead of building a service hierarchy based on service configuration files, you can use service auto-discovery policies to populate the service hierarchy on the HPOM management server. Service auto-discovery policies configure the discovery agent. The discovery agent runs the discovery script contained in the policy and collects the discovery data, for example, hardware resources, operating system attributes, applications, and other information from managed objects. The agent then sends the discovery data to the management server to be merged into the service hierarchy.

  After the first deployment, the auto-discovery policy is set to run periodically. Each time the discovery agent runs, it compares the service information retrieved with the results of the previous run. If the discovery agent finds any changes or additions to the services running on the managed node since the previous run, it sends that information to the HPOM management server, which updates the service hierarchy with the changes.

  For more information about the communication between the discovery agent and the management server, see "Discovery server and agent" (on page 9).

Service auto-discovery policies

Service auto-discovery policies are mostly supplied by HP Operations Smart Plug-ins (SPIs) to discover services in your managed environment and display them in a service hierarchy. You can also create your own custom service auto-discovery policies.

You must deploy a service auto-discovery policy to a managed node before it can be executed. When you remove a service auto-discovery policy from a managed node, the discovered services and relationships are removed from the service hierarchy.

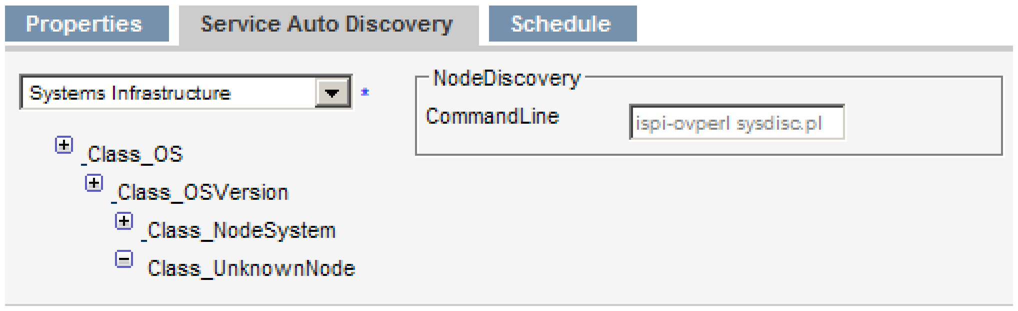
Usually there is no need to modify the SPI auto-discovery policies. However, certain SPIs may require that you configure this policy by adding parameter data such as a user name or password to allow access and discovery of specific applications on a managed node. You can also modify the execution schedule, if necessary. For details, see "Configure a service auto-discovery policy" (on page 12).

Management modules and service types

A service auto-discovery policy references a management module, a discovery script, and an execution schedule. The discovery script contains the discovery rules and actions that allow distributed discovery. The schedule defines when the script runs; it can be configured.

The management module contains the service types that define the service hierarchy structure to be populated by the discovery script. A service type is similar to a template. The service type is used when HPOM creates a service instance. The service instance inherits the properties and associations defined for the service type. For example, if you associate a service type with specific propagation rules, the rules are associated with every instance of that service that has been or will be created. The service type ensures that properties and associations are applied globally to all services of that type.

The following figure shows a service auto-discovery policy of the HP Operations Smart Plug-in for Systems Infrastructure. The policy references a hierarchy of service types, where the top-level service type is called `_Class_OS`. The policy runs the Perl script `ispi_ovperl sysdisc.pl`.



CommandLine: ispi-ovperl sysdisc.pl

A management module is stored on the management server. By itself it is not deployed or active. To be executed, any management module has to be referenced and started by a service auto-discovery policy.

HP Operations Smart Plug-ins (SPIs) provide predefined management modules and service types. The SPIs also provide the service auto-discovery policies and the discovery scripts.

HPOM does not allow you to create management modules, but does provide the Custom-Discovery management module. You can use the Custom-Discovery management module if you want to create your own custom service auto-discovery policies. For details, see "Custom Service Auto-Discovery" (on page 13).

Discovery server and agent

The service discovery server on the management server and the discovery agent are responsible for service discovery:

- The discovery policies configure the **discovery agent** (`agtrep`) to run scripts on the managed node that retrieve information about services. The discovery agent stores the services that it discovers in the agent repository (`agtrep.xml`), which is a local data store of services that exist on the managed node.

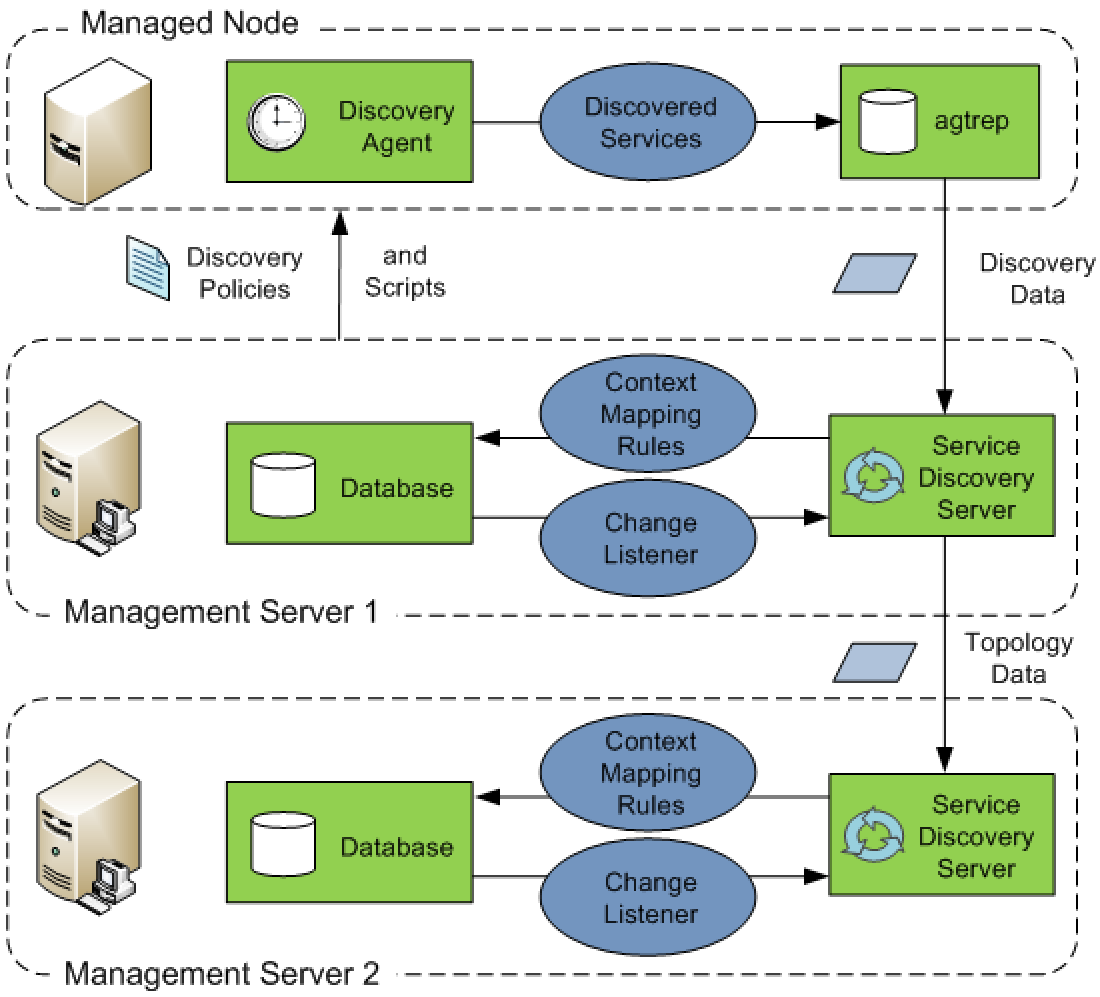
  When discovery runs for the first time, the agent sends all discovered services to the management server. After that the discovery agent forwards the changes only.

  You can modify the default behavior of the discovery agent by modifying configuration variables in the `agtrep` namespace. For details, see "ovagtrep" (on page 27).

- The **service discovery server** (`opcsvcdisc`) on the management server receives the discovered topology data and processes it by applying context mapping rules. Context mapping rules filter the topology data to determine which topology data updates the management server accepts. For details, see "Map topology data on the target" (on page 33).

  You can modify the default behavior of the discovery server by modifying configuration variables in the `om.svcdiscserver` and `om.svcdiscserver.mapping` namespaces. For details, see "Topology Synchronization Configuration Parameters" (on page 48) and "Creating a Synchronization Data Dump" (on page 50).

The following figure shows the flow of topology data from the discovery agent to the management server, and from there to another management server.



Note: The discovery agent sends the discovery data to the communication broker on the management server. The communication broker then forwards the discovery data to the discovery server. If a firewall separates the management server from the managed nodes, the communication broker port (default value 383) must be opened in the firewall. If you want to bind the discovery agent process `agtrep` to certain ports and IP addresses, configure the `CLIENT_PORT` and `CLIENT_BIND_ADDR` variables in the namespace `bbc.http.ext.agtrep.agtrep`. For more information, see the *HPOM Firewall Concepts and Configuration Guide*.

Removing discovered services

To remove a service, keep the following in mind:

- Undesired services in the service hierarchy that were discovered can be manually removed using the opcservice command line tool (for example with `opcservice -remove myService`). These services will not be rediscovered unless a change in the environment is detected. If the removed service is a parent service and has a component relationship, then the child service cannot be created. If a dependency relationship exists, the dependency cannot be created. You can find details of the failed actions in the service discovery server log file `/var/opt/OV/shared/server/log/OvSvcDiscServer.log`.

  You can use the ovagtrep command line tool (for example, `ovagtrep -publish`) to force agents to resend service discovery data . When the management server receives the service discovery data, it recreates the service hierarchy, including any services that you previously removed.

- The discovery agent deletes the services from the agent repository if a service discovery policy fails to discover existing services five times (by default). You can control how often a service auto-discovery policy must run before a missing service is automatically deleted by changing the agent parameter `INSTANCE_DELETION_THRESHOLD` in the `agtrep` namespace. For details, see "ovagtrep" (on page 27).

Configure a service auto-discovery policy

Usually there is no need to modify a SPI auto-discovery policy. However, certain SPIs may require that you configure a policy by adding parameter data such as a user name or password to allow access and discovery of specific applications on a managed node. You can also modify the schedule at which the policy runs.

1. Edit the service auto-discovery policy that you want to modify.
2. In the policy editor, click the **Service Auto Discovery** tab.

   The Service Auto Discovery tab shows the service types and the discovery command that the policy runs. Use the text boxes provided to enter or modify any editable parameters, for example user name or password.
3. Click the **Schedule** tab.

   The Schedule tab shows when and how often the service-discovery process runs. You can modify the schedule as needed.
4. Redeploy the modified policy to the managed nodes.
5. After the policy runs, the discovered services appear in the service hierarchy.

   To list all services use the following command:

   ```
   opcservice -list -subentity
   ```

   To assign the discovered services use the following command:

   ```
   opcservice -assign <operator> <service>
   ```

Custom Service Auto-Discovery

You can create new service auto-discovery policies to discover services in your environment and automatically populate your service hierarchy. The services that you discover can belong to any existing service type, including any new service types that you decide to configure.

Upload the custom discovery module

Custom service auto-discovery requires the custom discovery management module to discover services that are specific to your organization. For details, see "Service auto-discovery policies" (on page 7).

The patch installation only places the module configuration file (`DiscoveryInstances.mof`) on the management server. You must manually upload the file after the patch installation has completed. To upload the data, type the following command:

```
/opt/OV/bin/OpC/utils/mof_cfgupld.sh /opt/OV/contrib/\
OpC/HPOMDiscovery/DiscoveryInstances.mof
```

The module configuration file (`DiscoveryInstances.mof`) uses the MetaObject Facility (MOF) specification. For more information about MOF, see http://www.omg.org/mof/.

The command `mof_cfgupld.sh` logs information about the upload in the log file `/var/opt/OV/log/OpC/mgmt_sv/mof_upload.log`.

Create a service type

It is only possible to discover and synchronize services if the service type already exists on the target management server. A service type is similar to a template. The service instance inherits the properties and associations defined for the service type. For example, if you associate a service type with specific propagation rules, the rules are associated with every instance of that service that has been or will be created. The service type ensures that properties and associations are applied globally to all services of that type.

If the service type of a service does not exist on the target management server, you must create the service type manually. Alternatively, you can import service types that have previously been exported from HPOM for Windows.

To create a service type:

1. Create a text file that contains the service type definition.

   The following example defines a service type with the ID “My_Service_Type_Definition”, and also defines the calculation rule and the propagation rule when the service is placed under a service of the type “Class_ResourcePools”.

   ```
   <Services>
       <Service>
           <Name>My_Service_Type_Definition</Name>
           <OriginalId>$KeyFormat$</OriginalId>
           <Label>$Caption$</Label>
           <Description>$Description$</Description>
           <Icon>test.ico</Icon>
           <CalcRuleRef>DDK_DefaultCalculationRule</CalcRuleRef>
   ```

```
        </Service>
        <Association>
            <Composition/>
            <SourceRef>My_Service_Type_Definition</SourceRef>
            <TargetRef>Class_ResourcePools</TargetRef>
            <PropRuleRef>DDK_DefaultPropagationRule</PropRuleRef>
        </Association>
    </Services>
```

Tip: You can list existing service types using the following command:

```
/opt/OV/bin/ServiceTypeDefinitionCLI -l
```

2. Type the following command to upload the service type definition:

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

For more information about `ServiceTypeDefinitionCLI`, see "ServiceTypeDefinitionCLI" (on page 24).

To import service types from HPOM for Windows:

1. On the HPOM for Windows management server, export the HPOM for Windows service types, type:

   `ovpmutil cfg svt dnl <`*`service_types`*`>.mof /p <`*`service_type_ID`*`>`

   The following example command exports the parent service type `Root Service Type` with the ID `root` and all children service types to a file named `dnl-svt.mof`:

   ```
   ovpmutil cfg svt dnl c:\test\dnl-svt.mof /p root
   ```

2. On the HPOM for UNIX or Linux management server, import the HPOM for Windows service types, type:

   `/opt/OV/bin/OpC/utils/mof_cfgupld.sh <`*`service_types`*`>.mof`

   The following example command imports the service types contained in the file `dnl-svt.mof`:

   ```
   /opt/OV/bin/OpC/utils/mof_cfgupld.sh /tmp/dnl-svt.mof.mof
   ```

   The command `mof_cfgupld.sh` logs information about the upload in the log file `/var/opt/OV/log/OpC/mgmt_sv/mof_upload.log`.

Create a custom service auto-discovery script

Before you can configure a custom service auto-discovery policy, you must create a script (or program) that the agent can run on a managed node to discover services. This service discovery script must write details of each discovered service in XML to the standard output stream (STDOUT). The agent stores these details in the agent repository, which is a local data store of services that exist on the node. The agent publishes details of new, changed, and removed services to the management server, but does not resend details of unchanged services.

Tip: You can also add services to HPOM using the opcservice command on the management server. The services to add must be defined in a service configuration file, which you then upload to the management server. Compared to dynamic custom service discovery, which discovers the managed environment automatically, service configuration files are usually static and must be updated manually. For more information about adding services to HPOM using HPOM Sevice Navigator, see the *HPOM Administrator's Reference*.

Service XML Schema Definition (XSD)

Your service discovery script must output XML that conforms to the following schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="Service">
      <xs:complexType>
         <xs:choice maxOccurs="unbounded">
           <xs:element ref="NewInstance"/>
           <xs:element ref="DeleteInstance"/>
           <xs:element ref="NewRelationship"/>
           <xs:element ref="DeleteRelationship"/>
        </xs:choice>
      </xs:complexType>
      <xs:key name="InstanceKey">
         <xs:selector
xpath="NewInstance|DeleteInstance"></xs:selector>
         <xs:field xpath="Key"></xs:field>
      </xs:key>
      <xs:keyref refer="InstanceKey" name="InstanceKeyRef">
        <xs:selector xpath="NewInstance|DeleteInstance"></xs:selector>
        <xs:field xpath="@ref"></xs:field>
      </xs:keyref>
      <xs:keyref refer="InstanceKey" name="InstanceRef">
         <xs:selector
xpath="NewRelationship/*/Instance|DeleteRelationship/*/Instance"></xs:selector>
         <xs:field xpath="@ref"></xs:field>
      </xs:keyref>
   </xs:element>
   <xs:element name="NewInstance" type="InstanceType"/>
   <xs:element name="DeleteInstance" type="InstanceType"/>
   <xs:complexType name="InstanceType">
      <xs:sequence>
        <xs:element ref="Std"/>
        <xs:element ref="NodeGuid" minOccurs="0"/>
        <xs:element ref="Virtual" minOccurs="0"/>
        <xs:element ref="Key"/>
        <xs:element ref="GraphInstanceID" minOccurs="0"
maxOccurs="1"/>
        <xs:element ref="Attributes"/>
      </xs:sequence>
      <xs:attribute name="ref" type="xs:string" use="required"/>
   </xs:complexType>
   <xs:element name="NewRelationship" type="RelationType"/>
   <xs:element name="DeleteRelationship" type="RelationType"/>
   <xs:complexType name="RelationType">
```

```
         <xs:sequence>
            <xs:element ref="Parent"/>
            <xs:element ref="Components" minOccurs="0"/>
            <xs:element ref="DependentOn" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
      <xs:element name="Std">
         <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="DiscoveredElement"/>
           </xs:restriction>
         </xs:simpleType>
      </xs:element>
      <xs:element name="Virtual">
         <xs:complexType/>
      </xs:element>
      <xs:element name="NodeGuid" type="xs:string"/>
      <xs:element name="Key" type="xs:string"/>
      <xs:element name="GraphInstanceID" type="xs:string"/>
      <xs:element name="Attributes">
         <xs:complexType>
           <xs:sequence>
              <xs:element ref="Attribute" maxOccurs="unbounded"/>
           </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:element name="Attribute">
         <xs:complexType>
           <xs:attribute name="value" type="xs:string" use="required"/>
           <xs:attribute name="name" type="xs:string" use="required"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="Parent">
         <xs:complexType>
           <xs:sequence>
              <xs:element ref="Instance"/>
           </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:element name="DependentOn" type="InstanceList"/>
      <xs:element name="Components" type="InstanceList"/>
      <xs:complexType name="InstanceList">
         <xs:sequence>
            <xs:element ref="Instance" minOccurs="0"
   maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
      <xs:element name="Instance">
         <xs:complexType>
```

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

The following table describes the elements that the service XML document can contain.

Element	Description
NewInstance	Represents a discovered service. You must add a `ref` attribute, which must match the unique service ID that you specify in the *Key* element. You can then use this reference in *Instance* elements in the current XML document if you want to create or delete relationships.
DeleteInstance	Represents a service that you want to delete immediately. The agent automatically deletes previously discovered services from the agent repository if your service discovery script runs five times (by default) without including the service as a *NewInstance* in the XML document. You can control how often the service discovery script must run before a missing service is automatically deleted by changing the agent parameter `INSTANCE_DELETION_THRESHOLD` in the `agtrep` namespace. However, if you specify this element, the agent deletes the service immediately and publishes the change to the management server. If the service that you specify has related component services, the agent also deletes the component services.
NewRelationship	Defines a new relationship between services. This element must contain exactly one *Parent* element and can contain one or more *Components* and *DependentOn* elements.
DeleteRelationship	Defines relationships that you want to delete. This element must contain exactly one *Parent* element and can contain one or more *DependentOn* elements. **Note:** The agent does not allow to delete component relationships to avoid orphaned child services. To delete a component relationship, you must delete the component service using a *DeleteInstance* element.
Std	Must contain the string `DiscoveredElement`.
NodeGuid	Contains the core ID, primary node name, or IP address of the node that hosts this service. If this element contains an IP address, the management server must be able to resolve the corresponding primary node name (for example, from a correctly configured DNS server). You need to include this element only if the service is hosted on a different node. For example, you can include this element if your service discovery script runs on a central computer that has data about services on other computers.

Element	Description
Virtual	Include this element if the service is virtual. A virtual service is abstract and does not exist on any node. Omit this element if the service is hosted on a node.
Key	Contains the full service ID for this service, which must be unique. You must include this element in all *NewInstance* and *DeleteInstance* elements. You must not specify a *NewInstance* and *DeleteInstance* with the same key in the same XML document. **Tip:** If you do not want to invent IDs yourself, use one of the ID generators that are available on the Internet.
GraphInstanceID	Contains the unique ID of a graph that you want to associate with the service.
Attributes	Contains *Attribute* elements.
Attribute	Has a `name` attribute and a `value` attribute. You must include an *Attribute* element with the name `hpom_realstdid`. The value of this attribute must contain the unique ID of the service type that this service belongs to. The management server can use any other `name` and `value` pairs to generate the service's display name and description according to the formats in the service type. Alternatively, you can override the service type's display name and description formats, by including attributes with the names `hpom_captionformat` and `hpom_descriptionformat`.
Parent	Contains an *Instance* element, which defines the service that is the parent of this relationship. The parent instance that you specify must exist on the management server and in the agent repository on the node. Therefore, you may need to include a *NewInstance* element to add the parent to the agent repository, even if the parent already exists on the management server.
Instance	Has a `ref` attribute that refers to a *NewInstance* element in the current XML document.
DependentOn	Contains one or more *Instance* elements, which refer to the services that are dependent on the specified *Parent* element.
Components	Contains one or more *Instance* elements, which refer to the services that are components of the specified *Parent* element.

Your service discovery script must output an XML document that meets the following additional requirements:

- Each service instance must include an *Attribute* element with the name `hpom_realstdid`. The value of this attribute must contain the unique ID of the service type that this service belongs to. You can find this unique ID using the command `/opt/OV/bin/ServiceTypeDefinitionCLI -l`. In the output of this command, each `<Name>` element contains the unique ID of a service type.
- A service appears in the service hierarchy only after you create a component relationship with another service that already exists in the service hierarchy. For example, to add a service at the top level of the service hierarchy, you must create a component relationship with the root service, which has the service ID `Root_Services` and the service type unique ID `root`.

Example Service XML

The following examples show the XML that a script could output to create and maintain various services, relationships, and dependencies.

Create a new service instance and component relationship

The following XML creates a new service with the service ID `CustomServiceID`, which belongs to a custom service type that has the ID `{3C589DC6-6627-4C37-9818-91CB6B342E72}`. The XML specifies that the display name of the service is `Custom Service 1`, and the description is `Example custom service`.

The XML adds the new service to the service hierarchy as a component of the root service. The root service already exists on the management server, but is specified as a new service instance so that the agent adds it to the local agent repository.

```
<Service>
  <NewInstance ref="CustomServiceID">
    <Std>DiscoveredElement</Std>
    <Key>CustomServiceID</Key>
    <Attributes>
      <Attribute name="hpom_realstdid" value="My_Service_Type_
Definition_1" />
      <Attribute name="hpom_captionformat" value="Custom Service 1" />
      <Attribute name="hpom_descriptionformat" value="Example custom
service" />
    </Attributes>
  </NewInstance>
  <NewInstance ref="Root_Services">
    <Std>DiscoveredElement</Std>
    <Key>Root_Services</Key>
    <Virtual />
    <Attributes>
      <Attribute name="hpom_realstdid" value="root" />
    </Attributes>
  </NewInstance>
  <NewRelationship>
    <Parent>
      <Instance ref="Root_Services" />
    </Parent>
    <Components>
      <Instance ref="CustomServiceID" />
    </Components>
  </NewRelationship>
</Service>
```

Create a dependency relationship

The following XML creates two new services, which belong to different service types. The XML adds the new services to the service hierarchy as components of the root service. The XML also specifies a dependency relationship between the two new services.

Each service specifies an example attribute, which the management server can use in the service's display name. (The service type has a display name format that contains the corresponding variable `$ExampleAttribute$`.)

```
<Service>
  <NewInstance ref="ServiceA">
    <Std>DiscoveredElement</Std>
    <Key>ServiceA</Key>
    <Attributes>
      <Attribute name="hpom_realstdid" value="My_Service_Type_
Definition_2" />
      <Attribute name="ExampleAttribute" value="ExampleA" />
    </Attributes>
  </NewInstance>
  <NewInstance ref="ServiceB">
    <Std>DiscoveredElement</Std>
    <Key>ServiceB</Key>
    <Attributes>
      <Attribute name="hpom_realstdid" value="My_Service_Type_
Definition_3" />
      <Attribute name="ExampleAttribute" value="ExampleB" />
    </Attributes>
  </NewInstance>
  <NewInstance ref="Root_Services">
    <Std>DiscoveredElement</Std>
    <Key>Root_Services</Key>
    <Virtual />
    <Attributes>
      <Attribute name="hpom_realstdid" value="root" />
    </Attributes>
  </NewInstance>
  <NewRelationship>
    <Parent>
      <Instance ref="Root_Services" />
    </Parent>
    <Components>
      <Instance ref="ServiceA" />
      <Instance ref="ServiceB" />
    </Components>
  </NewRelationship>
  <NewRelationship>
    <Parent>
      <Instance ref="ServiceB" />
    </Parent>
    <DependentOn>
      <Instance ref="ServiceA" />
    </DependentOn>
  </NewRelationship>
</Service>
```

Move a dependency relationship

The following XML specifies that the services from the previous example still exist, but moves the dependency relationship from one service to another.

```
<Service>
  <NewInstance ref="ServiceA">
    <Std>DiscoveredElement</Std>
    <Key>ServiceA</Key>
    <Attributes>
      <Attribute name="hpom_realstdid" value="My_Service_Type_
Definition_2" />
      <Attribute name="ExampleAttribute" value="ExampleA" />
    </Attributes>
  </NewInstance>
  <NewInstance ref="ServiceB">
    <Std>DiscoveredElement</Std>
    <Key>ServiceB</Key>
    <Attributes>
      <Attribute name="hpom_realstdid" value="My_Service_Type_
Definition_3" />
      <Attribute name="ExampleAttribute" value="ExampleB" />
   </Attributes>
  </NewInstance>
  <NewInstance ref="Root_Services">
    <Std>DiscoveredElement</Std>
    <Key>Root_Services</Key>
    <Virtual />
    <Attributes>
      <Attribute name="hpom_realstdid" value="root" />
    </Attributes>
  </NewInstance>
  <NewRelationship>
    <Parent>
      <Instance ref="Root_Services" />
    </Parent>
    <Components>
      <Instance ref="ServiceA" />
      <Instance ref="ServiceB" />
    </Components>
  </NewRelationship>
  <DeleteRelationship>
    <Parent>
      <Instance ref="ServiceB" />
    </Parent>
    <DependentOn>
      <Instance ref="ServiceA" />
    </DependentOn>
  </DeleteRelationship>
  <NewRelationship>
    <Parent>
      <Instance ref="ServiceA" />
    </Parent>
    <DependentOn>
      <Instance ref="ServiceB" />
    </DependentOn>
  </NewRelationship>
</Service>
```

Delete service instances

The following XML immediately deletes a service with the service ID `CustomServiceID`, which belongs to a custom service type that has the ID `{3C589DC6-6627-4C37-9818-91CB6B342E72}`.

```
<Service>
  <DeleteInstance ref="CustomServiceID">
    <Std>DiscoveredElement</Std>
    <Key>CustomServiceID</Key>
    <Attributes>
      <Attribute name="hpom_realstdid" value="My_Service_Type_
Definition_1" />
    </Attributes>
  </DeleteInstance>
</Service>
```

Configure a custom service auto-discovery policy

After you create the script (or program) that the agent can run on a managed node to discover services, you must configure a service auto-discovery policy to start the script on the node with an appropriate schedule. You must also create an instrumentation category to ensure that the script and policy are deployed together.

1. Make sure that your management server meets the prerequisites for custom service auto-discovery. See "Prerequisites" (on page 5).

2. Create an instrumentation category, and then copy your service discovery script to the category subdirectory on the management server.

   For more information about creating instrumentation categories, see the *opcinstrumcfg(1m)* manual page.

3. Create a new policy of the type Service Auto-Discovery.
4. In the policy editor, click the Service Auto-Discovery tab.
5. In the list of management modules, select **Custom-Discovery**.
6. Modify **CommandLine** to specify the name of your service discovery script. You can specify a different command line for Windows and UNIX nodes.

   Use the variable `$ACTION_DIR` to represent the folder that contains instrumentation on managed nodes. For example, you could specify the command line `"$ACTION_DIR/custom_discovery.cmd"`.

   Escape any backslashes (\) with a second backslash (\\).
7. *Optional.* By default, the agent starts the service discovery script under the same account as the agent is running under. You can specify a different user and password if you want the script to run under a different account.
8. Configure the schedule in the Schedule tab.
9. In the Properties tab, specify a **Name** for the policy, and then click **Save**.
10. Assign to the policy the instrumentation category that contains the service discovery script.
11. Deploy the policy to the appropriate managed nodes.

    Tip: The maximum frequency that you can schedule for a discovery policy is hourly. This frequency may not be convenient when you are developing and testing your policy. However, after you , you can run the policy on demand using the command `ovagtrep -run <policy name>` on the .
12. After the policy runs, the discovered services appear below the AutoDiscovery service.

    To list all services use the following command:

    ```
    opcservice -list -subentity
    ```

    To assign the discovered services use the following command:

    ```
    opcservice -assign <operator> AutoDiscovery
    ```

ServiceTypeDefinitionCLI

The command `ServiceTypeDefinitionCLI` configures the service types used when an instance of a service is created.

SYNOPSIS

```
ServiceTypeDefinitionCLI [-l]
                         [-a <filename> [-o]]
                         [-d <definition>]
```

DESCRIPTION

A service type is similar to a template. The service type is used when HPOM creates a service instance. The service instance inherits the properties and associations defined for the service type. For example, if you associate a service type with specific propagation rules, the rules are associated with every instance of that service that has been or will be created. The service type ensures that properties and associations are applied globally to all services of that type.

The `ServiceTypeDefinitionCLI` command has the following options:

`-l`

Lists existing service types and their associated propagation rules.

`-a` *`<filename>`*

Uploads the service type definitions contained in *`<filename>`*. For more information about how to define service types in a text file, see "Create a service type" (on page 13).

`-o`

Updates existing service types with modified definitions. If you use the `-a` option without the `-o` option and *`<filename>`* contains modifications to existing services types, those modifications will not be uploaded.

`-d` *`<service_type_name>`*

Deletes service types and their associated propagation rules.

EXAMPLES

- You can list existing service types using the following command:

  ```
  /opt/OV/bin/ServiceTypeDefinitionCLI -l
  ```

- To upload service types defined in the file `my_service_types.txt` to the management server, type:

  ```
  /opt/OV/bin/ServiceTypeDefinitionCLI -a my_service_types.txt
  ```

- To update the service type `My_Service_Type_Definition` with a new definition, type:

  ```
  /opt/OV/bin/ServiceTypeDefinitionCLI -a my_service_types.txt -o
  ```

enableToposync

The `enableToposync.sh` script configures topology synchronization.

SYNOPSIS

```
/opt/OV/contrib/OpC/enableToposync.sh
          [ -upload ]
          [ -sched ]
          [ -online [-target <server_list>] ]
          [ -cert_import <file> ]
          [ -stop ]
```

DESCRIPTION

Topology synchronization enables you to automatically exchange and update node and service configurations between multiple HPOM management servers. The script `enableToposync.sh` helps you to configure topology synchronization.

The `enableToposync.sh` script has the following options:

`-upload`

This option is deprecated.

`-sched`

This option is deprecated.

`-online`

Configures the source management server to send any topology data changes immediately.

This option sets the configuration parameter `OPC_CONFIG_CHANGE_SYNC` in the namespace `opc` to `TRUE` and then restarts the service discovery server. For details, see "Topology Synchronization Configuration Parameters" (on page 48).

`-cert_import <`*`file`*`>`

Imports the target management server's trusted certificates stored in `<`*`file`*`>`.

To export a trusted certificate on a target management server, use the command `/opt/OV/bin/ovcert -exporttrusted -ovrg server -file <`*`file`*`>` and copy `<`*`file`*`>` to the source management server. You must also export the source management server's trusted certificate and import it to the keystore of the target management servers. For details see "Start synchronization" (on page 46).

`-target <`*`comma_separated_server_list`*`>`

Replace `<`*`comma_separated_server_list`*`>` with the fully qualified domain name of the target management server. If you have more than one target management server, separate each server name with a comma (,). Do not include spaces in the server list.

This option sets the configuration parameter `ForwardingTargets` in the namespace `om.svcdiscserver`. For details, see "Topology Synchronization Configuration Parameters" (on page 48).

`-stop`

Stops online synchronization of changes, clears the list of target servers, and restarts the service discovery server.

ovagtrep

`ovagtrep` enables configuration and control of the discovery agent and agent repository.

SYNOPSIS

```
ovagtrep  [-clearall] |
          [-run <policy name>] |
          [-publish]
```

DESCRIPTION

The discovery agent is an extension to the HTTPS agent, which runs service discovery policies that have been deployed from a management server. It stores the services that it discovers in the agent repository, which is a local data store of services that exist on the node.

The agent synchronizes the services in the agent repository with the management server. The management server receives details of new, changed, and removed services only. Details of unchanged services are not resent.

The `ovagtrep` command enables you to configure and control the discovery agent and agent repository. It has the following options:

`-clearall`

Clears all services from the agent repository. The next time that the discovery agent runs service discovery policies, it will recreate the services. The agent then synchronizes the services with the management server. This is enables you to force the agent to synchronize unchanged services with the management server.

`-run <policy name>`

Runs a service discovery policy. Use this to run a policy at an unscheduled time, to discover any changes immediately. The agent sends details of changes to the management server. You can find the names of installed policies using `ovpolicy`.

`-publish`

Resends details of all the services that are currently in the agent repository to the management server. Use this for troubleshooting if services fail to appear on the management server.

The discovery agent and agent repository are part of a component that is registered with the control service. You can start and stop the component with the commands `ovc -start agtrep` and `ovc -stop agtrep`.

The agent repository is stored in the `agtrep.xml` file:

- Windows:

  `%OvDataDir%datafiles\agtrep.xml`

- UNIX and Linux:

  `/var/opt/OV/datafiles/agtrep.xml`

You can use the command `ovconfchg` to modify the following settings in the `agtrep` name space:

`ACTION_TIMEOUT <minutes>`

Sets the maximum number of minutes that a service discovery policy can run. If the policy runs any longer, the discovery agent stops running the policy and logs an error in the system log (`<data_dir>/log/System.txt`).

`INSTANCE_DELETION_THRESHOLD <value>`

Sets the number of times that service discovery policies must fail to discover existing services before the agent deletes the services from the agent repository.

If a service discovery policy can no longer discover a service that exists in the agent repository, the discovery agent deletes the service from the agent repository only after the service discovery policy has run the number of times that you specify with this setting.

For example, to set the action timeout to five minutes with the following command:

```
ovconfchg -ns agtrep -set ACTION_TIMEOUT 5
```

After you change the action timeout or instance deletion threshold, restart the component with the following command:

```
ovc -restart agtrep.
```

opcsvcdisc

`opcsvcdisc` is the service discovery server process on the HPOM management server.

DESCRIPTION

The service discovery server (`opcsvcdisc`) on the management server receives the discovered topology data and processes it by applying context mapping rules. Context mapping rules filter the topology data to determine which topology data updates the target management server accepts. For details, see "Map topology data on the target" (on page 33).

Configuration variables

You can use the command `ovconfchg` to modify the following settings:

- `OPC_CONFIG_CHANGE_SYNC`

  Enables online configuration synchronization between management servers. The synchronization updates changes to nodes, node groups, and services. In addition, node to node group assignments are synchronized.

  Namespace: `opc`

  Default value: `FALSE`

  For details, see "Topology Synchronization Configuration Parameters" (on page 48).

- `ForwardingTargets`

  Comma-separated list of target management servers. Format: `<server>:port,`.... Requires a restart of the `opcsvcdisc` process.

  Namespace: `om.svcdiscserver`

  Default value: empty

  For details, see "Topology Synchronization Configuration Parameters" (on page 48).

- `LOG_LEVEL`

  The service discovery server supports the following log levels:

  - Log level 1 logs errors only.
  - Log level 3 logs errors and information (including raw data received from the agent).
  - Log level 10 logs tracing information for debugging purposes, for example method parameters.

  Namespace: `om.svcdiscsserver`

  Default value: `3`

  For details, see "Log Level Configuration" (on page 52).

- `opr.toposync.dumpData`

  Dumps the synchronization data into XML files.

  Namespace: `om.svcdiscserver.mapping`

Default value: `FALSE`

For details, see "Creating a Synchronization Data Dump" (on page 50).

Log files

The service discovery server logs are stored in the following log files:

- `/var/opt/OV/shared/server/log/OvSvcDiscServer.log`

  `OvSvcDiscServer.log` contains log information related to the discovery server (for example, receiving data, processing data, and storing data in the model).

  For details about increasing the level of detail maintained in the log file, see "Service Discovery Server Log Level Configuration" (on page 52).

- `/var/opt/OV/shared/server/log/opr-svcdiscserver.log`

  `opr-svcdiscserver.log` contains log information related to mapping and filtering.

  For details about increasing the level of detail maintained in the log file, see "Mapping Log Level Configuration" (on page 52).

Testing and troubleshooting

Manually run a discovery policy

To make sure that all discovered services are forwarded to the management server before the next scheduled discovery, complete the following steps:

1. On the managed node, clear the agent repository, type:

   ```
   ovagtrep -clearall
   ```

2. Run the discovery policy, type:

   ```
   ovagtrep -run <policy_name>
   ```

   For example, type `ovagtrep -run "SI-SystemDiscovery"` to run the Systems Infrastructure SPI policy SI-SystemDiscovery.

Log files

The discovery agent logs are stored in the following log files:

- Windows:

  ```
  %OvDataDir%log\System.txt
  ```

- UNIX and Linux:

  ```
  /var/opt/OV/log/System.txt
  ```

The service discovery server logs are stored in the following log files:

- `/var/opt/OV/shared/server/log/OvSvcDiscServer.log`

  `OvSvcDiscServer.log` contains log information related to the discovery server (for example, receiving data, processing data, and storing data in the model).

  For details about increasing the level of detail maintained in the log file, see "Service Discovery Server Log Level Configuration" (on page 52).

- `/var/opt/OV/shared/server/log/opr-svcdiscserver.log`

  `opr-svcdiscserver.log` contains log information related to mapping and filtering.

  For details about increasing the level of detail maintained in the log file, see "Mapping Log Level Configuration" (on page 52).

Chapter 3

Topology Synchronization

In an environment with multiple HPOM management servers, you can automatically exchange topology data between management servers by configuring topology synchronization.

You can synchronize topology data between the following products:

- HP Operations Manager for UNIX (HPOM for UNIX) version 9.10 with the appropriate server patch
- HP Operations Manager on Linux (HPOM on Linux) version 9.10 with the appropriate server patch
- HP Operations Manager for Windows (HPOM for Windows) version 8.16 and higher
- HP Operations Manager i (HP OMi) version 9.00 and higher

Topology synchronization includes the following types of topology data:

- Nodes
- Node groups
- Node hierarchies

  Note: If you synchronize hierarchical node groups from HPOM for Windows to an HPOM for UNIX or HPOM on Linux management server, the hierarchy becomes a flat list of node groups on the HPOM for UNIX or HPOM on Linux management server.

- Layout groups
- Services
- Assignments between these types of topology data

These types of topology data are referred to generically as configuration items (CIs).

Note: Topology synchronization does not forward other types of configuration data, for example policies, instruction texts, or applications. For information about exchanging the complete HPOM configuration (for example in server pooling environments), see the *HPOM Administrator's Reference*.

When you use topology synchronization, the source management servers forward topology data to a list of target management servers that you specify. The source management servers send all the topology data, and the target management servers process the data that they receive by applying context mapping rules. Context mapping rules filter the topology data to determine which topology data updates the target management server accepts.

Therefore, when you configure topology synchronization, make sure that you configure the context mapping rules on the target management server before the source management server starts to send topology data.

Map topology data on the target

Target management servers filter configuration items (CIs) in topology data that they receive by applying context mapping rules. A context is a group of CIs. A context mapping applies a context to CIs that match specified conditions. The management server synchronizes all the CIs that you map to any context, and ignores all other CIs.

HPOM provides default context mapping rules. The default context mapping rules map incoming CIs to a context, except for some default CIs and some CIs that relate to HP BSM Operations Management. Therefore, if you use the default context mapping rules, a target management server synchronizes almost all the topology data that it receives from source management servers.

You can modify the default context mapping rules so that the target management server synchronizes only the CIs that you need on that server. You modify the default context mapping rules by changing the following XML file:

```
/var/opt/OV/shared/server/conf/discovery/sync-
packages/default/contextmapping.xml
```

Note: The default context mapping file is part of a default synchronization package, which also contains other types of mapping files. The other mapping files are reserved for future use. You must not remove the default synchronization package.

The default context mapping file maps the context called `default` to those services that do not have the ID `root_services`, `systemservices`, or `applicationservices`, and to those nodes that are not of the type `agent`, `omserver`, `ipaddress`, or `interface`.

Default context mapping file

```
<?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="Filter out default and OMi related instances">
        <Condition>
          <And>
            <Not>
              <Equals ignoreCase="true">
                <OMId/>
                <Value>root_services</Value>
              </Equals>
            </Not>
            <Not>
              <Equals ignoreCase="true">
                <OMId/>
                <Value>systemservices</Value>
              </Equals>
            </Not>
            <Not>
              <Equals ignoreCase="true">
                <OMId/>
                <Value>applicationservices</Value>
              </Equals>
            </Not>
            <Not>
              <Equals>
                <OMType/>
                <Value>agent</Value>
              </Equals>
            </Not>
            <Not>
              <Equals>
                <OMType/>
                <Value>omserver</Value>
              </Equals>
            </Not>
            <Not>
              <Equals>
                <OMType/>
                <Value>ipaddress</Value>
              </Equals>
            </Not>
            <Not>
              <Equals>
                <OMType/>
                <Value>interface</Value>
              </Equals>
            </Not>
          </And>
        </Condition>
        <MapTo>
             <Context>default</Context>
        </MapTo>
      </Rule>
    </Rules>
  </Mapping>
```

If you want to exclude more topology data, you can modify the contents of the `Condition` element. The `Condition` element contains operator elements (for example `And`, `Not`, `Equals`). Some of these operator elements contain other operator elements, and others contain operand elements (for example, `OMId`, `OMType`, `Value`).

Example domain name condition

The following example shows a fragment of XML, which excludes nodes that are in the domain `example.com`:

```
<Not>
  <And>
    <IsNode/>
    <Matches>
      <OMAttribute>PrimaryNodeName</OMAttribute>
      <Value>.*\.example\.com</Value>
    </Matches>
  </And>
</Not>
```

Example service type ID condition

The following example shows a fragment of XML, which excludes services that have a service type ID that starts with "SiteScope":

```
<Not>
  <StartsWith>
    <OMType/>
    <Value>SiteScope</Value>
  </StartsWith>
</Not>
```

Example origin server condition

The following example shows a fragment of XML, which excludes CIs that originate from the server hpom5.example.com:

```
<Not>
  <Equals>
     <OriginServer/>
     <Value>hpom5.example.com</Value>
  </Equals>
</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. 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 child operator returns true.

Exists

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

The value of the operand must 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.

True if the element is a managed node in HPOM.

Is Root CI

```
<IsRootCI/>
```

True if the CI is a root CI (a root CI has no parent).

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.

Example:

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

For more information on applicable regular expressions, see:

http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html

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.

Is Deletion CI

```
<IsDeletionCI/>
```

True if the CI is used to delete CIs.

Operand Elements

HP Operations Manager Service ID

```
<OMId/>
```

Return type: String

Returns the HPOM ID string of the CI as stored in Operations Management. The HPOM ID returns different values as follows:

Services: HPOM ID is the service ID

Nodes: HPOM ID is the unique ID

Node Groups: HPOM ID is the node group ID

Operations Manager Type

```
<OMType/>
```

Return type: String

Returns the HPOM Type stored in Operations Management. For HPOM services, the HPOM Type is the service type definition. For nodes, the HPOM Type is set to the constant value “node”.

Caption

```
<Caption/>
```

Return type: String

Returns the caption string of the CI in the RTSM or BSM.

HP Operations Manager Attribute

```
<OMAttribute>[Name]</OMAttribute>
```

Return type: String

Returns the value of the HPOM attribute with the given name.

Replace

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

Return type: String

Replaces the 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://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`.

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.

Parent CI

```
<ParentCI/>
```

Return type: CI

Returns the parent CI of the current CI. If the current CI is the root CI, null is returned.

Tip: To check for the root CI, use the `IsRoot` operator.

Child CI

```
<ChildCI>
    [Operator]
</ChildCI>
```

```
<ChildCI relationType="[relationType]">
    [Operator]
</ChildCI>
```

Return type: CI

Description: Returns the first child CI of the current CI that matches the enclosed operator.

Optional elements:

`relationType`: Only follow relations with the specified relation type.

Child CI List

```
<ChildCIList>
     [Operator (Optional)]
</ChildCIList>
```

```
<ChildCIList relationType="[relationType]">
     [Operator (Optional)]
</ChildCIList>
```

Return type: List of CIs

Returns all CI children of the current CI.

Optional elements:

`Operator`: Only CIs that match the operator will be returned.

`relationType`: Only follow relations with the specified relation type.

Ancestor CI

```
<AncestorCI>
    [Operator]
</AncestorCI>
```

```
<AncestorCI relationType="[relationType]">
    [Operator]
</AncestorCI>
```

Return type: CI

Returns the first ancestor CI of the current CI that matches the enclosed operator. An ancestor CI is the parent or parent of the parent (and so on) of the current CI.

Optional elements:

`relationType`: The dependency must have the specified relation type.

Descendant CI

```
<DescendantCI>
    [Operator]
</DescendantCI>
```

```
<DescendantCI relationType="[relationType]">
    [Operator]
</DescendantCI>
```

Return type: CI

Returns the first descendant CI of the current CI that matches the enclosed operator. A descendant CI is the child or child of the child (and so on) of the current CI.

Optional elements:

`relationType`: Only follow relations with the specified relation type.

Descendant CI List

```
<DescendantCIList>
    [Operator (Optional)]
</DescendantCIList>
```

```
<DescendantCIList relationType="[relationType]">
    [Operator (Optional)]
</DescendantCIList>
```

Return type: List of CIs

Returns the all descendant CIs of the current CI. A descendant CI is the child or child of the child (and so on) of the current CI.

Optional elements:

`Operator`: Only CIs that match the operator will be returned.

`relationType`: Only follow relations with the specified relation type.

Dependency CI

```
<DependencyCI>
    [Operator]
</DependencyCI>
```

```
<DependencyCI relationType="[relationType]">
    [Operator]
</DependencyCI>
```

Return type: CI

Returns the first dependency CI that matched the included operator.

Optional elements:

`relationType`: Only follow relations with the specified relation type.

Dependency CI List

```
<DependencyCIList>
    [Operator (Optional)]
</DependencyCIList>
```

```
<DependencyCIList relationType="[relationType]">
    [Operator (Optional)]
</DependencyCIList>
```

Return type: CI

Returns the list of dependencies.

Optional elements:

`Operator`: Only CIs that match the operator will be returned.

`relationType`: The dependency must have the specified relation type.

Dependent CI

```
<DependentCI>
    [Operator]
</DependentCI>
```

```
<DependentCI relationType="[relationType]">
    [Operator]
</DependentCI>
```

Return type: CI

Returns the first dependent CI that matched the included operator.

Example for a dependent CI:

ServiceA > hosted_on > HostB

In this case ServiceA is a dependent CI of HostB. That means if you have HostB and want to have all services that depend on this host, you have to use the `<DependentCI>` operand. If you have ServiceA and want to have HostB, you have to use the `<DependencyCI>` operand instead.

Optional elements:

`relationType`: Only follow relations with the specified relation type.

Dependent CI List

```
<DependentCIList>
    [Operator (Optional)]
</DependentCIList>
```

```
<DependentCIList relationType="[relationType]">
    [Operator (Optional)]
</DependentCIList>
```

Return type: CI

Returns the list of dependent CI.

Example for a dependent CI:

ServiceA > hosted_on > HostB

In this case ServiceA is a dependent CI of HostB. That means if you have HostB and want to have all services that depend on this host, you have to use the `<DependentCI>` operand. If you have ServiceA and want to have HostB, you have to use the `<DependencyCI>` operand instead.

Optional elements:

Operator: Only CIs that match the operator will be returned.

relationType: The dependency must have the specified relation type.

From CI Get

```
<From>
    <CI>
        [CI Operand]
    </CI>
    <Get>
        [Operand]
    </Get>
</From>
```

Return type: Return type of the second operand.

Using this operand you can get values from another CI. The first operand [CI Operand] must return a CI instance. The second operand operates on that CI instance and the value of this second operand will be returned by this From operand.

Example:

```
<From>
    <CI>
        <ParentCI>
    </CI>
    <Get>
        <Caption/>
    </Get>
</From>
```

Returns the caption from the parent CI of the current CI.

Origin Server

`<OriginServer/>`

Return type: String

This operand returns the hostname of the server that originally received the discovery data before forwarding it to other servers.

Forward topology data

Topology synchronization enables you to automatically exchange and update node and service configurations between multiple HPOM management servers. A process on the source management server listens for additions, modifications, or deletions to topology data. When a change occurs, the source management server immediately forwards the changed data to the target management servers that you specify. The listener process ensures that the topology data on all servers is always up to date.

The listener process only forwards changes to topology data that occur *after* you enable it. You therefore need to forward all the topology data that already exists on HPOM management servers manually. (You forward topology data manually using the tool `startInitialSync` on the source management server.)

Caution: Topology synchronization only forwards changes to topology data that occur when the management server processes are running.

Note: Previous releases of HPOM also supported scheduled synchronization based on service auto-discovery policies. Because of performance issues associated with these service auto-discovery policies you are encouraged to migrate to online synchronization. For details, see "Migrate from scheduled synchronization" (on page 47).

Start synchronization

To start forwarding topology data, complete the following steps:

1. Make sure that your management server meets the prerequisites for topology synchronization. See "Prerequisites" (on page 5).
2. Configure each management server to trust certificates from the other management servers:
   a. On the source and target management server, export the trusted certificate to a file using the following command:
      ```
      /opt/OV/bin/ovcert -exporttrusted -ovrg server -file <file>
      ```
      The command generates a file with the name that you specify.
   b. Copy each file to the corresponding management server, and then import the trusted certificate using the following commands:
      ```
      /opt/OV/bin/ovcert -importtrusted -ovrg server -file <file>
      ```
      ```
      /opt/OV/bin/ovcert -importtrusted -file <file>
      ```
3. Type the following command to enable topology synchronization:
   ```
   /opt/OV/contrib/OpC/enableToposync.sh -online -target <comma_
   separated_server_list>
   ```
   Replace `<comma_separated_server_list>` with the fully qualified domain name of the target management server. If you have more than one target management server, separate each server name with a comma (,). Do not include spaces in the server list.

   This command restarts the service discovery server. The source management server begins to send any topology data changes immediately.

4. Type the following command to start the initial synchronization of topology data:

   ```
   /opt/OV/bin/OpC/startInitialSync.sh
   ```

For more information about enableToposync.sh, see "enableToposync" (on page 25).

Migrate from scheduled synchronization

To migrate from scheduled synchronization, complete the following steps:

1. Make sure that your management server meets the prerequisites for topology synchronization. See "Prerequisites" (on page 5).
2. Clear the agent repository cache on the management server using the following command:

   ```
   /opt/OV/bin/ovagtrep -clearall
   ```

3. Remove the service auto-discovery policies from the management server node using the following command:

   ```
   /opt/OV/bin/ovpolicy -remove DiscoverOM
   ```

   ```
   /opt/OV/bin/ovpolicy -remove DiscoverOMTypes
   ```

4. Deassign the service auto-discovery policies from the management server node using the following command:

   ```
   /opt/OV/bin/OpC/utils/opcnode -deassign_pol node_name=<management_
   server> net_type=NETWORK_IP pol_name=DiscoverOMTypes
   pol_type=svcdisc
   ```

   ```
   /opt/OV/bin/OpC/utils/opcnode -deassign_pol node_name=<management_
   server> net_type=NETWORK_IP pol_name=DiscoverOM
   pol_type=svcdisc
   ```

   ```
   /opt/OV/bin/OpC/opcragt -dist <management_server>
   ```

   Replace `<management_server>` with the name of the management_server.

5. Type the following command to enable topology synchronization:

   ```
   /opt/OV/contrib/OpC/enableToposync.sh -online
   ```

   This command restarts the service discovery server. The source management server begins to send any topology data changes immediately.

6. Type the following command to start the initial synchronization of topology data:

   ```
   /opt/OV/bin/OpC/startInitialSync.sh
   ```

Stop forwarding topology data

Type the following command to stop forwarding topology data:

```
/opt/OV/contrib/OpC/enableToposync.sh -stop
```

This command stops online synchronization of changes, clears the list of target servers, and restarts the service discovery server.

Testing and Troubleshooting

Topology Synchronization Configuration Parameters

The script `enableToposync.sh` sets two main configuration parameters. You can also set these parameters directly.

- To change the target management servers, type the following command:

  ```
  /opt/OV/bin/ovconfchg -ovrg server -ns om.svcdiscserver -set
  ForwardingTargets <comma_separated_server_list>
  ```

  If you change this parameter, you must restart the service discovery server by typing the following command:

  ```
  /opt/OV/bin/ovc -restart opcsvcdisc
  ```

- To enable synchronization:

  ```
  /opt/OV/bin/ovconfchg -ovrg server -ns opc -set OPC_CONFIG_CHANGE_
  SYNC TRUE
  ```

  You can change this parameter at any time, without restarting the service discovery server.

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 suitable XML editor. You can use Eclipse or another editor of your choice that is capable of validating an XML file against a schema.

XSD[1] is a standard from World Wide Web Consortium (W3C) for describing and validating the contents of XML files. XSD files are provided for all XML configuration files.

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

The XML schema definition that you can use to validate `contextmapping.xml` is in the following file:

```
/var/opt/OV/shared/server/conf/discovery/schemas/mapping.xsd
```

Validating Files Automatically

Each configuration file is automatically validated against the associated XSD file whenever it is read. 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

With a modern XML editor, you can 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:

---

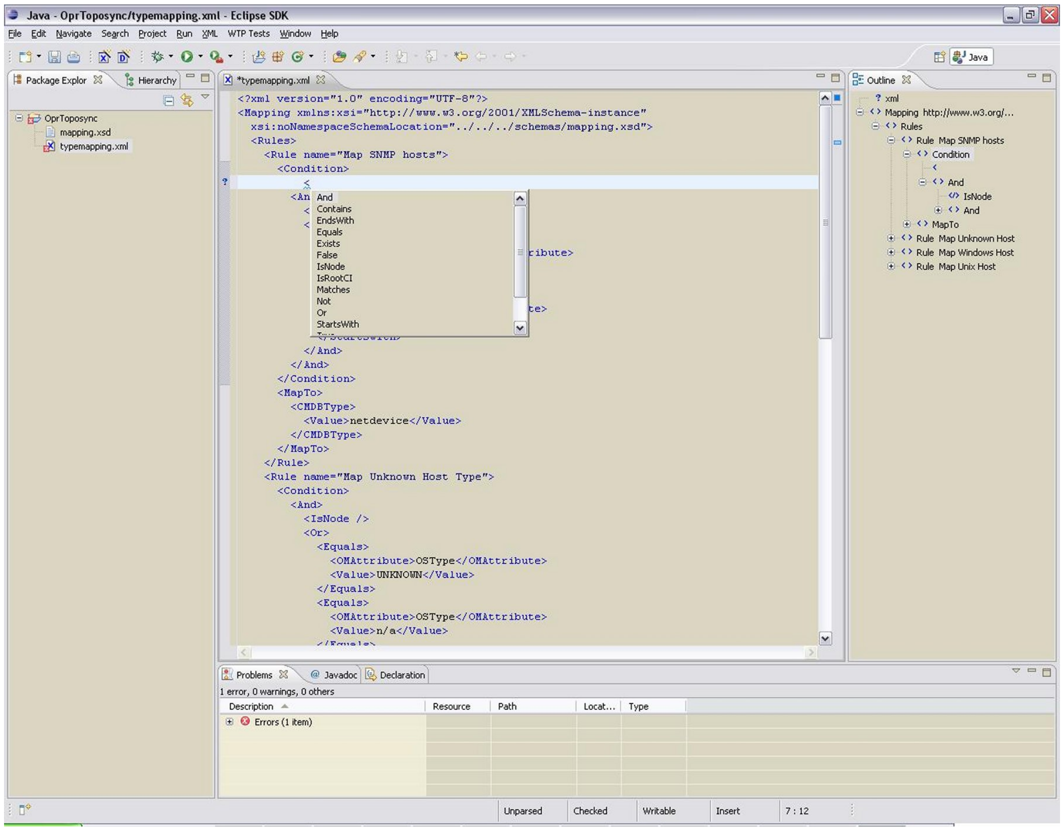
[1]XML Schema Definition

```
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 a `contextmapping.xml` file, add the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=
"/var/opt/OV/shared/server/conf/discovery/schemas/mapping.xsd">
...
</Mapping>
```

After you have added the reference, the Eclipse editor validates the file and suggests valid elements when pressing **CTRL+SPACE** during editing. See the following figure for an 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="Map SNMP hosts">
      <Condition>
        <
          <An
            <
            <
                                    ribute>

                                    te>

            </And>
          </And>
        </Condition>
        <MapTo>
          <CMDBType>
            <Value>netdevice</Value>
          </CMDBType>
        </MapTo>
      </Rule>
      <Rule name="Map Unknown Host Type">
        <Condition>
          <And>
            <IsNode />
            <Or>
              <Equals>
                <OMAttribute>OSType</OMAttribute>
                <Value>UNKNOWN</Value>
              </Equals>
              <Equals>
                <OMAttribute>OSType</OMAttribute>
                <Value>n/a</Value>
```

```
? xml
<> Mapping http://www.w3.org/...
  <> Rules
    <> Rule Map SNMP hosts
      <> Condition
        <
        <> And
          <> IsNode
          <> And
      <> MapTo
    <> Rule Map Unknown Host
    <> Rule Map Windows Host
    <> Rule Map Unix Host
```

1 error, 0 warnings, 0 others

Description	Resource	Path	Locat...	Type
Errors (1 item)				

Note: 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 to the data in the database, and the data changed and added during the mapping.

Creating a Synchronization Data Dump

A synchronization data dump contains the synchronized topology data in XML files.

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. Type the following command:

   ```
   ovconfchg -ovrg server -ns om.svcdiscserver.mapping -set
   opr.toposync.dumpData TRUE
   ```

2. Type the following command to restart the service discovery server:

   ```
   ovc -restart opcsvcdisc
   ```

Data Dump Example

Here is an example extract from a data dump after mapping has been performed:

```
<CI>
  <OMId>Root</OMId>
  <OMType />
  <Caption>Root</Caption>
  <Node>false</Node>
  <Service>false</Service>
  <OMAttributes />
  <CMDBId />
  <CMDBAttributes />
  <CMDBType />
  <RootContainerId />
  <Children>
    <RelationType>container_f</RelationType>
    <CI>
      <Context>operations-agent</Context>
      <OMId>03a2f7b2-ec88-7539-0532-c5b07da188dd</OMId>
      <OMType>agent</OMType>
      <Caption>Operations-agent on met</Caption>
      <Node>false</Node>
      <Service>false</Service>
      <OMAttributes>
        <AgentId>03a2f7b2-ec88-7539-0532-c5b07da188dd</AgentId>
        <Name>met.deu.hp.com</Name>
      </OMAttributes>
      <CMDBType>hp_operations_agent</CMDBType>
      <RootContainerId>{8BB8864B-CEC9-4B26-BD4C-
41F2C97C108E}</RootContainerId>
      <Dependencies>
        <RelationType>hosted_on</RelationType>
```

```
          <CI>
            <Context>VISPI</Context>
            <Context>nodegroups</Context>
            <OMId>{8BB8864B-CEC9-4B26-BD4C-41F2C97C108E}</OMId>
            <OMType>node</OMType>
            <Caption>met</Caption>
            <Node>true</Node>
            <Service>false</Service>
            <NodeGroupList>
              <NodeGroupID>OpenView_Windows2000</NodeGroupID>
              <NodeGroupID>Root_Nodes</NodeGroupID>
            </NodeGroupList>
            <MACAddressList />
            <OMAttributes>
              <AgentId>03a2f7b2-ec88-7539-0532-c5b07da188dd</AgentId>
              <CommType>HTTPS</CommType>
              <DiscoveryDomain>${DefaultDomain}</DiscoveryDomain>
              <Domain>deu.hp.com</Domain>
              <Name>met.deu.hp.com</Name>
              <OSType>Windows_32</OSType>
              <OSVersion>2000 (5.0)</OSVersion>
              <SystemType>x86/x64 Compatible</SystemType>
              <VirtualNodeType>0</VirtualNodeType>
            </OMAttributes>
            <CMDBId />

            <CMDBType>nt</CMDBType>
            <RootContainerId />
          </CI>
        </Dependencies>
      </CI>
    </Children>
  </CI>
```

Viewing a Synchronization Data Dump

To view synchronization data dumps, navigate to the directory:

`/var/opt/OV/shared/server/tmp/discovery`

The directory contains the following subdirectories:

- Contains the synchronization data after the CI data structure has been normalized. The data reflects what has been loaded from the source management server.
- Contains the synchronization data after the mapping rules have been executed on the normalized data.

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

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 48) for more information.

Matching Against Existing Attributes Only

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

Log Level Configuration

Topology synchronization logs details of the synchronization process in log files. You can change the level of detail for debugging purposes.

Service Discovery Server Log Level Configuration

The service discovery server supports the following log levels:

- Log level 1 logs errors only.
- Log level 3 logs errors and information (including raw data received from the agent). This is the default.
- Log level 10 logs tracing information for debugging purposes, for example method parameters.

To change the log level of the service discovery server:

1. Type the following command:

   ```
   ovconfchg -ovrg server -ns om.svcdiscserver -set LOG_LEVEL [1|3|10]
   ```

2. Type the following command to restart the service discovery server:

   ```
   ovc -restart opcsvcdisc
   ```

The service discovery server generates the following log file:

```
/var/opt/OV/shared/server/log/OvSvcDiscServer.log
```

Mapping Log Level Configuration

To change the log level of the mapping engine, follow these steps:

1. Open the following file in a text editor:

   ```
   /var/opt/OV/shared/server/conf/discovery/opr-svcdisc.properties
   ```

2. Locate the line starting with `loglevel=`
3. Set the log level to any of the following values (for example, `loglevel=INFO`):

   `DEBUG` designates fine-grained informational events that are most useful to debug an application.

   `INFO` designates informational messages that highlight the progress of the application at coarse-grained level.

`WARN` designates potentially harmful situations.

`ERROR` designates error events that might still allow the application to continue running.

`FATAL` designates very severe error events that will presumably lead the application to abort.

The mapping engine generates the following log file:

```
/var/opt/OV/shared/server/log/opr-svcdiscserver.log
```