



Hewlett Packard
Enterprise

HPE Operations Manager i

Software Version: 10.11

OMi Extensibility Guide

Document Release Date: 25 May 2016

Software Release Date: May 2016

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise 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. HPE 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 HPE required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© Copyright 2015-2016 Hewlett Packard Enterprise Development LP

Trademark Notices

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

AMD, the AMD Arrow symbol and ATI are trademarks of Advanced Micro Devices, Inc.

Citrix® and XenDesktop® are registered trademarks of Citrix Systems, Inc. and/or one more of its subsidiaries, and may be registered in the United States Patent and Trademark Office and in other countries.

Google™ and Google Maps™ are trademarks of Google Inc.

Intel®, Itanium®, Pentium®, and Intel® Xeon® are trademarks of Intel Corporation in the U.S. and other countries.

iPad® and iPhone® are trademarks of Apple Inc.

Java is a registered trademark of Oracle and/or its affiliates.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft®, Windows®, Lync®, Windows NT®, Windows® XP, Windows Vista® and Windows Server® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

NVIDIA® is a trademark and/or registered trademark of NVIDIA Corporation in the U.S. and other countries.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates.

Red Hat® is a registered trademark of Red Hat, Inc. in the United States and other countries.

SAP® is the trademark or registered trademark of SAP SE in Germany and in several other countries.

UNIX® is a registered trademark of The Open Group.

Documentation Updates

The title page of this document 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: <https://softwaresupport.hpe.com/manuals>.

This site requires an HP Passport account. If you do not have one, click the **Create an account** button on the HP Passport Sign in page.

Support

Visit the HPE Software Support website at: <https://softwaresupport.hpe.com>

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

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

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HPE 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 <https://softwaresupport.hpe.com> and click **Register**.

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

HPE Software Solutions & Integrations and Best Practices

Visit HPE Software Solutions Now at <https://softwaresupport.hpe.com/km/KM01663677> to explore how the products in the HPE Software catalog work together, exchange information, and solve business needs.

Contents

Part I: Extending OMi	13
Chapter 1: Prerequisites	14
Part II: Content Development	16
Chapter 2: Integration Content	17
Topology	17
Event Type and Health Indicators	17
Correlation Rules	18
Additional Event Processing	18
Tools	18
View Mappings	18
Graphs	19
Topology	19
Integrating New Applications	19
Enrichment Rules	28
Topology View of the New Application	29
Impact Propagation	30
Chapter 3: Event Type and Health Indicators	31
Mapping Events to CIs	31
Setting the Custom Message Attribute RelatedCiHint in OM	32
Setting the Custom Message Attribute SubCiHint in OM	33
RelatedCiHint Values	34
Creating Event Type and Health Indicators	35
Custom Message Attributes in OM Messages	44
Chapter 4: Correlation Rules and Mapping	46
Chapter 5: Additional Event Processing	48
Chapter 6: Tools	49
Chapter 7: View Mappings	50
Chapter 8: Performance Dashboards	51
Integrating Performance Data	51
Chapter 9: Packaging Content	52
Create RTSM Packages	53
Create a Content Pack	53
Upload the Content	56
Upload RTSM Packages	56
Upload Content Packs	56

Part III: Populating the Run-time Service Model	57
Chapter 10: Topology Synchronization	58
Topology Synchronization Overview	58
Topology Synchronization Architecture	60
Synchronization Packages and Mapping	62
Scripting and Topology Data	62
CI Resolution Using a Mapping Table	62
Topology Synchronization File Locations	64
Topology Synchronization Settings	65
Managing Synchronization Packages	65
Uploading OM SPI Service Type Definitions to the Database	66
Chapter 11: Synchronization Packages	67
Synchronization Packages Overview	68
Standard Out-of-the-box Synchronization Packages	69
Additional Out-of-the-box Synchronization Packages	69
Package Descriptor File: package.xml	70
Mapping Files	71
Context Mapping (Filtering): contextmapping.xml	71
Type Mapping: typemapping.xml	71
Attribute Mapping: attributemapping.xml	71
Relation Mapping: relationmapping.xml	71
Configuring Topology Synchronization: ACME Example	72
Configure Package Descriptor File: package.xml	73
Configure Context Mapping (Filtering) File: contextmapping.xml	73
Configure Type Mapping File: typemapping.xml	74
Configure Attribute Mapping File: attributemapping.xml	75
Configure Relation Mapping File: relationmapping.xml	76
Customizing Synchronization and Scripting	78
Synchronization Package Locations	78
Chapter 12: Scripting	79
Groovy Scripts	79
Enabling and Disabling Scripts	80
Groovy Script Location	80
Script Variables	81
Handling Errors	82
Sample Script: preUpload.groovy	82
Chapter 13: Testing and Troubleshooting	84
Validating XML Configuration Files	84
XSD Files	85
Validating Files Automatically	85
Validating Files Manually	86
Dumping Synchronization Data	87
Creating a Synchronization Data Dump	87
Data Dump Example	87
Viewing a Synchronization Data Dump	90

Validating Mapping Rules	91
Writing Rules	91
Simplifying Rule Development	91
Avoiding Complex XPath Queries	91
Matching Against Existing Attributes Only	91
Avoiding Broad XPath Expressions	92
Log Level Configuration	93
Service Discovery Server Log Level Configuration	93
Mapping Log Level Configuration	93
Troubleshooting, Common Issues, and Tips	94
Limitations	95
Delta Detection Limitations	95
Topology Synchronization Limitation	95
Chapter 14: Mapping Engine and Syntax	96
Common Mapping File Format	96
Mapping File Syntax	97
Rules	97
Rule Conditions	97
Operator Elements	99
Operand Elements	102
Mapping Elements	109
Filtering	109
Type Mapping	110
Attribute Mapping	111
Attribute Mapping Example	114
Relation Mapping	115
XPath Navigation	117
Data Structure	118
Example of an XPath-Navigated Data Structure	120
XPath Expressions and Example Values	120
Part IV: Event Processing Interface	122
Chapter 15: Event Processing Interface	123
Event Processing Interface and Scripts	123
Entry Points for Running EPI Scripts	124
Specifying a Script	124
EPI Script Execution	125
EPI Script for Event Enrichment	125
Tips and Limitations	131
Creating Scripts	131
Chapter 16: Scripts For Custom Actions	132
Specifying Custom Actions Scripts	132
Creating Scripts	133
Chapter 17: Creating EPI and Custom Action Scripts	134

Script Definition Attributes	134
Groovy Script API	134
Script Definition Format	135
EPI Groovy Script Samples	135
SimpleExampleEPI.groovy	136
RegExample.groovy	139
ResolveLocationFromDB.groovy	140
Custom Actions Groovy Script Samples	142
SimpleExample.groovy	142
Chapter 18: EPI Troubleshooting	144
Log Files	144
Debugging	144
Part V: Integrating the OMi UI with Other Applications	145
Chapter 19: URL Launch of the Event Browser	146
Specifying a URL Launch	146
Default URL Launch	146
Specifying Optional Parameters	146
Parameters and Parameter Values	147
Defining Columns	149
Setting Filters	151
Filtering by String Attributes	152
Filtering by Time Properties	153
Filtering by Priorities	154
Filtering by CIs and CI Types	154
Filtering by Global CI ID	154
Filtering by ETIs and ETI Values	155
Filtering by Other Event Characteristics	155
URL launch of the Event Details	156
Part VI: Automating Operator Functions and Event Change Detection	157
Chapter 20: Event Web Service Interface Reference	158
How to Access the Event Web Service	158
How to Detect New Events	160
Receiving Events as Atom Feeds	161
Specifying Parameters to Filter the Event List	162
How to Detect Event Changes	162
How to Modify Events	162
Modifying Events Using a REST Client	162
Modifying Events Using the RestWsUtil Utility	166
How to Create New Events	167
Example: How to Create a New Event	167
Advanced Modification of Event Properties	168
Bulk Update of Events	169

Bulk Insert of Events	170
Chapter 21: Event Web Service Query Language	171
HTTP Query Parameters	171
List of HTTP Query Parameters	172
Filtering by Date and Time: watermark	173
Filtering by Event Attributes: query	174
Paging	175
Ordering	176
Data Inclusion	177
Media Type	178
Query Filter Criteria Properties	178
Operator Aliases	182
Value Types	183
Query Using the POST Method	183
URL Escape Codes	184
White Spaces in URLs	185
Complex Attributes	185
Editable Properties	186
History Lines	187
Recorded Property Changes	187
Event Changes List	188
File Locations	189
Chapter 22: RestWsUtil Command-Line Interface	190
Part VII: Integrating External Event Processes	202
Chapter 23: Forwarding Events and Synchronizing Event Changes	203
Event Forwarding and Synchronization Process	203
Forwarding Events and Event Changes to an External Event Process	203
Receiving Event Changes Back from an External Event Process	205
Performing a URL Launch of the Event Browser from an External Application	207
Chapter 24: Integrating External Event Processes Using Groovy Scripts	208
Sample Groovy Script: Logfile Adapter	208
Configure a Connected Server using the Logfile Adapter	208
Configure an Event Forwarding Rule	210
Groovy Script Interface	211
Groovy Script Methods	212
init, destroy, and ping Methods	212
Methods for Forwarding Events to a Connected Server	214
Methods for Receiving Synchronization Data from a Connected Server	220
Additional Methods	222
Methods for Retrieving Data from a Connected Server	224
Method for Supplying Data to a Connected Server	225
Management of Groovy Script Methods	226
Chapter 25: Event Synchronization Web Service Interface	227

Chapter 26: Integrating External Event Processes: Frequently Asked Questions	228
Getting Started	228
Groovy Scripts and Programming	231
Integration Script Methods	233
Event Properties	237
Troubleshooting	238
Logging	239
Chapter 27: Integrating an External Event Processing Service Defined by a WSDL	240
Generate Java Code from WSDL	240
Configure the External Event Processing Application as a Connected Server	242
Test the External Event Creation	243
Chapter 28: Error Handling	244
Chapter 29: Service Manager Integration	246
Part VIII: Web Service Interfaces	247
Chapter 30: Reference Information For All Web Services	248
Authentication	248
Error Handling	249
Error Verbosity	249
Securing Modify Operations	250
Environments with CA SiteMinder	256
Session Handling	256
Settings	256
Chapter 31: Action Web Service Interface	258
Action Web Service Request Reference	260
Examples	264
Chapter 32: Downtime Web Service Interface	268
Downtime Schedule Examples	271
Example of a Downtime Schedule with One Occurrence	271
Example of a Weekly Downtime Schedule	271
Example of a Monthly Downtime Schedule	272
Downtime REST Examples	272
Importing Downtime Data From an External Source	273
Import Example	273
Chapter 33: Event Synchronization Web Service Interface	275
(prerequisite) Configure a Connected Server	275
Forward Events and Event Changes from OPR Client	275
Synchronize Event Changes Back from External Client	277
Event Updates	278
Event List Updates	279
Event List Changes	280
Event Attributes that Support Back Synchronization	281
Event Update: Logfile Adapter Examples	281

Event Change Creation: Logfile Adapter Examples	283
Event Update to event_list Example	291
Event Submit Examples	292
Submit a New Event	292
Submit a New Event with Synchronization Requested	292
Submit a New Event with Synchronization and Transfer Control Requested	292
Submit a List of New Events	293
Event Change Creation for event_change_list Examples	293
Submit a New Event Change	293
Submit a List of New Event Changes	294
Chapter 34: Event Web Service Interface	295
Chapter 35: Monitoring Automation Web Service Interface	296
Using the Monitoring Automation Web Service Interface	296
Monitoring Automation Web Service Interface Reference	302
Examples	307
Tool Execution Web Service Interface	311
Tool Execution Web Service Request Reference	312
Chapter 36: User Management Web Services Interface	322
User Management Web Services Request Reference	324
Examples	328
Part IX: Groovy Scripts	335
Chapter 37: Best Practices	337
Chapter 38: Development and Deployment of Scripts	341
Event Processing Interface Scripts	341
Custom Action Scripts	347
Certificates Scripts	349
Service Health Scripts	351
Topology Synchronization Scripts	352
Event Forwarding Scripts	354
External Instruction Retrieval Scripts	357
Chapter 39: Reference Information	362
The Groovy Console	362
Available APIs	365
The opr-script Command-Line Interface	367
Part X: Service Health	372
Chapter 40: Service Health Rules API	373
API Group and Sibling Rule	374
API Sample Rule	376
API Duration-Based Sample Rule	377
Creating Rules with the Rules API	378
How to Define an API Rule in the KPI and Health Indicator Customizations per CI Page	379

How to Create a Text File-Based API Rule	380
How to Define an API Rule in the Rule Repository	383
How to Work with Tooltip Entries	384
How to Write to Log Files From the Rules API Code	385
How to Include a CI Property in Rules API Calculations	386
Examples - API Sample Rule	386
Example - Average Availability Rule	386
Example - Average Performance Rule	387
Example - Average Performance Rule Using a Rule Parameter Filter	388
Examples - API Group and Sibling Rule	389
Example - Worst Child Rule	390
Example - Worst Sibling Status Rule	391
Example - Specific Child CI Group Rule	392
Example - Sibling Rule Based on Availability and Performance KPIs	392
Example - Group Average Value by CI Type	393
Example - Worst Health Indicator Rule	394
Example - Using Groovy Closure	395
Chapter 41: Service Health External APIs	397
Retrieve Indicator Data API	397
Reset Health Indicator State API	402
Service Health Database Query API	403
Send Documentation Feedback	406

Part I: Extending OMi

This guide provides information on how to customize and extend the functionality of HPE Operations Manager i (OMi).

The guide is divided into the following sections:

- ["Content Development" on page 16](#) provides steps required for content developers to add management capabilities for a new application, using the fictitious ACME environment as a simple example. The example illustrates the various integration steps required in order to make management information of the new application available in OMi.
- ["Populating the Run-time Service Model" on page 57](#) provides information for developers to create their own topology synchronization mapping rules, to augment the out-of-the-box mapping rules to populate the Run-Time Service Model (RTSM) with configuration items (CIs) and CI relationships from nodes and services in OM.

The ACME environment example, introduced in ["Content Development" on page 16](#), is developed further to illustrate how to create topology synchronization rules specific to a particular service model.

- ["Event Processing Interface" on page 122](#) describes the role of event processing scripts and custom actions for modifying and enhancing events during event processing.
- ["Integrating the OMi UI with Other Applications" on page 145](#) describes how to integrate parts of the OMi user interface with an external application using a drill-down URL launch.
- ["Automating Operator Functions and Event Change Detection" on page 157](#) provides integrators with information to allow them to programmatically automate operator functions and detect event changes using the Event Web Service. Everything that an operator can do in the console while working on events can be done programmatically, to improve efficiency.

The main use case for the Event Synchronization Web Service interface is to synchronize events and changes to events with an external manager such as an ITIL incident manager, for example, HPE Service Manager or BMC Remedy Service Desk.

- ["Integrating External Event Processes" on page 202](#) describes the Event Synchronization Web Service interface that enables integrations with external applications, where the aim is to integrate external processes into event processing. The interface enables notification of forwarded events and subsequent changes to be received programmatically.
- ["Web Service Interfaces" on page 247](#) describes several web services that allows integrators to automate OMi functions.
- ["Groovy Scripts" on page 335](#) describes how Groovy scripts are developed and deployed to implement customizations.
- ["Service Health" on page 372](#) describes Service Health rules APIs and Service Health external APIs.

Chapter 1: Prerequisites

The prerequisites for extending and customizing OMi are as follows:

- You should be familiar with the relevant HPE Software products and components, including the associated documentation. As an example, the following list includes the essential steps required to extend the integration between OM:

- **Run-Time Service Model (RTSM)**. Detailed administrative and operational knowledge is assumed. For details about RTSM concepts, including CI types, CI attributes, and views, see the *Modeling Guide*. For details on importing data from Excel Workbooks and other external sources, see the *HPE Universal CMDB Discovery and Integration Content Guide*.
- **HP Operations Manager (OM)** for Windows and UNIX. In particular, take a look at the Service Tree in OM, and decide which services should be synchronized with the RTSM.

Note: Services that are used only to build up the parents of a tree (for example, Applications, and Systems Infrastructure) probably do not need to be synchronized, because the RTSM uses graphs and does not require a tree structure.

- **Operations Manager i (OMi)**. For example, the following steps must be executed in OMi:
 - i. Using the OM service tree as reference, mark the services that you want to synchronize with a Context in the `contextmapping.xml` file.
 - ii. Choose the CI type for each OM service. Enter the mapping from an OM service to a CI type in the `typemapping.xml` file.
 - iii. Make sure that all required CI Attributes (key attributes) are filled from OM service attributes in the `attributemapping.xml` file.
 - iv. Create all relationships in accordance with the RTSM model in the `relationmapping.xml` file.

Note: Some CI types (for example, Running Software) have keys that consist of simple CI attributes and relations. For these CI types, the `attributemapping.xml` must set all key attributes, and the `relationmapping.xml` must create the correct relations, to instantiate the CI correctly.

You can refer to the standard topology synchronization packages, such as `default`, `operations-agent`, and `nodegroups`, as examples of how to work with synchronization packages. You can find these under:

```
<OMi_HOME>/conf/opr/topology-sync/sync-packages
```

This information is contained in the *OMi Extensibility Guide*.

- **HPE Operations Manager i (OMi)**.
- Knowledge of Groovy scripting and syntax is required for creating Groovy scripts. Groovy is supported for scripting, and Groovy scripts are used in the topology synchronization process, for Event Processing Interface (EPI) and custom action scripts, for automating operator functions, and for integrating external event processes.

- Knowledge of the XPath query language is required to navigate through the CI data structure in the mapping engines.

Part II: Content Development

This section describes the steps required to:

- Customize the existing monitoring configuration data supplied out-of-the-box according to customer requirements.
- Add monitoring capabilities for new applications and elements of the IT environment.

The steps are illustrated using a simple example for the ACME environment.

This section is structured as follows:

- ["Integration Content" on page 17](#)
- ["Topology" on page 19](#)
- ["Event Type and Health Indicators" on page 31](#)
- ["Correlation Rules and Mapping" on page 46](#)
- ["Additional Event Processing" on page 48](#)
- ["Tools" on page 49](#)
- ["View Mappings" on page 50](#)
- ["Performance Dashboards" on page 51](#)
- ["Packaging Content" on page 52](#)

Chapter 2: Integration Content

When you integrate a new application area into a monitoring solution, the following areas are important to the integration:

- ["Topology" below](#)
- ["Event Type and Health Indicators" below](#)
- ["Correlation Rules" on the next page](#)
- ["Additional Event Processing" on the next page](#)
- ["Tools" on the next page](#)
- ["View Mappings" on the next page](#)
- ["Graphs" on page 19](#)

Topology

Topology data is contained in a Run-time Service Model (RTSM). The RTSM contains definitions of configuration item types (CI types), and how those CI types can potentially be related to other CI types. Configuration items (CIs) are instances of CI types.

To integrate a new application into an Operations Manager i monitoring solution, and create a topology view of the new application, it may be necessary to:

- Create new CI types for the new application.
- Identify the key attribute values for the new CI types.
- Establish the relationships (for example, membership, dependency, and composition relationships) for the new application.
- Create CIs and CI relationships in the RTSM.

The effort required to integrate the topology data for a new application depends on what data already exists. For example, integrating an application where you can re-use existing RTSM objects will require less effort than integrating one where you need to start at the beginning, and define all the RTSM CI types and their relations.

For more details about the role of topology data in integrating a new application, see ["Topology" on page 19](#).

Event Type and Health Indicators

To benefit from the advanced health-based monitoring features for your new application area, it is necessary to:

- Populate the RTSM with CIs. Events must be mapped to the correct CIs in the RTSM.
- Transform received events into data about the service health of CIs. This involves analyzing incoming events for the various CI types and creating meaningful event type indicators (ETIs) and

health indicators (HIs).

- Assign HIs to health-based key performance indicators (KPIs).

For more details, see ["Event Type and Health Indicators" on page 31](#).

Correlation Rules

The event management process is simplified for you by not only consolidating events from all sources in a central console, but also correlating events using topology-based event correlation (TBEC).

TBEC rules make associations between a known root-cause event and related symptom events. Symptom events are those events that occur as a consequence of the root-cause event. TBEC greatly reduces the number of events displayed in the browser by avoiding duplication and overload. This enables you to manage the problem of large numbers of similar (related) symptom events in a large network.

HI and ETI values are used to represent the events that have an impact on the configuration item types specified in the TBEC rule. These values are used to create correlation rules.

For more details about correlation rules, see ["Correlation Rules and Mapping" on page 46](#).

Additional Event Processing

You can perform additional event processing to modify and enrich events using Groovy scripts.

The Event Processing Interface (EPI) lets you run a number of user-defined Groovy scripts during event processing.

You can also configure custom actions to apply to events.

For more details about additional event processing, see ["Additional Event Processing" on page 48](#).

Tools

You can configure tools to help you manage and monitor specific events, and to solve event-related problems associated with your new application area.

For an example of a tool configured for the new application, see ["Tools" on page 49](#).

View Mappings

You can map configuration item types to RTSM views using the RTSM Modeling Studio, so that views are available for selection and use in the Health Top View pane.

For more details about mapping CI types to RTSM views, see ["View Mappings" on page 50](#).

Graphs

Graphs and charts provide you with additional data to help you visualize and analyze performance-related problems and trends affecting the configuration items impacted by an event for your new application area.

For more information about graphs, see ["Performance Dashboards" on page 51](#).

Topology

This section shows how to integrate a new application, and create a topology view of the new environment, using an example application environment called "ACME".

This section is structured as follows:

- ["Integrating New Applications" below](#)
- ["Enrichment Rules" on page 28](#)
- ["Topology View of the New Application" on page 29](#)
- ["Impact Propagation" on page 30](#)

Integrating New Applications

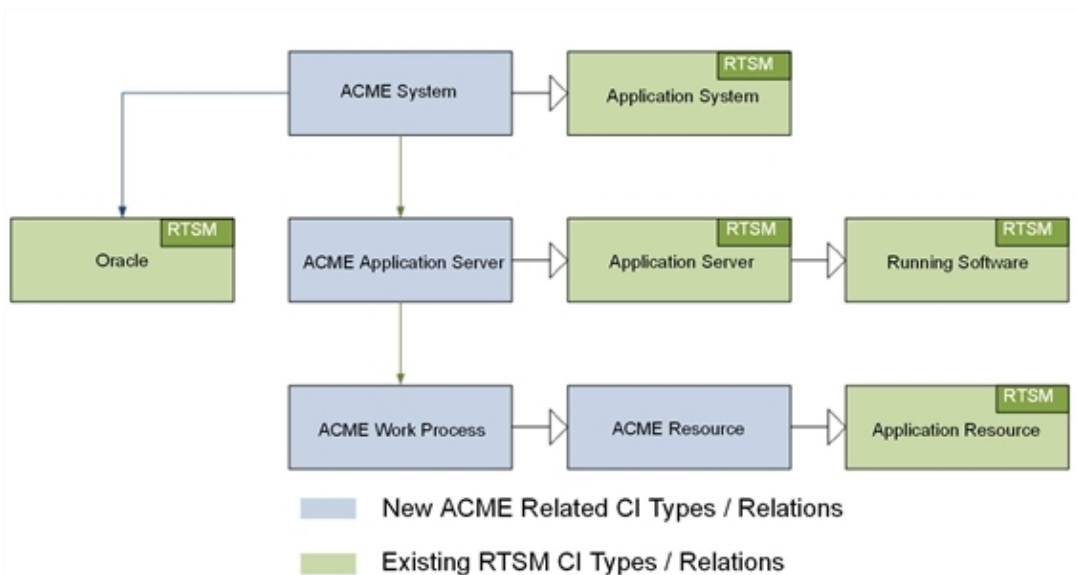
To integrate a new application into an OMi monitoring solution and create a topology view of the new application, it is necessary to:

- Create new CI types for the new application (if existing CI types cannot be reused)
- Set the key attribute values for the new CI types.
- Establish the relations for the new application.
- Create CIs and CI relationships in the RTSM.

Create New CI Types for the New Application

The first step in integrating your new application is to create new CI types for the application.

The following graphic shows the topology model of the “ACME” environment.



In this example ACME environment, an **ACME system** contains various **ACME application servers**. These application servers employ **ACME work processes** to execute user requests.

The ACME environment uses an **Oracle database** to store all information. The graphic shows four new CI types, depicted in blue:

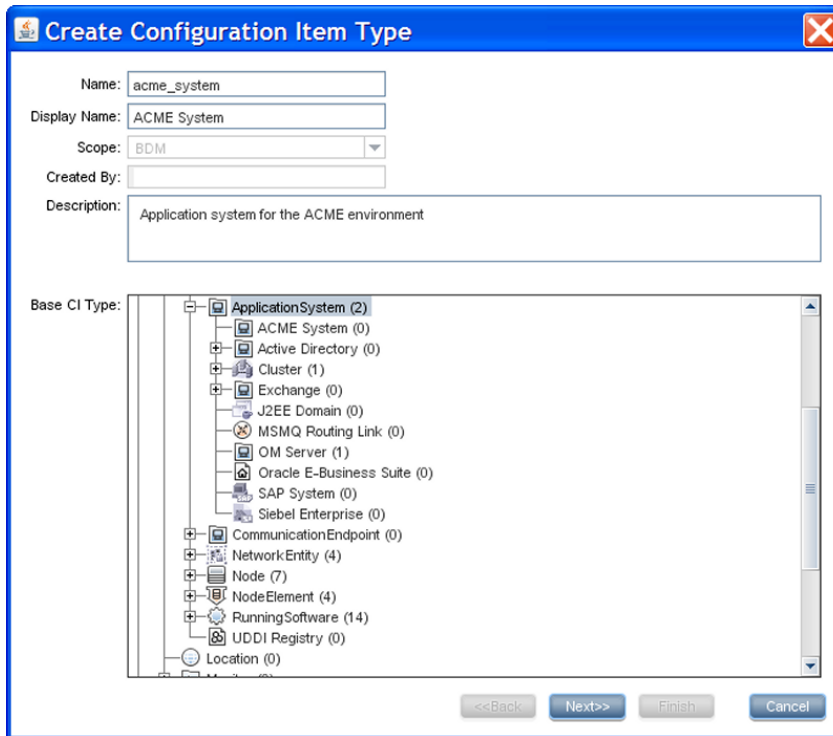
- ACME System
- ACME Application Server
- ACME Work Process
- ACME Resource

These new CI types are child elements of CI types that already exist in the RTSM (shown in green).

Comprehensive details about how to create new CI types and how to work with the concepts of the RTSM are described in the *Modeling Guide*.

To create a new CI type, do the following:

1. Navigate to the CI Type Manager:
RTSM Administration > Modeling > CI Type Manager
2. In the CI Types pane, activate the CI Types tree by selecting **CI Types** from the drop-down menu.
3. In the CI Types tree, navigate to the folder where you want to add your new application, for example:
Configuration Item > Infrastructure Element > Application System
4. Right-click, and click the New (🌸) button. The Create Configuration Item Type dialog box opens.



5. In the Name field, enter the name of the CI type to be created: **acme_system**.
In the Display Name field, enter the display name of the CI type: **ACME System**.
Optional. In the Description field, enter a description of the CI type you are creating.
Click **Next** to proceed to the next page of the Create Configuration Item Type dialog, where you set the key attributes for the new CI you created, as described in "[Set Key Attribute Values for New CI Types](#)" on the next page below.

Set Key Attribute Values for New CI Types

It is essential to identify the new CI types with unique key attributes. By setting unique key attributes, you can make sure there are no duplicate CIs created, for instance, by different discovery sources.

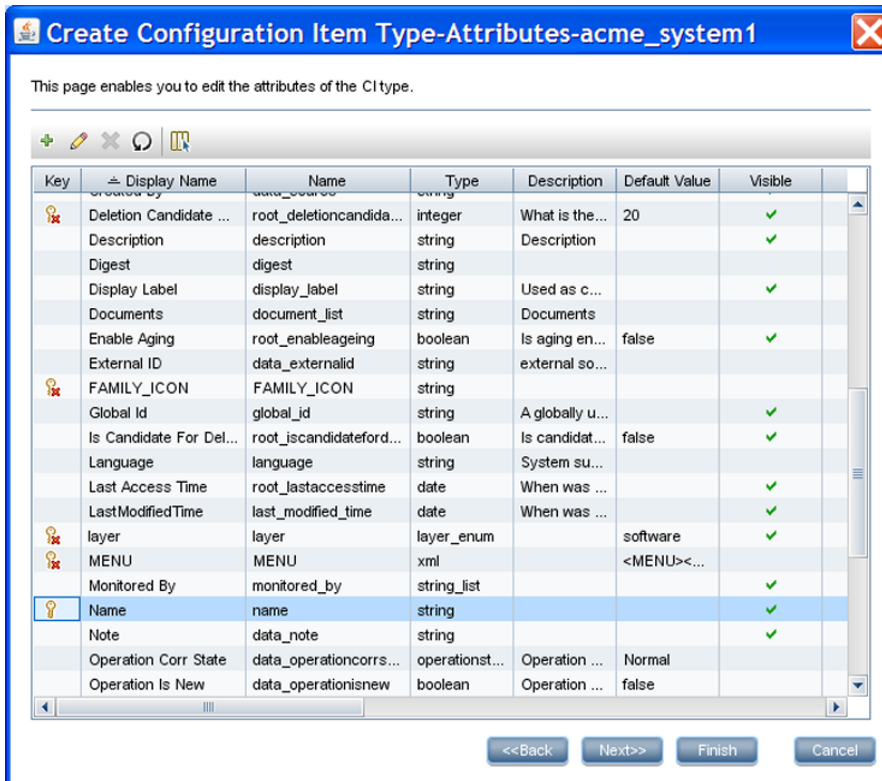
The following table shows a list of potential key attributes for an ACME environment.

List of CI Type Attributes for the ACME Environment

CI Type Display Name	Attribute	Description	Value
ACME System	name	Name of an ACME system	System ID
ACME Application Server	name	A unique name that identifies an ACME server within an ACME System landscape	ACME server name
	root_container	The container host	CI reference (CI ID) of the host
ACME Work Process	name	A logical single-instance representation of a certain type of work process	Work process category; batch, dialog or spool

1. In the Create Configuration Item Type-Attributes dialog, set the key attributes for the CI type, for example, ACME System.

To identify your new CI type with an existing attribute, click in the column marked **Key** on the same row as the attribute you want to set as a key attribute. A small key icon then appears. (Click again if you do not wish to set that attribute.)



- In addition to setting key attributes for the new CI type, you may want to create your own attributes used solely for that CI type.

To create a new attribute:

- a. In the Create Configuration Item Type-Attributes dialog, click the Add (+) button. The Add Attribute dialog box opens.

The screenshot shows the 'Add Attribute' dialog box with the following fields and values:

- Attribute Name: acme_instance_number
- Display Name: ACME Instance Number
- Scope: BDM
- Description: Attribute solely for use with the ACME environment
- Attribute Type: Primitive (selected), Enumeration/List (unselected)
- Value Size: (empty)
- Default Value: (empty)

- b. In the Attribute Name field, enter the name of the new attribute: `acme_instance_number`
In the Display Name field, enter the display name of the new attribute: **ACME Instance Number**
In the Scope field, choose the scope: **BDM**
Optional. In the Description field, enter a description for the new attribute.
 - c. Set the attribute type, and if applicable, fill out the Value Size and Default Value fields.
 - d. Click **OK**.
3. Set the newly created attribute as a key attribute for the ACME System CI type, as described in step 1.
4. Click **Finish**.

Establish Relationships for the New Application

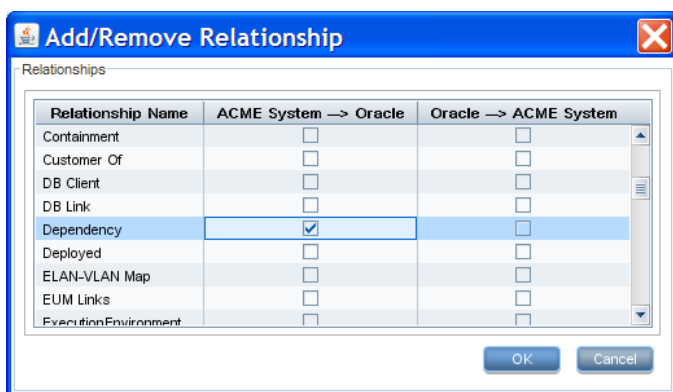
The next step in integrating the new application is to establish the relationships for the new application.

In the out-of-the-box RTSM model, there is a **membership** relationship between the ACME system and the ACME application server. Our example ACME system depends on an Oracle database. This **dependency** relationship, however, does not exist in the out-of-the-box RTSM model. You must create this relationship using the CI Type Manager:

RTSM Administration > Modeling. CI Type Manager

To create the dependency relationship between the ACME system and the Oracle database, do the following:

1. In the CI Type Manager, select **ACME System** and **Oracle**.
2. Right-click, and select **Add/Remove Relationship**.
3. In the Add/Remove Relationship dialog, check the box for Dependency in the **ACME System > Oracle** column, to establish the relationship that our ACME system depends on an Oracle database.



4. Click **OK**.

Creating CIs and CI Relationships in the RTSM

The next step in integrating the new application is to create CIs and CI relationships in the RTSM.

There are three methods to create CIs and CI relationships:

- Create the CIs and CI relationships using discovery features of the RTSM and HPE Data Flow Management (DFM). See ["Creating CIs Using RTSM/HPE Data Flow Management Discovery" on the next page](#).
- Create the CIs and CI relationships using topology synchronization. Topology synchronization reuses the OM service model to create corresponding CIs and CI relationships. This, of course, requires that a suitable service model in OM has been created. See ["Creating CIs Using Topology Synchronization" on page 27](#).

For more details about topology synchronization, see ["Populating the Run-time Service Model" on page 57](#).

- Create CIs manually in the RTSM. This option is really only practical if you need to create just a limited number of CIs, and that these CIs are stable in nature, and are not expected to change.

See also "[Considerations when Choosing a Discovery Method](#)" on page 28.

Creating CIs Using RTSM/HPE Data Flow Management Discovery

HPE Data Flow Management automatically discovers and maps logical application assets in Layers 2 to 7 of the Open System Interconnection (OSI) Model. The discovery technology is based on discovery patterns.

The Data Flow Management licensing structure is as follows:

- UCMDB Foundation License. The Foundation license includes UCMDB as the backbone component for BTO products. This version enables data flow between multiple instances of UCMDB, and integration with BTO products to enable solution deployment.
- UCMDB Integration License. The Integration license adds third party integrations on top of the UCMDB Foundation license.
- UCMDB DDM Advanced License. The DDM Advanced license includes all discovery capabilities to discover the IT infrastructure elements and feed that information as CIs and Relationships to the RTSM. DDM Advanced license also allows users to extend the RTSM model and write their own discovery patterns.

Using Data Flow Management, it is also possible to query external data sources:

- Comma Separated Value (CSV) Files
- Properties Files
- Databases

For more details, see the *Data Flow Management Guide* and the *RTSM Developer Reference Guide*.

Creating CIs Using Topology Synchronization

Many OM customers use OM service views or the Service Navigator to visualize dependencies between IT resources and IT services to their operators.

If you take this route, then you would use either the service discovery features of the Operations Smart Plug-ins or your own discovery mechanisms to create the service tree.

If this is the case, then you can use the topology synchronization feature to create CIs and CI relationships, based on the OM service model and topology synchronization mapping rules. Out-of-the-box mapping rules are provided that are able to map service models created by the following Operations Smart Plug-ins, all of which are enabled to work with OMi:

- Operations Smart Plug-in for Databases (Oracle and MS SQL Server only)
- Operations Smart Plug-in for IBM WebSphere Application Server
- Operations Smart Plug-in for BEA WebLogic Application Server
- Operations Smart Plug-in for Microsoft Active Directory
- Operations Smart Plug-in for Microsoft Exchange Server
- Operations Smart Plug-in for Virtualization Infrastructure
- Operations Smart Plug-in for Systems Infrastructure
- Operations Smart Plug-in for Cluster Infrastructure

You may have invested significant effort in creating your own custom service model, together with a corresponding manual or automatic service model creation process. You can reuse that model to create corresponding RTSM configuration items automatically. All you need to do is to write corresponding topology synchronization mapping rules for your model.

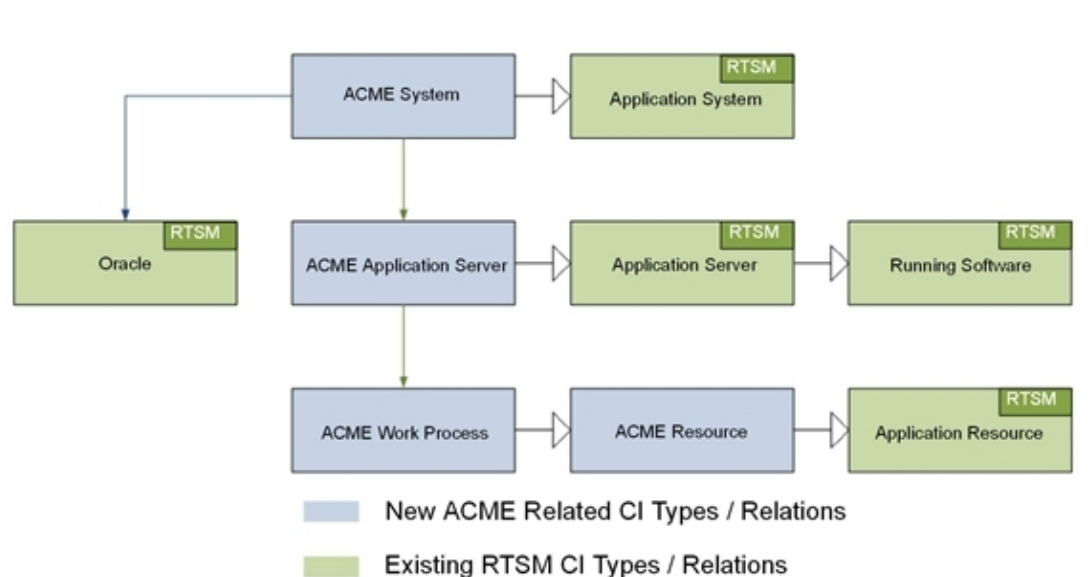
For more details about topology synchronization, see ["Populating the Run-time Service Model" on page 57](#).

Considerations when Choosing a Discovery Method

The following considerations are useful in helping you decide which discovery option to use to populate the RTSM.

- When to Use Data Flow Management
Use Data Flow Management if:
 - You already use Data Flow Management to populate the RTSM, or are planning to use it.
 - You do **not** already have a service model in OM. We recommend using Data Flow Management, as it is the preferred discovery method for populating the RTSM.
- When to Use Topology Synchronization
Use out-of-the-box topology synchronization rules provided in the synchronization packages if:
 - You are not using (and have no plans to use) Data Flow Management to populate the RTSM, *and*
 - You already have a service model in OM containing services corresponding to your ACME topology.
- When to Create CIs Manually
You can create CIs manually if the CIs you want to create are limited in number, and are stable in nature, so are unlikely to change.

Enrichment Rules

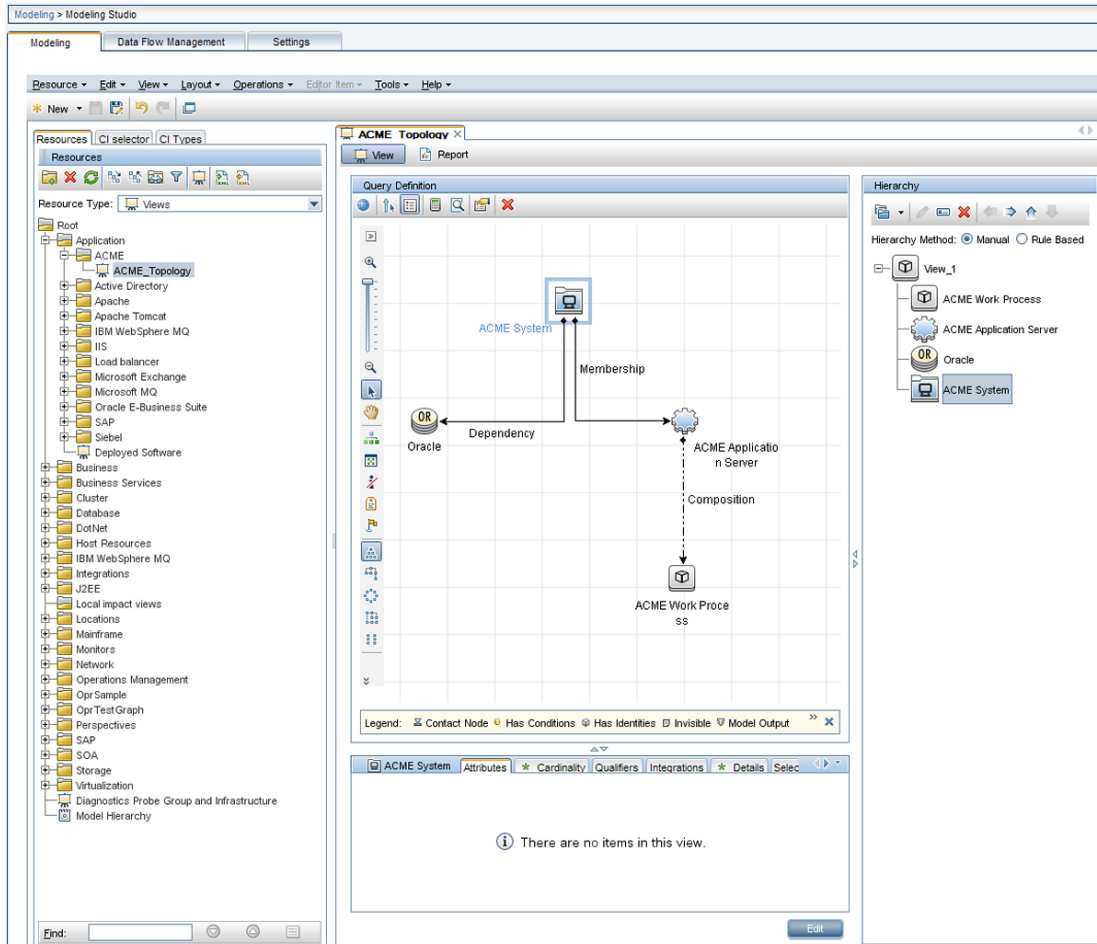


The illustration shows the ACME model, where there is a cross-domain relation between an ACME system and an Oracle database. If there are insufficient key attributes for the ACME discovery to create the Oracle database CI, this dependency relation can be generated by an enrichment rule.

For more information on how to create and maintain enrichment rules, see the *Modeling Guide*.

Topology View of the New Application

Use the RTSM Modeling Studio to create a view to display the ACME topology. The following screenshot shows the ACME topology view in the RTSM Modeling Studio.



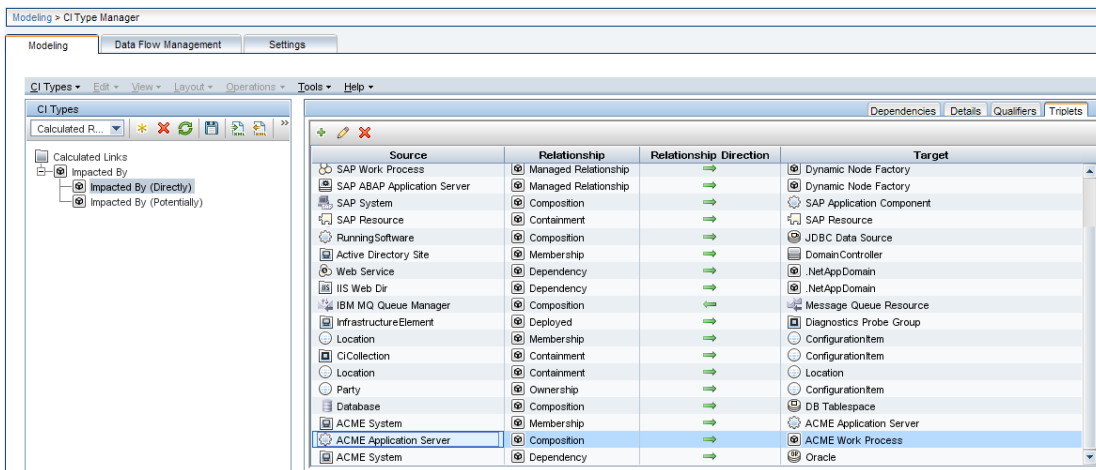
ACME topology view in RTSM Modeling Studio

Impact Propagation

Impact modeling is performed using calculated relations. Impact relationships are important for KPI calculations. For a detailed description of impact modeling in the RTSM, including concepts, see the *Modeling Guide*.

Note: You can create a specific view to verify the impact propagation within your topology. Create a new view and choose `impacted_by` as the relationship type between each CI type.

The following graphic shows a calculated relation (a triplet), which uses a propagation rule to create an impact relation between the ACME Application Server and the ACME Work Process.



Creating an impact relation between the ACME Application Server and the ACME Work Process.

Chapter 3: Event Type and Health Indicators

This section shows how to enrich events for your new application to benefit from the advanced event correlation and health-based monitoring features offered as part of an OMi solution.

In our ACME example, we assume that OM provides a Smart Plug-In for an ACME landscape. If there were no HPE Operations Smart Plug-in, you would have to analyze the ACME application itself to determine its interfaces, and also create policies to monitor such a landscape.

To be able to use the advanced event correlation and health-based monitoring capabilities, it is necessary to:

- Populate the RTSM with CIs and map OMi events to the correct CIs in the RTSM.
- Analyze incoming events for the various CI types and create meaningful event type indicators (ETIs) and health indicators (HIs).
- Assign HIs to health-based key performance indicators (KPIs).

Mapping Events to CIs

The fundamental requirement underlying all advanced event and health monitoring features provided by OMi is that events are mapped to the correct CIs in the RTSM.

When events in OMi are mapped to the correct CIs:

- Event type and health indicators can provide useful information about problems.
- Correlation rules trigger properly.
- The Health Perspective view shows informative health and KPI data.

Therefore, it is essential that:

- The RTSM is populated with CIs.
- Adjustments are made (if necessary) to the event integrations in such a way that events can be mapped to these CIs.

Smart mapping technology is employed to map events to CIs. The system looks for hints in certain event attributes, compares these hints with CI attributes, and then maps an event to the best matching CI.

Most events contain at least the DNS name of the affected host (in the OM message node name field). As this DNS name is used as one possible hint, the smart mapping will almost always be able to map an incoming event to at least the host CI (assuming, of course, that the host CI exists in the RTSM).

However, to make use of the complex IT topology model, it is important to map:

- Database events to corresponding database CIs.
- Events related to other applications to their respective CIs.

To achieve this, an evaluation is made of the following additional event attributes:

- Application
- Object

- OM service ID or the CI Resolution hint attributes
If the CI Resolution hint attribute is set, the OM service ID attribute is ignored.

The more identifying hints that can be provided in these attributes, the higher the likelihood that the event will be mapped to the correct CI.

Hints must be specified using a certain format to allow smart mapping to identify what belongs to one attribute.

The default format is:

- `<hint 1>:<hint 2>:...:<hint n>` for the Application and Object attribute
- `<hint 1>:<hint 2>:...:<hint n>@@<hostname>` for the OM service ID and CI Resolution hint attributes

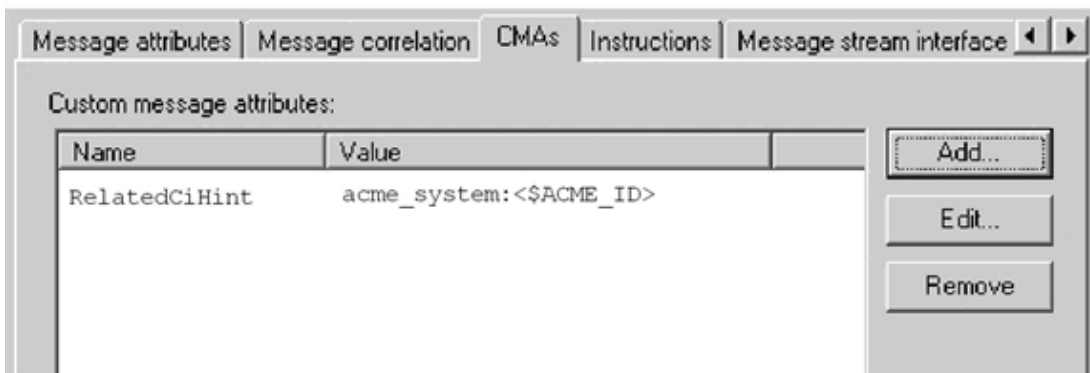
The separator (:) can be configured in the OMi Infrastructure Settings. For details, see OMi Help.

All the hints are then evaluated to find a matching CI in the RTSM. The hints in the Application, Object and OM service ID attributes are evaluated for backward compatibility reasons. In the past, many Operations Smart Plug-ins used these fields to transport information about which object (or configuration item, in RTSM terms) the event is related to. If this information is sufficient to identify the correct CI, then there is no need to change anything.

However, if you find that an event is related to an incorrect CI, then you should set the necessary hints in the CI Resolution hint attribute. You can do this by setting a custom message attribute (CMA) called `RelatedCiHint` in the OM message in OM.

Setting the Custom Message Attribute `RelatedCiHint` in OM

The following graphic shows an example of how to set the CMA `RelatedCiHint` in the OM message:



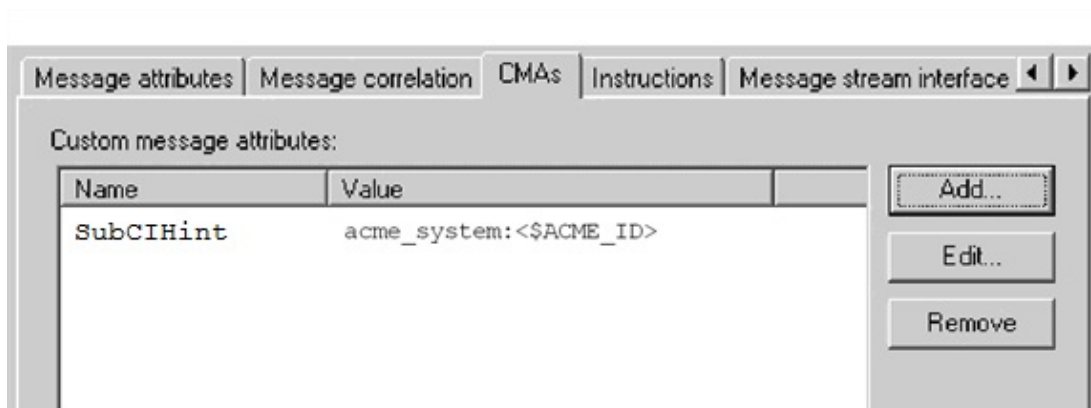
Setting custom message attribute `RelatedCiHint` in the OM message

The following considerations regarding best practice are important to bear in mind when setting the `RelatedCiHint` variable:

- The CMA `RelatedCiHint` should have sufficient hints to find the corresponding CI.
- It is necessary to differentiate between CIs that have a composition relationship to a host, and those that do not have such a relation.

Setting the Custom Message Attribute SubCiHint in OM

If you want to distinguish a status for a CI type that is not modeled in the RTSM, you can refer to the CI that contains the CI subtype you want to track, and then pass the CMA subCiHint. The following graphic shows an example of how to set the CMA SubCiHint in the OM message:



Setting custom message attribute SubCiHint in the OM message

The following considerations regarding best practice are important to bear in mind when setting the SubCiHint variable:

- The CMA SubCiHint should have sufficient hints to find the corresponding CI.
- It is necessary to differentiate between CIs that have a composition relationship to a host, and those that do not have such a relation.

RelatedCiHint Values

In general, the `RelatedCiHint` variable should have the following value:

- **For “hosted on” CIs**

Note: When the `RelatedCiHint` contains `@@hostname`, then the CI resolver will only consider CIs that are hosted on that node. This restriction of the search space can yield considerably faster search results.

`<key-attribute-1>:<key-attribute-2>:<key-attribute-n>@@hostname`

Typically, a “hosted on” CI is a sub-type of “software element”. For example, a CI of type `websphereas` has a `container-link` relation to the host.

Another example is the exchange server role CI type `exchangeclientaccessserver`. The root-container for this CI type is a software element, and for that CI type the root-container is `host`.

- **For virtual CIs**

`<key-attribute-1>:<key-attribute-2>:<key-attribute-n>`

A virtual CI does not have a strong containment relation (`container-link` or `root-container`) to a host.

An example of a typical virtual CI type is `cluster`. This CI type does not have a strong containment relation to a host.

Custom Message Attributes in OM

The following table lists custom message attributes that are stored in event properties directly, instead of custom attributes.

CMAs in OM

CMA	Description
Description	Text used to describe the cause of the event.
NoDuplicateSuppression	Event is not suppressed when it is a duplicate of an existing event.
Solution	Text field used to document solutions to help operators solve the problem indicated by the event.
SubCategory	Name of the logical subgroup (category) to which the event belongs (for example, Oracle (database), Accounts (security), or Routers (network)).
EtiHint	Event information used to identify the ETI for each CI.
NodeHint	Event information used to identify node CI related to the event.
RelatedCiHint	Event information used to identify the CI related to the event.
SourceCiHint	Event information used to identify the source CI related to the event.
SourcedFromExternalId	Event sourced from external system with specified ID.
SourcedFromExternalUrl	Event sourced from web application with specified URL.
SourcedFromDnsName	Event sourced from external system with specified DNS name.

Creating Event Type and Health Indicators

After mapping the monitored OM events to the correct CIs in the RTSM, the next step is to create event type indicators and health indicators.

Analyze Events and Define Indicators

Incoming events for the various CI types need to be analyzed, so that meaningful event type indicators (ETIs) and health indicators (HIs) can be created.

ETIs are attributes of events (they do not exist as instances in their own right). ETIs are used to categorize incoming events according to the type of occurrence in the managed IT environment. For example, you can configure messages in OM to include the custom message attribute (CMA) `EtiHint`, which is used to set event type attributes. If the CMA is not configured, ETIs can be set using applicable mapping rules. At least one value is required for an ETI, which is used to describe the event occurrence in the environment. An example would be `Lost database Connection:Occurred`.

Any occurrence on the monitored system of a given type causing an event must be assigned the same ETI. After defining appropriate correlation rules, events are correlated based on the ETIs. The correlation rules relate types of events that can occur on the CI.

HIs determine and display the health of specified aspects of a monitored CI. An HI is used to indicate if a hardware resource is available, and uses one value to represent the normal state of the CI. An example would be `ACME System Status:Available`. One or more values are used to indicate abnormal states for the CI, such as `ACME System Status:Unavailable`.

HIs can also indicate the state of a software application, for example, when the load on certain process is normal, high or exceeded. An example of an abnormal state would be `Job Queue Length:Too Long` for the `ACME Work Process` CI type.

Only events that provide CI state information can set a health indicator. Health indicators are assigned to a specific configuration item type through the associated ETI.

HIs also provide the data needed by a key performance indicator (KPI) to calculate the availability and performance of monitored resources. See ["Assigning HIs to KPIs" on page 43](#), which shows how to assign HIs from the ACME example to health-based KPIs.

The new CI types for our ACME example environment were introduced in the section ["Integrating New Applications" on page 19](#), and illustrated in ["" on page 20](#).

For these CI types, there are specific ETIs and HIs. ["Overview of ETIs and HIs" below](#) shows the result of an investigation into which ETIs/HIs are important within an ACME environment.

Overview of ETIs and HIs

CI Type Display Name	Category	Name	Value	Severity	Policy
ACME System	HI	ACME System Status	Available	Normal	ACME_SystemStatus001
ACME System	HI	ACME System Status	Unavailable	Critical	ACME_SystemStatus001
ACME Application Server	HI	ACME Application Server Status	Available	Normal	ACME_AppServerStatus_001
ACME Application Server	HI	ACME Application Server Status	Unavailable	Major	ACME_AppServerStatus_001
ACME Work Process	ETI	Job Aborted	Occurred		ACME_opcmsg_001
ACME Work Process	ETI	Job Start Passed	Occurred		ACME_opcmsg_002
ACME Work Process	HI	Job Queue Length	Normal	Normal	ACME_JobQueue001

Overview of ETIs and HIs, continued

CI Type Display Name	Category	Name	Value	Severity	Policy
ACME Work Process	HI	Job Queue Length	Too Long	Major	ACME_JobQueue001
ACME Work Process	ETI	Lost Database Connection	Occurred		ACME_LogFile001

Note: Health indicators should show the current state of the health of the monitored object. Therefore, events should only set HIs if there is continuous monitoring of the health of the monitored object.

You create HIs and ETIs in the following area of the OMi user interface:

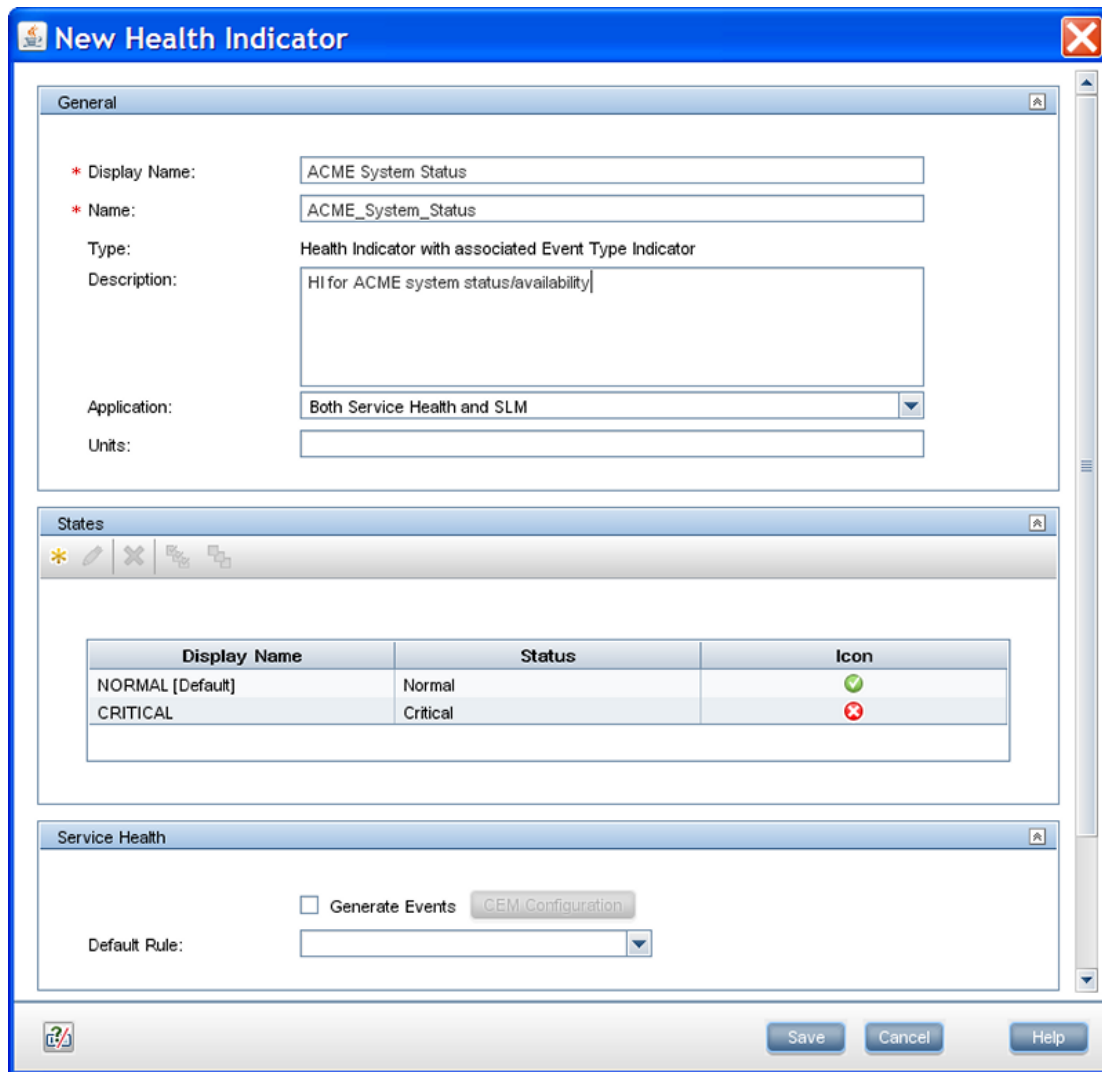
Administration > Service Health > Health- and Event Type Indicators

For full details about how to create HIs and ETIs, see OMi Help.

Here is an example of how you would create an HI for the CI type ACME System:

1. Navigate to the following location:
Administration > Service Health > Health- and Event Type Indicators
2. In the CI Types pane, right-click on the CI type you want to set an indicator for, in this example, ACME System.
3. Click the New (🌸) button, and select the kind of indicator you want to create: either Health Indicator or Event Type Indicator. In this case, click **Health Indicator**. The New Health Indicator dialog opens.
4. In the General area of the New Health Indicator dialog, enter the following information:
In the Display Name field, enter the display name of the HI to be created: **ACME System Status**.
By default, the Name field is filled automatically. For example, if you enter **ACME System Status** as the Display Name for the target HPE Service Manager server, **ACME_Service_Status** is automatically inserted in the Name field. Of course, you can specify your own name in the Name field, if you want to change it from the one suggested automatically.
Optional. In the Description field, enter a description of the CI type you are creating.

In the Application field, select Service Health.

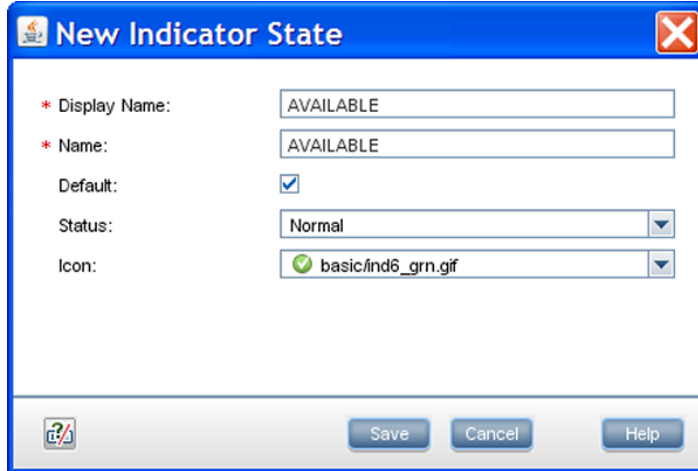


5. In the States area of the New Health Indicator dialog, you can add indicator states by clicking the New (✳) button, or edit an existing state.

In this example, you are adding new indicator states. You will make one state the default state with Normal severity with the value AVAILABLE, and another state with the severity Critical with the value UNAVAILABLE.

To add the default indicator state with the value `AVAILABLE` and with normal severity, do the following:

- a. Select the New (✱) button. The Edit Indicator State dialog opens.



- b. In the **Display Name** field, enter the display name for the HI indicator state: `AVAILABLE`.
 - c. In the **Name** field, enter the system name of the HI indicator state: `AVAILABLE`.
 - d. Check the **Default** checkbox to make this the default (normal case) state.
 - e. In the **Status** field, choose `Normal`.
 - f. In the **Icon** field, choose an icon for this normal state.
 - g. Click **Save**.
6. To add the indicator state with the value `UNAVAILABLE` and with critical severity, do the following:
 - a. Click the New (✱) button.
 - b. In the **Display Name** field, enter the display name for the HI indicator state: `UNAVAILABLE`.
 - c. In the **Name** field, enter the system name of the HI indicator state: `UNAVAILABLE`.
 - d. Clear the **Default** check box.
 - e. In the **Status** field, choose **Critical**.
 - f. In the **Icon** field, choose the icon for this critical state.
 - g. Click **Save**.
 7. Repeat this procedure for other HIs and ETIs you want to create for your CI types.

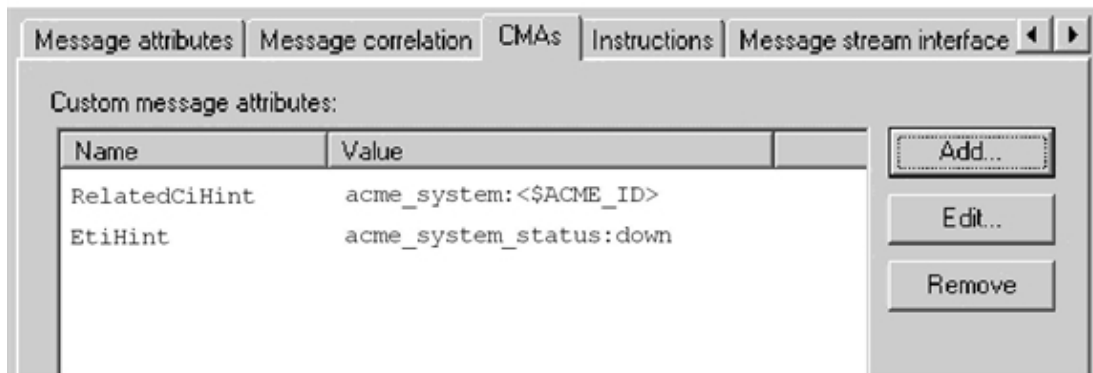
Assigning Event Type Indicators to an Event

There are two ways to assign ETIs to an event. Which option you want to use highly depends on the possibility you have to modify the incoming OM message. We recommend that you use CMAs to set ETIs for incoming OM messages/OMi events. However, it is also possible to assign an ETI to an event with Indicator Mapping Rules.

- Set the custom message attribute `EtIHint` within an OM policy

"Overview of ETIs and HIs" on page 36 shows a list of the analyzed ETIs and HIs for the ACME monitoring system. The last column gives the information about which OM policy creates the event. These OM policy conditions must be enriched with the CMA EtiHint.

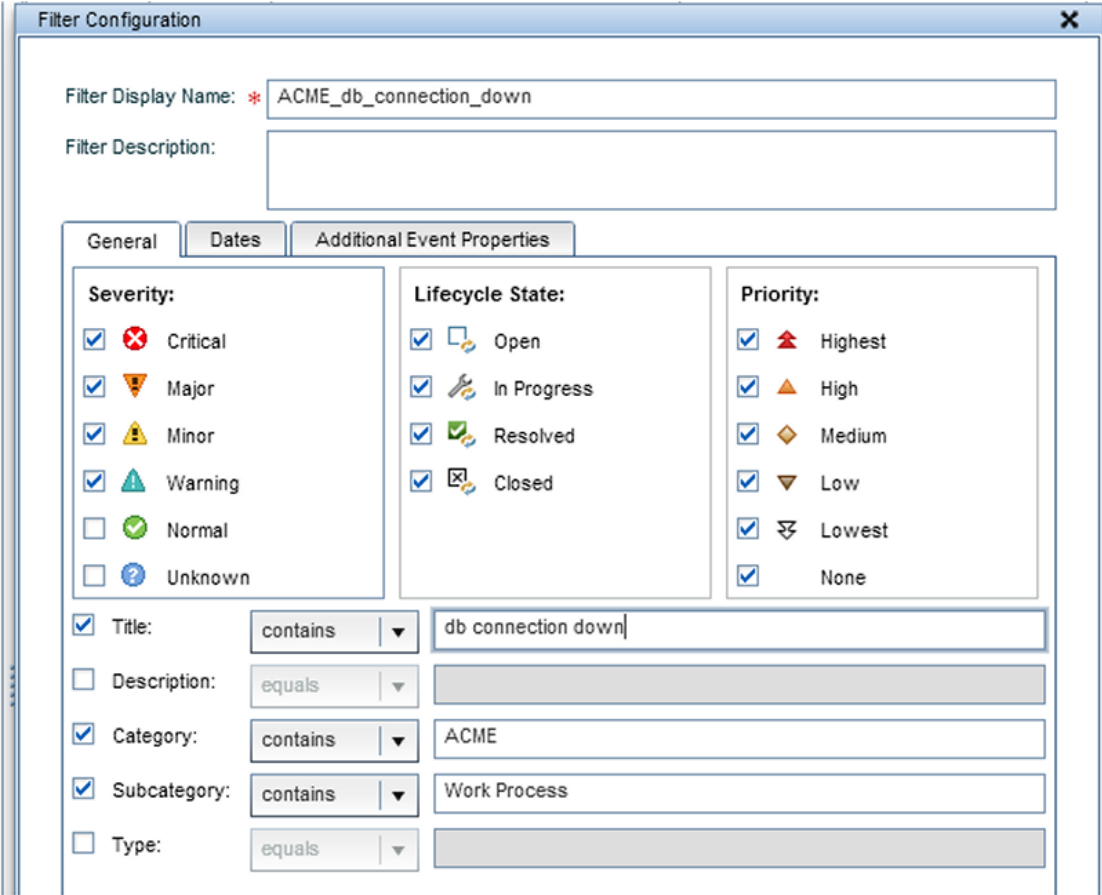
The following graphic shows an example configuration of the CMA EtiHint.



- Use Indicator Mapping Rules

In the ACME example, all messages are sent using OM policies. Therefore, the CMAs are used to set the ETIs. However, it is also possible to use mapping rules to set ETIs for incoming OM messages.

The following graphic shows an example of a filter configuration for db connection down related messages.



The following graphic illustrates the configuration of the mapping rule itself for the Lost Database Connection ETI, using the filter configuration shown above.

The screenshot shows a dialog box titled "ACME WP lost db connection - Create New Mapping Rule". It has three main sections:

- General:**
 - Display Name: * ACME WP lost db connection
 - Name: * ACME_WP_lost_db_connection
 - Description: (empty text box)
 - Active:
- Filter Events:**
 - Event Filter: * ACME_db_connection_down
- Map Event to Indicator:**
 - Indicator: * Lost Database Connection
 - Map to Indicator Value: Based on severity, Specific Indicator Value: * OCCURRED

At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

With this mapping rule, you match every incoming message where Category=ACME, Sub Category=Work Process and Title=db connection lost to the ETI Lost Database Connection for CI type ACME Work Process.

Assigning HIs to KPIs

The next step is to assign HIs to health-based KPIs. HIs provide the data that KPIs need to calculate the availability and performance of monitored resources represented by CIs. You assign HIs to KPIs in the following area of OMi:

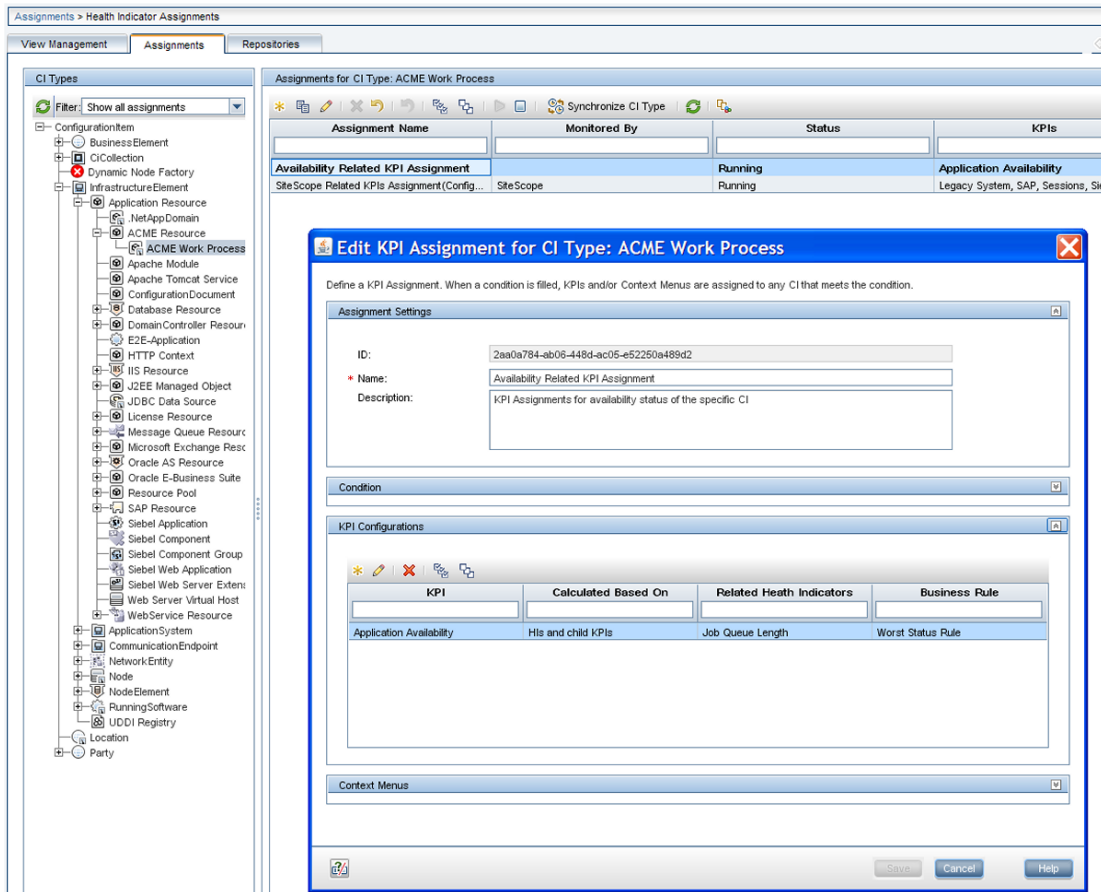
Administration > Service Health > CI Status Calculation > KPI Assignments

For details about KPI assignments, see OMi Help.

The HIs from the ACME example, shown in "[Overview of ETIs and HIs](#)" on page 36, mostly represent the availability status of the specific CI. Therefore, those HIs are assigned to the Operations Availability KPI.

The following graphic illustrates the assignment of the HI Job Queue Length to the Operations Performance KPI. As a result of this configuration, the HI Job Queue Length is also shown in the Health Perspective for events which are related to CIs of type ACME Work Process.

In the Condition area of the Add/Edit KPI Assignment for CI Type dialog, you can limit an assignment to a subset of CIs, for example, those monitored by a certain application, or with a specific set of CI attributes. For full details, see OMi Help.



When you have finished with assigning HIs to KPIs, perform a CI type synchronization for the assignment to take effect on existing CIs.

Tracking Status for CIs not modeled in the RTSM

In cases where a high level of detail is not required, you may have CIs or CI subtypes that are not modeled in the RTSM. This lowers the work associated with discovering and maintaining those CIs, and simplifies your overall model. However, Service Health will not track individual statuses of unmodeled CIs. Instead, a generic ETI for the node is expressed. You can use the custom message attribute SubCiHint to track statuses of unmodeled CI types. When SubCiHint is set, status is tracked separately for each unmodeled CI individually, enabling Service Health to calculate the overall status as the most critical of all the subCis.

For example, you have CPUs with the RelatedCiHint: `cpu:<$acme_cpu_id>:<$acme_nodename>@@<$acme_nodename>` and SubCiHint: 0. If the CPUs are modeled in the RTSM, the RelatedCiHint will resolve to the CPU CI. Each CPU can have its own status and all this information can be aggregated on the node using calculation rules in Service Health.

If the CPUs are not modeled in the RTSM, the system will not take into account the distinct statuses of each CPU. In this case, if you get the notification `CPU0:bad`, followed by `CPU1:bad` and `CPU0:good`, Service Health calculates the overall status as "good", even though only one of the CPUs became "good" again. The status will flip between good and bad, only taking into account the most recent status update.

In this example, when SubCiHint is set, status is tracked for each CPU. When CPU0 converts to "good" but CPU1 remains "bad", the ETI on the node remains "bad".

For more information on setting the SubCiHint, see ["Mapping Events to CIs" on page 31](#).

Custom Message Attributes in OM Messages

The following table lists the supported custom message attributes that can be specified in OM messages and affect the value of the corresponding event property.

CMAs in OM

CMA	Description
Description	Text used to describe the cause of the event.
NoDuplicateSuppression	Event is not suppressed when it is a duplicate of an existing event.
Solution	Text field used to document solutions to help operators solve the problem indicated by the event.
SubCategory	Name of the logical subgroup (category) to which the event belongs (for example, Oracle (database), Accounts (security), or Routers (network)).
EtiHint	Event information used to identify the ETI for each CI.
NodeHint	Event information used to identify node CI related to the event.
RelatedCiHint	Event information used to identify the CI related to the event.

CMAs in OM, continued

CMA	Description
SourceCiHint	Event information used to identify the source CI related to the event.
SourcedFromExternalId	Event sourced from external system with specified ID.
SourcedFromExternalUrl	Event sourced from web application with specified URL.
SourcedFromDnsName	Event sourced from external system with specified DNS name.

Chapter 4: Correlation Rules and Mapping

You can define correlation rules that correlate related events occurring in the same or different domains of the managed IT environment. Topology-Based Event Correlation (TBEC) is used to automatically identify and display the real cause of problems. Events that are only symptoms of the cause event can be filtered out using the Top Level Items filter, resulting in a clearer overview of the actual problems that need to be solved. Correlating events reduces the number of events displayed in the Event Browser and helps operators to locate the cause of the problems more quickly and efficiently.

You should think about which events of the ACME landscape are symptoms or causes of other events that you receive and check whether correlation rules can be defined.

For example, as illustrated in the following graphic, the ACME landscape has a dependency to an Oracle database. We assume here that the monitoring of such a database is realized through the Operations Smart Plug-In for Oracle (SPI for Oracle) and that an event is received when the database is no longer available. At the same time, one of the ACME policies detects a lost database connection as well, so it makes sense to define a correlation rule for these two related events. The following graphic shows such a rule, which defines that the event with the ETI Lost Database Connection Occurred is the symptom event, and the event with the ETI Database Server Status Down is the cause event.

The screenshot displays the 'ACME Oracle DB Availability Correlation Rule - Finish Creating Correlation Rule' window. It features a 'Rule Topology' section with a hierarchical diagram showing the following structure:

- ACME System (Root)
 - Oracle (Connected via 'Dependency')
 - ACME Application Server (Connected via 'Membership')
 - ACME Work Process (Connected to ACME Application Server via 'Composition')

Below the diagram, the 'Symptoms and Causes' table is visible:

Type	CI Type	Indicator	Indicator Value
Cause	Oracle	Database Server Status	Down
Symptom	ACME Work Process	Lost Database Connection	OCCURRED

Another example where defining a correlation rule would be appropriate is a scenario where a long job queue event is the cause of many Job Start Passed messages.

Of course, you can define more elaborate TBEC rules for multiple symptoms and multiple causes. For full details about correlation rules and mapping, see OMi Help.

Chapter 5: Additional Event Processing

You can perform additional event processing to modify and enrich events. For example, you can enrich the Title and Description fields of an event with either or both node and CI labels by using the variable placeholders: `${node.label}` and `${relatedCi.label}`.

The Event Processing Interface (EPI) lets you run any user-defined Groovy scripts during event processing. You may want to enrich events, for example, from an external SQL database. For more information about developing and deploying Groovy scripts, see ["Groovy Scripts" on page 335](#).

For an overview of the event processing pipeline, the Event Processing Interface, and EPI scripts, see ["Event Processing Interface" on page 123](#).

You can also configure custom actions to apply to events. You can configure Groovy scripts to make custom actions available in the Event Browser.

For an overview of custom actions and scripts, see ["Scripts For Custom Actions" on page 132](#)

For details about how to create EPI and custom action scripts, including some sample Groovy scripts provided with the product, see ["Creating EPI and Custom Action Scripts" on page 134](#).

Chapter 6: Tools

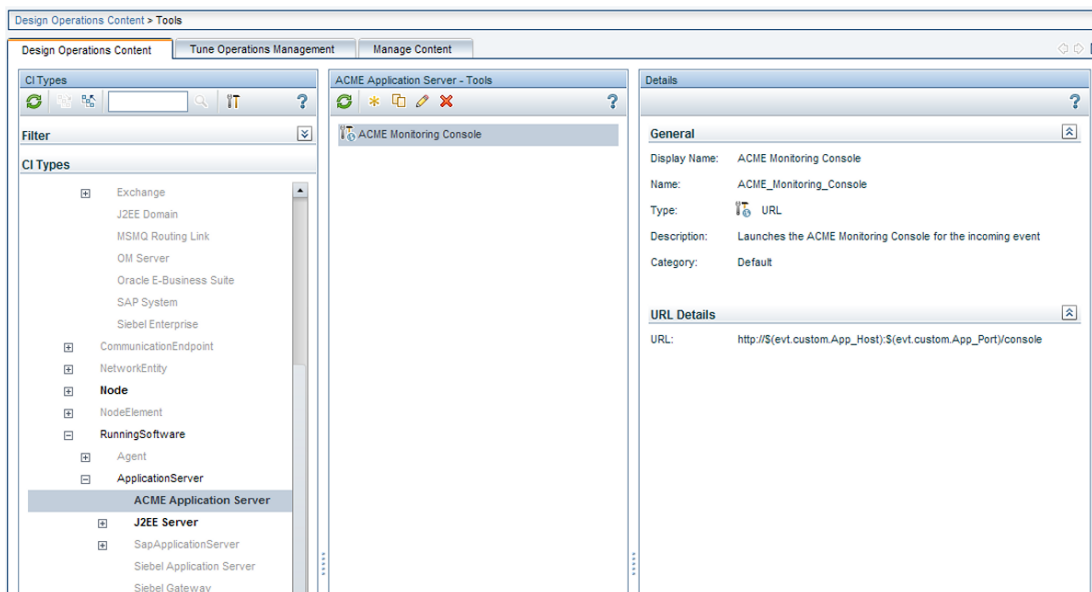
You can configure tools to help domain operators manage and monitor specific events and CIs, and to solve problems that occur in your new application area.

By assigning tools to a particular CI type, you can make sure that the assigned tools are always available in the context of any event that has an impact on any instance of the selected CI type.

In the ACME example, we provide a cross-URL launch from the Event Browser to the ACME administrator console.

For more details about tools, see OMi Help.

The following graphic shows an example of a URL launch to the ACME monitoring console. The tool uses the custom event attributes `App_Host` and `App_Port` to generate the appropriate URL.

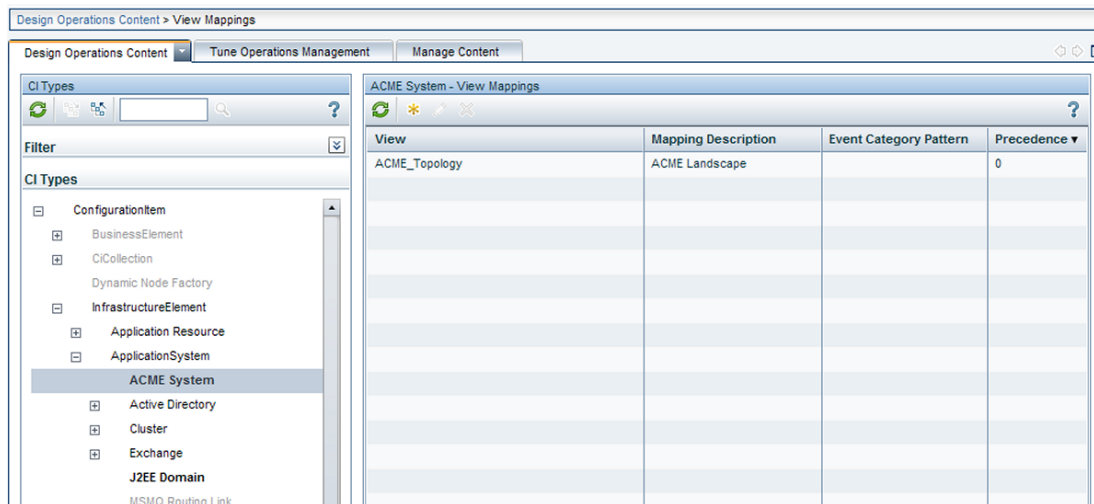


Chapter 7: View Mappings

To ensure that views are available for selection and use in the Health Top View pane, the configuration item impacted by the event selected in the event browser must be explicitly mapped to at least one existing view. The RTSM Modeling Studio enables users to create new views in the RTSM. You can associate CI types to more than one view. Multiple views are listed by precedence, and the view with the highest precedent is shown as the default view.

With the View Mappings manager you can map CI types to RTSM views. Only those views that are mapped to the CI type related to the selected event are available for display in the Selected Views drop-down list in the Health Top View pane.

The following graphic shows the View Mappings pane. The CI type ACME System is mapped to (has an association with) the view ACME_Topology.



Chapter 8: Performance Dashboards

You can associate performance dashboards with a specific CI type so that performance dashboards are always available in the context of any event that has an impact on the selected CI type.

To do this for your new ACME application, you need to:

- Create a new performance dashboard or edit an existing performance dashboard, using the OMi Performance Dashboard. For details, see [Performance Perspective](#).
- Map suitable performance dashboards to the ACME CI types.

Integrating Performance Data

The process of integrating performance data involves:

- Gathering metrics
- Storing the metrics
- Using the metrics to make graphs

For example, performance data for OM can be collected by the embedded performance component (EPC) of the HPE Operations Agent or by the HPE Performance Agent.

Data collected by these agents can be used in OM measurement threshold policies to create alarm messages, and historical data can be displayed using HPE Performance Manager or the OMi Performance Dashboard of OMi, helping operators to analyze and fix problems.

An easy way to integrate performance data is to use the OM measurement threshold policy type.

The OM measurement threshold policy enables you to collect performance data from several sources:

- External / Program – data sent from an external program (using the `opcmon` command line interface or API)
- MIB – collect metrics from a SNMP Management Information Base (MIB)
- Real Time Performance Measurement – collect metrics from Windows Performance Counters
- WMI – collect metrics from Windows Management Instrumentation

The metrics collected from these sources can be stored easily in the Embedded Performance Component using the `Store in Embedded Performance Component` option.

Chapter 9: Packaging Content

This section describes how to transfer the content that you create on one OMi instance to another. If you want to use the content you create on another OMi instance, you need to:

- Create and export the RTSM package.
- Create and export the ACME content pack.
- Upload the files to the other OMi instance.

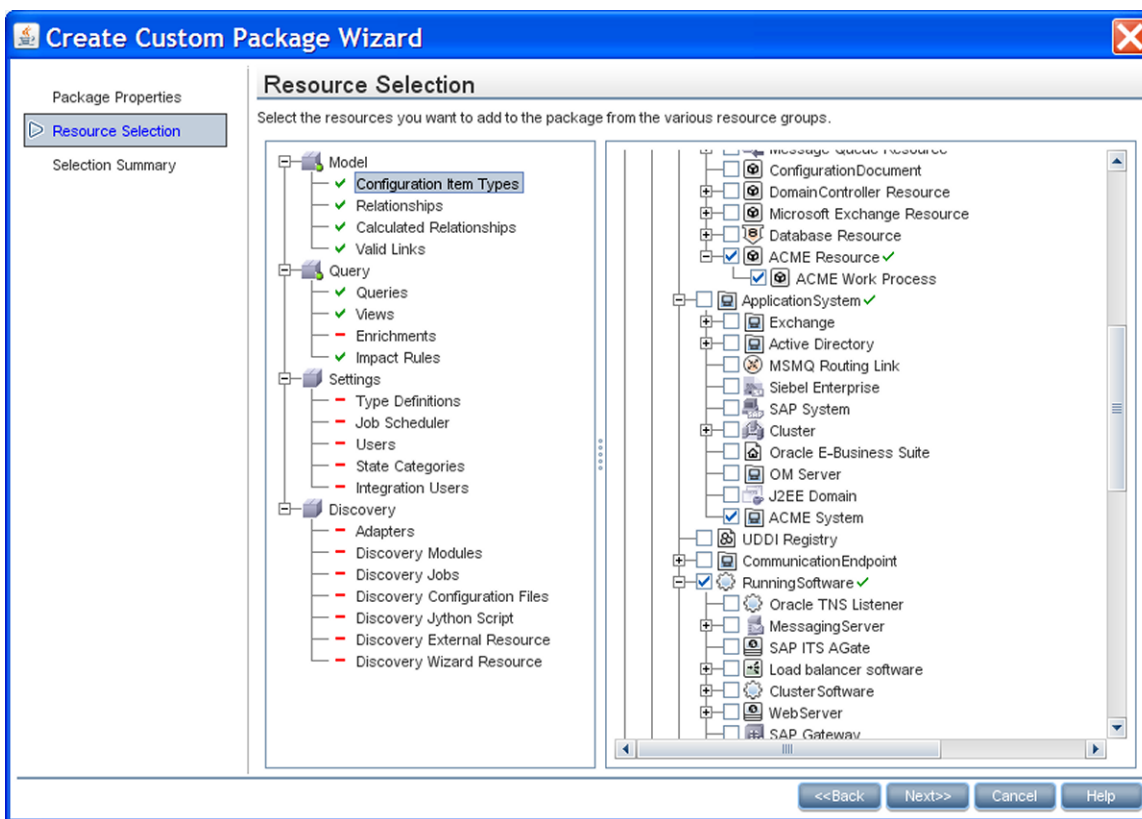
Create RTSM Packages

RTSM packages are essential parts of content. You can use RTSM packages to manage view models and configuration item types. For example, you can use packages for exporting, importing, and updating content, either on the same server or between different instances of OMi.

The workflow for creating an ACME RTSM package is summarized below:

1. Create the ACME RTSM package.
For more information about how to create a content pack, see OMi Help.
2. Add the CI types (see ["Topology" on page 19](#)), and views (see ["View Mappings" on page 50](#)) that you created in previous steps.

The following graphic shows the selected items which are part of the ACME RTSM package. Usually CI types, TQLs and views are part of such a package.



Create a Content Pack

A content pack can contain a complete snapshot of all (or any part of) the rules, tools, graphs, mappings, and assignments that you define and configure to help users manage your IT environment.

You can share content between content packs. In your content pack definition, you can include content directly in your content pack. Content you include in your content pack definition may depend on

referenced content. Referenced content is content not explicitly included in your content pack, but referenced in other content pack definitions. For example, when you include a correlation rule as content in your content pack definition, it may require indicators that are referenced to another content pack definition.

How referenced content is handled is guided by the principle that referenced content should be added to a content pack definition only if it is not included in other content pack definitions. Therefore, a content pack definition is very likely to have dependencies to other content pack definitions. Dependencies to other content packs mean that you have to consider what impact the dependencies have. For example, when importing a content packs on another system, the order of loading content packs could be important, and, of course, the referenced content must be present on the target system.

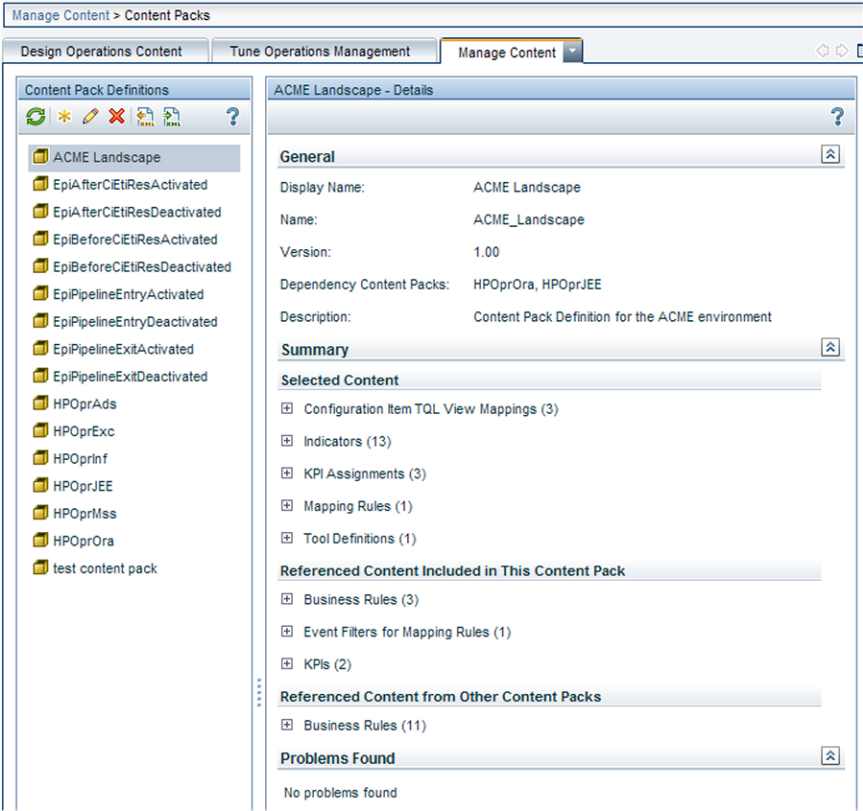
The handling of referenced content is summarized in the following use cases:

- Case 1: Referenced content is not included in other content pack definitions. In this case, the content is added to the content pack you are creating.
- Case 2: Referenced content is included in another content pack definition. In this case, rather than adding the content to your current content pack, a dependency to the referenced content is marked to the other content pack definition.
- Case 3: Referenced content is included in more than one content pack definition. This is similar to Case 2, and the dependency to the referenced content is marked to one of the content pack definitions where the referenced content is included.

The workflow for creating an ACME content pack is summarized below:

- Create the ACME content pack.
For more information about how to create a content pack, see OMi Help.
- Add all correlation rules, HIs, ETIs, KPI assignments, tools, graphs and graph assignments, and view mappings that you created in previous steps. EPI scripts, custom actions, policy templates, aspects, management templates, instrumentation and parameters can also be defined in content packs.
- *Optional.* Include content from any dependent content pack definitions.
- Export the content pack ZIP file (ACME.zip, for the ACME environment content pack) using the content manager. You can exchange content between instances of OMi by defining and creating packages using content management tools. The package you create can be exported to a file which you can then use to deploy the same content on another instance of OMi.

The following graphic shows a selection in the content manager within the ACME environment content pack.



Upload the Content

Copy the exported files to another OMi instance, and upload them to the other system in the following sequence:

- Upload the RTSM packages.
- Upload the content packs.

Upload RTSM Packages

To upload an RTSM package:

1. Place the zip-file in a suitable directory of your choice in the file system.
2. Use the RTSM package manager to upload the RTSM package.

Upload Content Packs

To upload a content pack:

1. Place the exported XML file in a suitable directory of your choice on the file system.
2. Upload the content pack using the content manager.

Part III: Populating the Run-time Service Model

This section provides information for developers to create their own topology synchronization mapping rules to populate the Runtime Service Model (RTSM (Run-time Service Model)) with configuration items (CIs) and CI relationships from nodes and services in OM.

This section reuses the ACME environment example, introduced in Section I: Content Development, which illustrates how to create topology synchronization rules specific to a particular service model.

This section is structured as follows:

- ["Topology Synchronization" on page 58](#)
- ["Synchronization Packages" on page 67](#)
- ["Scripting" on page 79](#)
- ["Testing and Troubleshooting" on page 84](#)
- ["Mapping Engine and Syntax" on page 96](#)

Chapter 10: Topology Synchronization

You can use topology synchronization instead of, or in conjunction with, HPE DFM (Data Flow Management) discovery or service auto-discovery policy templates to create CIs in the RTSM (Runtime Service Model). Accurate and up-to-date CI topology data in the RTSM is essential for Topology-based Event Correlation (TBEC), context-specific tools, and OMi-wide service health monitoring (Health Perspective).

Note: Alternatively, you can import existing topology information into the RTSM. For details on importing data from Excel Workbooks and other external sources, see the *HPE Universal CMDB Discovery and Integration Content Guide*.

Topology synchronization can be defined as a set of rules that determine how services, nodes, node groups, and node layout groups monitored by OM are mapped to OMi configuration items (CIs) in the operational database (RTSM).

Topology synchronization is a standard part of the OMi license. It offers a solution to populate the RTSM with CIs from service models (or alternatively, service trees or service graphs) defined in OM. Topology synchronization is especially interesting for OM customers who have created their own service model.

Mapping rules used for topology synchronization are contained in topology synchronization packages. The installation copies the topology synchronization rules to the local file system. The rules are then uploaded to the database when synchronization is used for the first time. You can also write your own topology synchronization rules to populate the RTSM based on your service model or the discovered topology data.

To automate the discovery process, you can use service auto-discovery policies. These enable you to run scripts or programs that discover CIs in your managed environment and automatically populate RTSM. For more information, see [Configuring Service Auto-Discovery Policies](#).

Topology Synchronization Overview

Topology synchronization makes it possible to share topology data from OM between any supported combination of multiple OM and OMi instances, in a distributed, hierarchical environment.

An OMi instance can be configured to receive topology data from multiple OM and other OMi instances. An OMi instance can also be configured to forward topology data to other OMi instances.

Topology data is forwarded to the OMi instance dynamically, that is, as soon as a topology change is discovered by an HPE Operations Agent (and written to the OM service model). Topology synchronization enables near-real-time discovery of infrastructure changes. For example, if a node is added to the environment, this change is immediately forwarded, and the database updated.

For details of supported OM versions, see Support Matrices for Operations Center products.

When you first configure topology synchronization, the source system (HPE Operations Agent or OM) sends all its topology data to one or more target systems (OM or OMi). After this first synchronization, topology synchronization forwards only the discovered topology changes. Changes in topology of an

OMi instance are *not* forwarded, and *not* synchronized back to an OM system using topology synchronization.

For details about database synchronization, refer to the RTSM documentation.

Once configured, topology synchronization runs continuously in the background. Also running continuously in the background is a process that prevents aging in the RTSM by touching all elements from the previous synchronization.

For more details about RTSM aging, see the *Modeling Guide*.

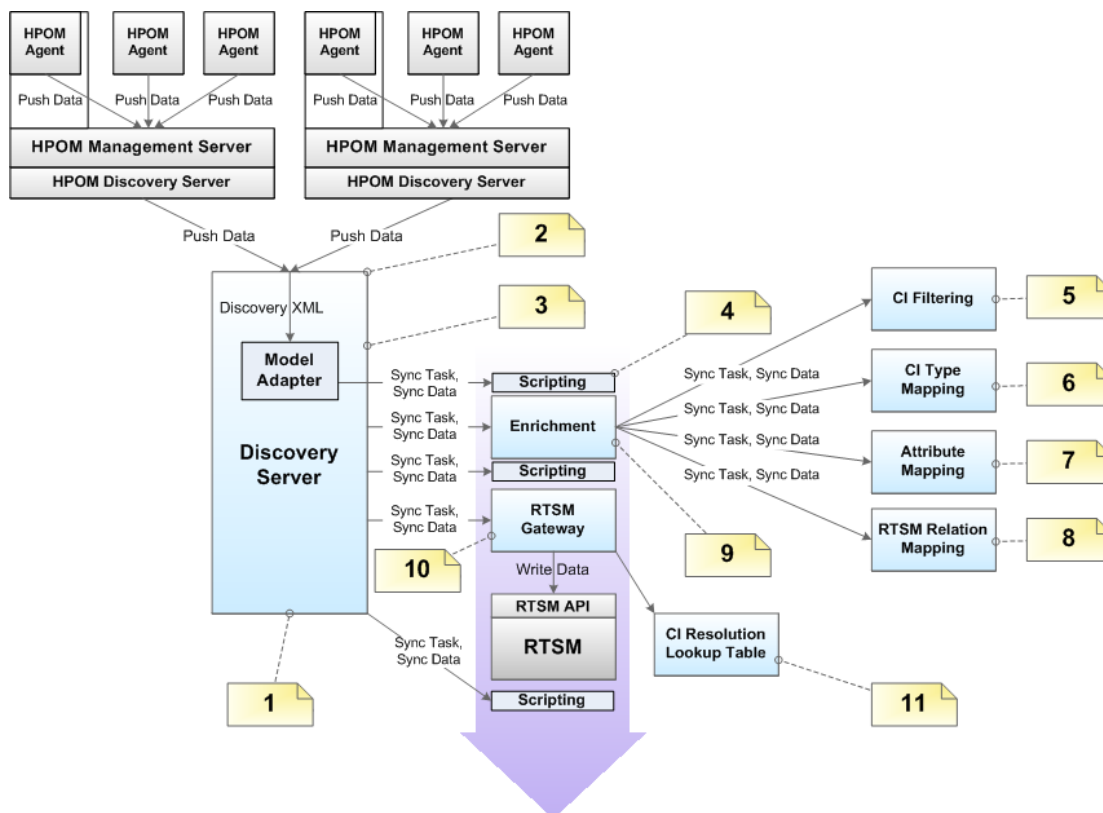
To create CIs in the RTSM, based on mapping rules and using OM topology data as sources, topology synchronization does the following:

- Forwards the discovered topology data from configured source forwarding servers to configured target servers, using HTTPS-based communication requiring trusted certificate exchange.
- Receives discovered topology data asynchronously.
- Converts the topology data into a format compatible with OMi.
- Uploads the topology data to the RTSM.
- Updates the data in the RTSM on demand.
- Performs delta detection and deletes elements from the RTSM that are deleted in OM.
- Provides a mapping table for CI resolution.

Note: The definition of particular CI types is especially important when synchronizing the topologies of OMi and OM because the topology synchronization process cannot create new CI types. If the topology synchronization process attempts to map to a CI type that does not exist in the RTSM, the topology synchronization process aborts.

Topology Synchronization Architecture

The following graphic provides an overview of the topology synchronization architecture.



The reference numbers in the diagram refer to the following notes:

Reference	Refers to	Note
1	Discovery Server	<ul style="list-style-type: none"> Controls the synchronization process and passes data from one component to the other. Runs in hpsbm_wde process on the gateway servers.
2	Receiving topology data on the Discovery Server	Receive topology data in the form of a discovery XML forwarded from an OM server or another OMi server.
3	Model Adapter	Convert the discovery XML to the synchronization data structure.
4	Scripting	Execute Groovy scripts to manipulate synchronization data.
5	CI Filtering	Use the mapping engine to specify which CIs are part of the synchronized model, and which are not.

Reference	Refers to	Note
6	CI Type Mapping	Map Service Type Definitions (STDs) for RTSM (Run-time Service Model) types.
7	Attribute Mapping	Map OM relations that are not typed to RTSM relations.
8	RTSM Relation Mapping	<ul style="list-style-type: none"> • Create the relations for the RTSM. • Map the root container to the CI to be able to create the CI in the RTSM according to the model.
9	Enrichment	Control the whole enrichment process and return the modified synchronization data.
10	RTSM Gateway	Write the synchronization data to the RTSM.
11	CI Resolution Mapping Table	For each synchronized CI, update a mapping table for the CI Resolution component that maps the OM Service ID to the RTSM CI ID.

Synchronization Packages and Mapping

Topology synchronization uses synchronization packages to create CIs in the RTSM. Topology synchronization packages contain the mapping between one or more services, nodes, or node groups on the OM side to one or more CIs on the RTSM side.

A topology synchronization package consists of XML configuration files (see ["Mapping Files" on page 71](#)). These files are responsible for transforming OM services, nodes and node groups into CIs in the RTSM, and synchronizing those CIs with data from specified services, nodes, node groups, and node layout groups in OM.

There are two types of topology synchronization packages:

- Standard packages provided out-of-the-box as part of the installation package.
For more information, see ["Standard Out-of-the-box Synchronization Packages" on page 69](#).
- Additional, out-of-the-box packages that are aligned with a subset of OM SPIs and available content packs.
For more information, see ["Additional Out-of-the-box Synchronization Packages" on page 69](#).

You can also create your own topology synchronization packages. An example of how to configure topology synchronization and create your own synchronization package is provided in the section ["Configuring Topology Synchronization: ACME Example" on page 72](#).

For an out-of-the-box installation, synchronization packages are automatically loaded from the file system to the RTSM. When you make new synchronization packages, or changes to existing packages, you need to run a command-line tool to upload the new or changed packages to the RTSM. For details, see ["Managing Synchronization Packages" on page 65](#).

For more information about synchronization packages and mapping, see ["Synchronization Packages" on page 67](#).

Scripting and Topology Data

Scripting enables you to perform additional processing and customizing during the synchronization process before the mapping and before and after the upload of topology data from OM to the RTSM.

Groovy scripts are supported to manipulate the synchronization data during the synchronization process. Groovy scripts can be placed into a topology synchronization package. For more information about topology synchronization scripts, see ["Scripting" on page 79](#). For more information about developing and deploying Groovy scripts, see ["Groovy Scripts" on page 335](#).

CI Resolution Using a Mapping Table

Topology synchronization creates a mapping table for all CIs synchronized from OM. This mapping table can be used as a short-cut for CI resolution. The service ID from the OM service is searched in the table, and mapped to a CI in the RTSM. When use of the mapping table is enabled, the table is analyzed first before CI resolution is used. If the mapping table yields no match, CI resolution then continues with the mapping process, including Smart Message Mapping.

The use of this mapping table is enabled by default (the **Use Topology Sync Shortcut** setting in the CI Resolver settings is set to **true**).

You would typically enable use of the mapping table in situations where there is a direct, one-to-one relationship between a service in the OM service tree and a CI for that service in the RTSM.

There are situations in which you would not use the mapping table shortcut, for example, where the service tree structure and RTSM structure are quite different, and there is no longer a one-to-one relationship between a service and a CI for that service in the RTSM. There may be many CIs in the RTSM that provide information about the cause of a service failure, and CI resolution is the quickest, most reliable way to find the most appropriate CI for the service object.

If you do not want to use the mapping table, you can disable it in the CI Resolver settings in the OMi Infrastructure Settings Manager:

Administration > Setup and Maintenance > Infrastructure Settings

Edit **Operations Management - CI Resolver Settings > Use Topology Sync Shortcut**.

Topology Synchronization File Locations

Initial product installation copies topology synchronization files to the locations specified in this section on the local file system on the OMi data processing server.

You can find the topology synchronization files in the following locations:

Binaries:

Windows: <OMi_HOME>/bin/opr-sdtool.bat

Linux: <OMi_HOME>/bin/opr-sdtool.sh

<OMi_HOME>/opr/lib/opr-ts-*.jar

<OMi_HOME>/opr/lib/OvSvcDiscServer.jar

Log Files:

<OMi_HOME>/log/wde/opr-svcdiscserver.log

<OvDataDir>/log/OvSvcDiscServer.log

Log File Configuration Files:

<OMi_HOME>/conf/core/Tools/log4j/wde/opr-svcdiscserver.properties

Topology Synchronization Packages:

<OMi_HOME>/conf/opr/topology-sync/sync-packages

Schema Files:

<OMi_HOME>/conf/opr/topology-sync/schemas

Topology Synchronization Settings

For the successful synchronization, make sure that the **OM Topology Synchronization Settings** are correctly configured.

Note: These settings are mandatory for the correct configuration and successful synchronization of the object topology in the environments monitored by OMi and OM.

To access

Administration > Setup and Maintenance > Infrastructure Settings

Operations Management - OM Topology Synchronization Settings

Managing Synchronization Packages

The command-line tool `opr-sdtool` uploads and enables new or changed synchronization packages from the file system to the database and downloads synchronization packages from the database to files.

This section includes:

- ["Uploading and enabling synchronization packages" below](#)
- ["Downloading synchronization packages" on the next page](#)
- ["opr-sdtool user permissions" on the next page](#)
- ["Managing Synchronization Packages with the Content Packs Manager" on the next page](#)

Uploading and enabling synchronization packages

Topology synchronization uses the synchronization packages that are loaded in the database. As a consequence, if you make changes to the synchronization package files, you must upload the synchronization packages again to the database.

OMi only processes synchronization packages that are enabled. The optional `-enablepackage` option adds the synchronization package to the **Packages for Topology Sync** infrastructure setting:

Administration > Setup and Maintenance > Infrastructure Settings

Operations Management - HPOM Topology Synchronization Settings > Packages for Topology Sync

To upload and enable new or changed synchronization packages to the database, run the following command:

Windows: `opr-sdtool.bat -uploadpackage <path of synchronization package> [-enablepackage]`

Linux: `opr-sdtool.sh -uploadpackage <path of synchronization package> [-enablepackage]`

Downloading synchronization packages

You can also use the download option to ensure that the latest version of the synchronization packages is available for editing on each OMi server when OMi is installed in a distributed environment.

To download synchronization packages from the database, run the following command:

```
Windows: opr-sdtool.bat -downloadpackage [<syncPackageName>] [-path  
[<downloadPath>]]
```

```
Linux: opr-sdtool.sh -downloadpackage [<syncPackageName>] [-path [<downloadPath>]]
```

If you do not specify the name of the synchronization package, the tool downloads all packages that are currently in the database.

The tool by default downloads the packages to <OMi_HOME>/conf/opr/topology-sync/sync-packages unless you specify a download path.

opr-sdtool user permissions

The user running the opr-sdtool command-line interface must be a local user (Windows) or the user under which the OMi processes are running (Linux). If the SQL Server instance uses Windows Authentication Mode, the user running opr-sdtool must be granted access to the Events database.

Managing Synchronization Packages with the Content Packs Manager

You can also use the Content Packs Manager to import and export synchronization packages. For details, see the *OMi Administration Guide*.

After importing the synchronization packages, you must manually enable the packages by adding them to the **Packages for Topology Sync** infrastructure setting. For details, see "[Uploading and enabling synchronization packages](#)" on the previous page.

Uploading OM SPI Service Type Definitions to the Database

OM Smart Plug-ins (SPI) Service Type Definitions are used to process received services from agents.

The command-line tool opr-sdtool is provided to upload new or changed service type definitions from OM Smart Plug-ins (SPIs) to the database, using the -uploadstd command-line option.

To upload new or changed service type definitions from OM SPIs to the database, run the following command:

```
Windows: opr-sdtool.bat -uploadstd <path of MofFile>
```

```
Linux: opr-sdtool.sh -uploadstd <path of MofFile>
```

The user running the opr-sdtool command-line interface must be a local user (Windows) or the user under which the OMi processes are running (Linux). If the SQL Server instance uses Windows Authentication Mode, the user running opr-sdtool must be granted access to the Events database.

Chapter 11: Synchronization Packages

This section describes the topology synchronization packages that contain the rules for mapping OM topology data to CIs in the RTSM.

The chapter is structured as follows:

- ["Synchronization Packages Overview" on the next page](#)
- ["Standard Out-of-the-box Synchronization Packages" on page 69](#)
- ["Additional Out-of-the-box Synchronization Packages" on page 69](#)
- ["Package Descriptor File: package.xml" on page 70](#)
- ["Mapping Files" on page 71](#)
- ["Configuring Topology Synchronization: ACME Example" on page 72](#)
- ["Customizing Synchronization and Scripting" on page 78](#)
- ["Synchronization Package Locations" on page 78](#)

Synchronization Packages Overview

Topology synchronization packages contain the mapping between one or more service models, nodes, or node groups on the OM side to one or more CIs on the RTSM side.

A topology synchronization package contains a set of XML configuration files that define the mapping rules (context, CI type, attributes, and so on) during topology synchronization. The configuration files are used to:

- Transform topology data (for example, OM services, nodes, node groups, and node layout groups) into CIs in the RTSM.
- Synchronize CIs in the RTSM with topology data from OM.

A topology synchronization package must include the package descriptor file (`package.xml`) to define the synchronization package (see ["Package Descriptor File: package.xml" on page 70](#)).

Mapping files that can be part of a synchronization package are:

- `contextmapping.xml`
- `typemapping.xml`
- `attributemapping.xml`
- `relationmapping.xml`

For more information about the XML configuration files, see ["Mapping Files" on page 71](#).

For basic information on mapping, see ["Mapping Engine and Syntax" on page 96](#).

Groovy scripts can also be placed into a topology synchronization package to manipulate the synchronization data during the synchronization process, or to carry out post-synchronization activities, for example, for auditing purposes. You can include the following Groovy scripts in a topology synchronization package:

- `preEnrichment.groovy`
- `preUpload.groovy`
- `postUpload.groovy`

For more information about Groovy scripts, see ["Groovy Scripts" on page 79](#).

Standard Out-of-the-box Synchronization Packages

You can specify the content you want to update when synchronizing the topology between OMi and OM.

There are three out-of-the-box topology synchronization packages:

- `default`
Contains basic type mappings for nodes, and basic attribute mappings for nodes and node groups.
Does not create any CIs in the RTSM.
Should not be removed from the list of enabled packages.
- `operations-agent`
In addition to creating the host CI itself, creates a CI instance of type `hp_operations_agent` for each OM managed node with an agent, and relates it to the host CI. Also creates CIs of type `omservers` and relates it to its host and to all the `hp_operations_agent` CIs.
- `nodegroups`
In addition to creating the host CI itself, maps OM node groups to the RTSM CI type `ci_group`, creates instances of the CI type `ci_group`, and creates relations for the contained nodes. Also creates CIs of type `ipaddress` and `interface` and relates them to their host.
- `layoutgroups`
In addition to creating the host CI itself, maps OM layout groups to the RTSM CI type `CI Collection (ci_group)` CI type `CI Collection (ci_group)`, creates instances of the CI type `CI Collection (ci_group)`, and creates relations for the contained nodes.

Note: `ci_group` is visually known as **CI Collection**.

In the **Packages for Topology Sync** setting in OM Topology Synchronization settings, you can list the packages whose contents should be updated during the topology synchronization process:

Administration > Setup and Maintenance > Infrastructure Settings

Operations Management - HPOM Topology Synchronization Settings > Packages for Topology Sync

The entries in the list must be separated by a semicolon (;) as illustrated in the following example:

```
default;nodegroups;operations-agent
```

By default, packages are located in the following directory:

```
<OMi_HOME>/conf/opr/topology-sync/sync-packages
```

Additional topology synchronization packages are provided in the content packs.

Additional Out-of-the-box Synchronization Packages

Additional topology synchronization packages are provided out-of-the-box in content packs. Content packs include the following:

- ActiveDirectory
- Exchange
- MS SQL Server
- Oracle
- J2EE (includes WebSphere and WebLogic content)
- Infrastructure (includes UNIX and Windows operating systems, Virtualization Systems, and Cluster Systems)

These additional topology synchronization packages are not enabled by default. To enable them:

1. Load the content pack(s) you wish to use.
2. Enable the synchronization packages manually in the Infrastructure Settings:

Administration > Setup and Maintenance > Infrastructure Settings

Operations Management - HPOM Topology Synchronization Settings > Packages for Topology Sync

Topology synchronization packages are written to the following directory:

```
<OMi_HOME>/conf/opr/topology-sync/sync-packages
```

For example, the Oracle content pack uses the package (and directory) name HP0prOra. This is the name you enter in the list if you want the mapping rules to be executed during topology synchronization. If we add the Oracle package to the list of standard out-of-the-box packages we had in the example in the section "[Standard Out-of-the-box Synchronization Packages](#)" on the previous page, the list would look like this:

```
default;nodegroups;operations-agent;HP0prOra
```

Note: If you are adding custom packages, note that the package name is the same as the name of the directory in which the package is located. Be aware that when a synchronization package is removed, CIs added to the RTSM by previous runs of that synchronization package, and that have not been reconciled with other CIs, are also removed.

Package Descriptor File: package.xml

A topology synchronization package must include the package descriptor file (package.xml). The package.xml file defines a topology synchronization package and includes:

- <Name> The name of the package must be the same as the name of the subdirectory in which you place the synchronization package:

```
<OMi_HOME>/con/opr/topology-sync/sync-packages
```

- <Description> Description of the package.
- <Priority> Priority level of the package.

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

Note: There may be more than one synchronization package with the same priority. The order

of execution of the rules between synchronization packages with the same priority is not specified.

Mapping Files

The following mapping files can be included in a topology synchronization packages.

Context Mapping (Filtering): `contextmapping.xml`

You can determine which elements of an OM service tree you want to include in the topology synchronization for mapping in the RTSM by configuring the filtering file, `contextmapping.xml`. Filtering involves assigning a context to those CIs you want to synchronize. Configuring the context enables you to apply mapping rules selectively to CIs of the same context.

For example, specified OM services are tagged, and all subsequent mapping rules contained in other configuration files are applied to those tagged services. A service that has no context assigned is not included for synchronization.

Type Mapping: `typemapping.xml`

The type mapping file `typemapping.xml` defines the mapping from a service in OM based on their attributes to the type of a CI in the RTSM.

Attribute Mapping: `attributemapping.xml`

The attribute mapping file `attributemapping.xml` defines the mapping between the attributes of a service in OM and the attributes of a CI in the RTSM.

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

Relation Mapping: `relationmapping.xml`

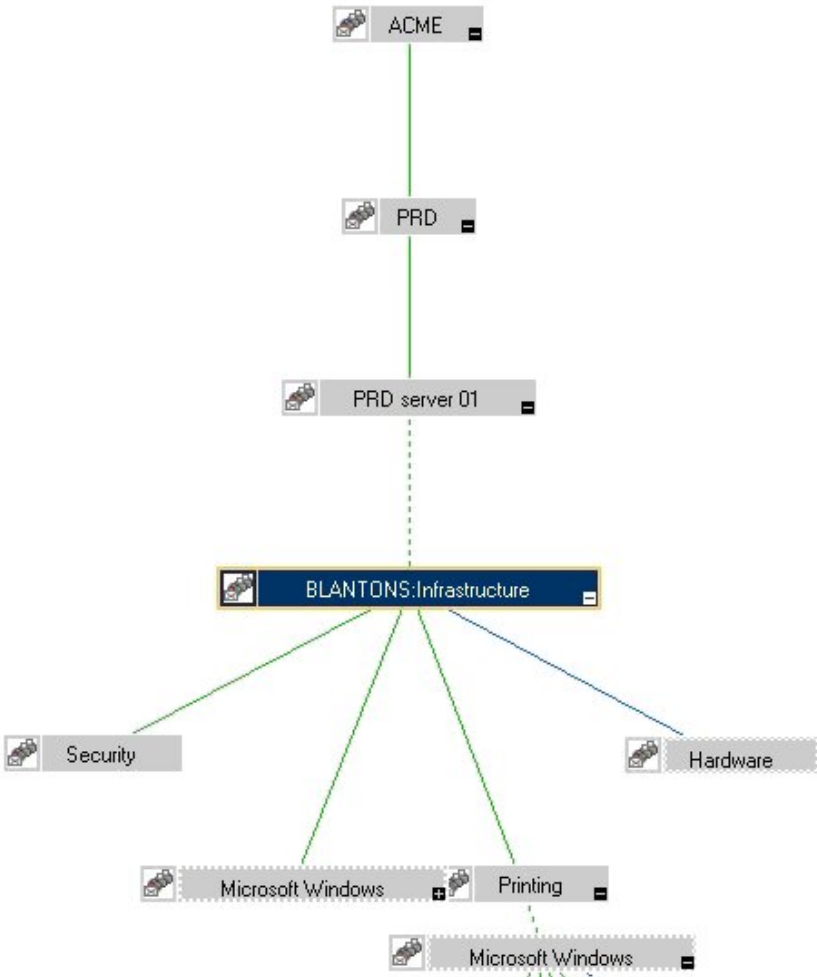
Using the relation mapping file `relationmapping.xml`, you can define the CI relationships created in the RTSM between specified OM services.

Make sure that the specified OM services are created as CIs in the RTSM. Otherwise it is not possible for topology synchronization to create a relationship in the RTSM.

Configuring Topology Synchronization: ACME Example

This section provides a walk-through of how to configure topology synchronization, using the fictitious “ACME” content area as an example.

The following graphic shows a service tree from OM discovery.



Configure Package Descriptor File: package.xml

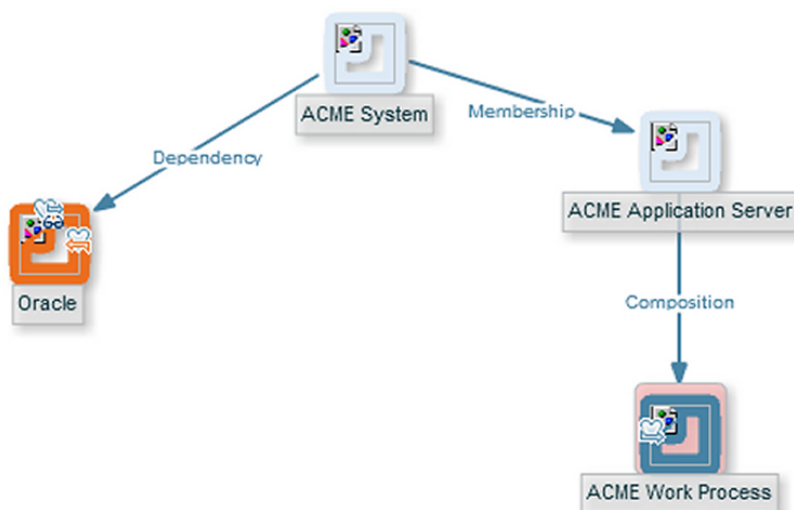
Configure the file `package.xml` to define the name and provide a description of your topology synchronization package, together with a priority level. For the ACME topology synchronization package, the `package.xml` file looks like this:

```
<Package>  
  <Name>ACME</Name>  
  <Description>Service to RTSM Mapping for ACME Landscape.</Description>  
  <Priority>5</Priority>  
</Package>
```

Configure Context Mapping (Filtering) File: contextmapping.xml

Configure the file `contextmapping.xml` to tag which elements included in topology data you want to include in the topology synchronization for mapping in the RTSM. The mapping rules contained in other configuration files are applied to the tagged elements.

The following graphic represents the view for ACME in the RTSM.



An example configuration of contextmapping.xml follows, where a context called ACME_Landscape is assigned to those OM service elements, of type ACME_System and ACME_Application_Server, for which you want to create CIs in the RTSM:

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../schemas/mapping.xsd">
<!-- CONFIGURE THE CIs THAT DEFINE THE CONTEXT FOR THE MAPPING -->
  <Rules>
    <Rule name="Filter ACME Items">
      <Condition>
        <!-- Select all Service Elements of interest
        further refinements will be made later -->
        <Or>
          <Equals>
            <OMType />
            <Value>ACME_System</Value>
          </Equals>
          <Equals>
            <OMType />
            <Value>ACME_Application_Server</Value>
          </Equals>
        </Or>
      </Condition>
      <MapTo>
        <Context>ACME_Landscape</Context>
      </MapTo>
    </Rule>
  </Rules>
</Mapping>
```

Configure Type Mapping File: typemapping.xml

Configure the type mapping file typemapping.xml to define the mapping between the service type definition of a service in OM and the type of a CI in the RTSM.

For the ACME example, the type mapping is defined in the following table.

OM Service Type	CI Type (CI Name)
ACME_System	acme_system
ACME_Application_Server	acme_appserver

Here is an example configuration of the type mapping file typemapping.xml for the ACME synchronization package, using the context ACME_Landscape. OM service elements of type ACME_System are mapped to CIs of type acme_system in the RTSM. OM service elements of type ACME_Application_Server are mapped to CIs of type acme_appserver in the RTSM.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../schemas/mapping.xsd">
  <Rules Context="ACME_Landscape">
    <Rule name="Map ACME System">
      <Condition>
        <Equals>
          <OMType />
          <Value>ACME_System</Value>
        </Equals>
      </Condition>
      <MapTo>
        <CMDBType>
          <Value>acme_system</Value>
        </CMDBType>
      </MapTo>
    </Rule>
    <Rule name="Map ACME Application Server">
      <Condition>
        <Equals>
          <OMType />
          <Value>ACME_Application_Server</Value>
        </Equals>
      </Condition>
      <MapTo>
        <CMDBType>
          <Value>acme_appserver</Value>
        </CMDBType>
      </MapTo>
    </Rule>
  </Rules>
</Mapping>
    
```

Configure Attribute Mapping File: attributemapping.xml

Configure the attribute mapping file `attributemapping.xml` to define the mapping between the attributes of a service in OM and the attributes of a CI in the RTSM.

For the ACME example, the following table shows which OM service attributes are mapped to which CI attributes in the RTSM.

Service Type	Service Attribute Name	CI Type	CI Attribute Name
ACME_System	Caption	ALL	display_label
ACME_Application_Server	OMId	ALL	data_name

Here is an excerpt of the corresponding configuration of the attribute mapping file `attributemapping.xml` for the ACME synchronization package. This shows two attribute mappings:

- For the OM service elements of type `ACME_system`, the OM service attribute `Caption` is mapped to the CI attribute `display_label` in the RTSM.
- For the OM service elements of type `ACME_Application_Server`, the OM service attribute `OMId` is mapped to the CI attribute `data_name` in the RTSM.

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../schemas/mapping.xsd">
  <Rules Context="ACME_Landscape">
    <Rule name="Map ACME System attributes">
      <Condition>
        <Equals>
          <OMType />
          <Value>ACME_System</Value>
        </Equals>
      </Condition>
      <MapTo>
        <CMDBAttribute>
          <Name>display_label</Name>
          <SetValue>
            <Caption />
          </SetValue>
        </CMDBAttribute>
      </MapTo>
    </Rule>
    <Rule name="Map ACME Application Server attributes">
      <Condition>
        <Equals>
          <OMType />
          <Value>ACME_Application_Server</Value>
        </Equals>
      </Condition>
      <MapTo>
        <CMDBAttribute>
          <Name>data_name</Name>
          <SetValue>
            <OMId />
          </SetValue>
        </CMDBAttribute>
      </MapTo>
    </Rule>
  </Rules>
</Mapping>
```

Configure Relation Mapping File: relationmapping.xml

Configure the relation mapping file `relationmapping.xml` to define the CI relationships created in the RTSM between specified OM services.

Here is an example configuration of the relation mapping file `relationmapping.xml` for the ACME synchronization package. This shows the creation of the following relations:

- The `root_container` CI attribute of CIs with the OM service type `ACME_Application_Server` is set to the host. Additionally, a `container_f` relation is created implicitly between the host and the CI.
- A composition relation `container_f` between OM service elements of type `ACME_System` and `ACME_Application_Server`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../schemas/mapping.xsd">
  <Rules Context="ACME_Landscape">
    <Rule name="Create relation ACME Application Server to node">
      <Condition>
        <StartsWith>
          <OMType />
          <Value>ACME_Application_Server</Value>
        </StartsWith>
      </Condition>
      <MapTo>
        <RootContainer>
          <DependencyCI relationType="hosted_on">
            <True />
          </DependencyCI>
        </RootContainer>
      </MapTo>
    </Rule>
    <Rule name="Create relation between ACME Application Server and ACME
System">
      <Condition>
        <And>
          <Equals>
            <OMType />
            <Value>ACME_Application_Server</Value>
          </Equals>
          <Equals>
            <AncestorCI relationType="container_f">
              <Equals>
                <OMType/>
                <Value>ACME_System</Value>
              </Equals>
            </AncestorCI>
            <ParentCI/>
          </Equals>
        </And>
      </Condition>
      <MapTo>
        <RelationFrom>
```

```
        <From>
          <AncestorCI relationType="container_f">
            <Equals>
              <OMType/>
              <Value>ACME_System</Value>
            </Equals>
          </AncestorCI>
        </From>
        <Type>member</Type>
      </RelationFrom>
    </MapTo>
  </Rule>
</Rules>
</Mapping>
```

Customizing Synchronization and Scripting

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

For details, see ["Scripting" on page 79](#). For more information about developing and deploying Groovy scripts, see ["Groovy Scripts" on page 335](#).

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:

```
<OMi_HOME>/conf/opr/topology-sync/sync-packages/<SyncPackageName>
```

Chapter 12: Scripting

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

- Pre-mapping scripts are executed before the mapping rules are applied.
- Pre-upload scripts are executed after mapping, but before the upload of data to the RTSM (Run-time Service Model).
- Post-upload scripts are executed after the upload of data to the RTSM.

One script of each type can be associated with each synchronization package. These optional script files are located in the associated synchronization package directory. For details of synchronization package locations, see ["Synchronization Package Locations" on page 78](#).

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

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

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

Groovy Scripts

Groovy scripting is supported. Groovy is a high level, object-oriented scripting language for the Java platform, which compiles down to Java bytecode.

Groovy is aimed at Java developers, and is tightly integrated with the Java platform. It provides you with a similarly powerful and concise coding syntax to that provided by languages such as Python or Ruby, while enabling you to stay on the Java Virtual Machine (JVM). Java beans are supported, and Java and Groovy classes are interchangeable inside the JVM. As Groovy integrates well with Java code and libraries, and enables you to reuse the semantics of Java, and all J2SE and J2EE APIs, you do not have to learn, implement and maintain new semantics and APIs.

For more information about developing and deploying Groovy scripts, see ["Groovy Scripts" on page 335](#).

There are three Groovy scripts that can be placed into a topology synchronization package, and these are identified using fixed names within synchronization package directories. Each script runs at a defined point in the synchronization process:

- `preEnrichment.groovy` — script to be executed before mapping
The `preEnrichment.groovy` script is executed before starting the topology synchronization's mapping process.

- `preUpload.groovy` — script to be executed before upload
The `preUpload.groovy` script is executed after the mapping process but before writing any data to the RTSM (Run-time Service Model) (for example, to create additional CIs or add extra details to existing CI instances).
- `postUpload.groovy` — script to be executed after upload
The `postUpload.groovy` script is executed after saving the uploaded data in the RTSM, to modify data saved during the upload process (for example, for logging or auditing purposes).

The upload is performed between execution of the `preUpload.groovy` scripts and the `postUpload.groovy` scripts.

Enabling and Disabling Scripts

By default, the use of Groovy scripts is enabled (in the OM Topology Synchronization settings, the `Enable usage of Groovy scripts` value is set to `true`).

To help identify the cause of synchronization failure, you can disable scripting. If there is an error in a script, disabling scripting should make successful synchronization more likely.

To disable topology synchronization package script execution, change the `Enable usage of Groovy scripts` setting from `true` to `false` in the OM Topology Synchronization settings in the Infrastructure Settings manager:

Administration > Setup and Maintenance > Infrastructure Settings

Operations Management - HPOM Topology Synchronization > Enable usage of Groovy scripts

Groovy Script Location

The Groovy scripts must be located in the same directory as the topology-synchronization mapping rules:

```
<OMi_HOME>/conf/opr/topology-sync/sync-packages/<SyncPackageName>
```


Script Variables

Each script has two predefined variables:

- `ScriptInterface`

Object Type: `com.hp.opr.ts.scripting.ScriptInterface`

Description: Enables access to CI information function calls to manipulate synchronization data and control the synchronization.

The object type implements the following interface:

`com.hp.opr.ts.interfaces.scripting.IScriptingInterface`

- `SyncData`

Object Type: `com.hp.opr.ts.common.data.sync.SyncData`

Description: Provides access to the data that is synchronized.

The object type implements the following interface:

`com.hp.opr.ts.interfaces.data.sync.ISyncData`

For more information about the interfaces and object types required for developing your own topology synchronization scripts, see the Java API Documentation document located in the following directory:

`<OMi_HOME>/opr/api/doc/opr-ts-interfaces-javadoc.zip`

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

Handling Errors

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

```
scriptInterface.setAbortSyncOnError(boolean)
```

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

The following table shows the relationship between synchronization status (successful or unsuccessful synchronization) and script behavior.

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

Sample Script: preUpload.groovy

The following script is an extract of a sample `preUpload.groovy` script:

```
import com.hp.opr.ts.interfaces.data.ci.* ;
import com.hp.opr.ts.common.data.ci.* ;
import java.util.*;
import java.lang.String;

List resourceGroups = new LinkedList ();
List haMembers      = new LinkedList ();

// Get all HPOM services, hosts and node groups
for (ICi ci : syncData.getConfigurationsItems()) {

    if (ci.getOmTypeId() == "Class_RG") {
// If type is "Class_RG", then create a CI of type IP for all entries // of the
HPOM attribute ip address

        scriptInterface.logInfo ("add resource group");
        resourceGroups.add (ci);
    }
}
```

```
    }  
  
}  
  
// Create ip-ci and relationship to the cluster package  
for (ICi ipCi : resourceGroups) {  
    HashMap hm = new HashMap();  
// Get HPOM service-specific attributes  
    hm = ipCi.getOmAttributes();  
  
// Create CI for ip-address attribute  
    ICi newCi = scriptInterface.createCi();  
    newCi.setContext ("cluster");  
    newCi.setCmdbAttribute ("ip_address", hm.get("ipaddr"));  
    newCi.setCmdbAttribute ("ip_domain", "\\${DefaultDomain}");  
    newCi.setCmdbTypeId ("ip");  
    scriptInterface.logInfo ("create relationship between two ip-ci: "  
        + hm.get("ipaddr") + " and cluster package " );  
// Create the "contained" relationship between the cluster package // and ip  
    scriptInterface.createCmdbRelation(ipCi, newCi, "contained");  
  
}  
}
```

Chapter 13: Testing and Troubleshooting

This chapter contains information on:

- "Validating XML Configuration Files" below
- "Dumping Synchronization Data" on page 87
- "Writing Rules" on page 91
- "Log Level Configuration" on page 93
- "Troubleshooting, Common Issues, and Tips" on page 94

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 XML Schema Definition 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>.

XSD Files

The schema files are stored in the following directory:

```
<OMi_HOME>/conf/opr/topology-sync/schemas
```

The files are:

package.xsd

Validates the package.xml file in each synchronization package.

containmentrelations.xsd

Validates the containmentrelations.xml file.

datadump.xsd

Validates synchronization data files that are created through enabling data dumps or used as input for the enrichment simulator.

mapping.xsd

Validates the following mapping files contained in the synchronization packages:

- Context mapping - contextmapping.xml
- Type mapping - typemapping.xml
- Attribute mapping - attributemapping.xml
- Relation mapping - relationmapping.xml

nodetypes.xsd

Validates the node type mapping file nodetypes.xml file in each synchronization package.

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

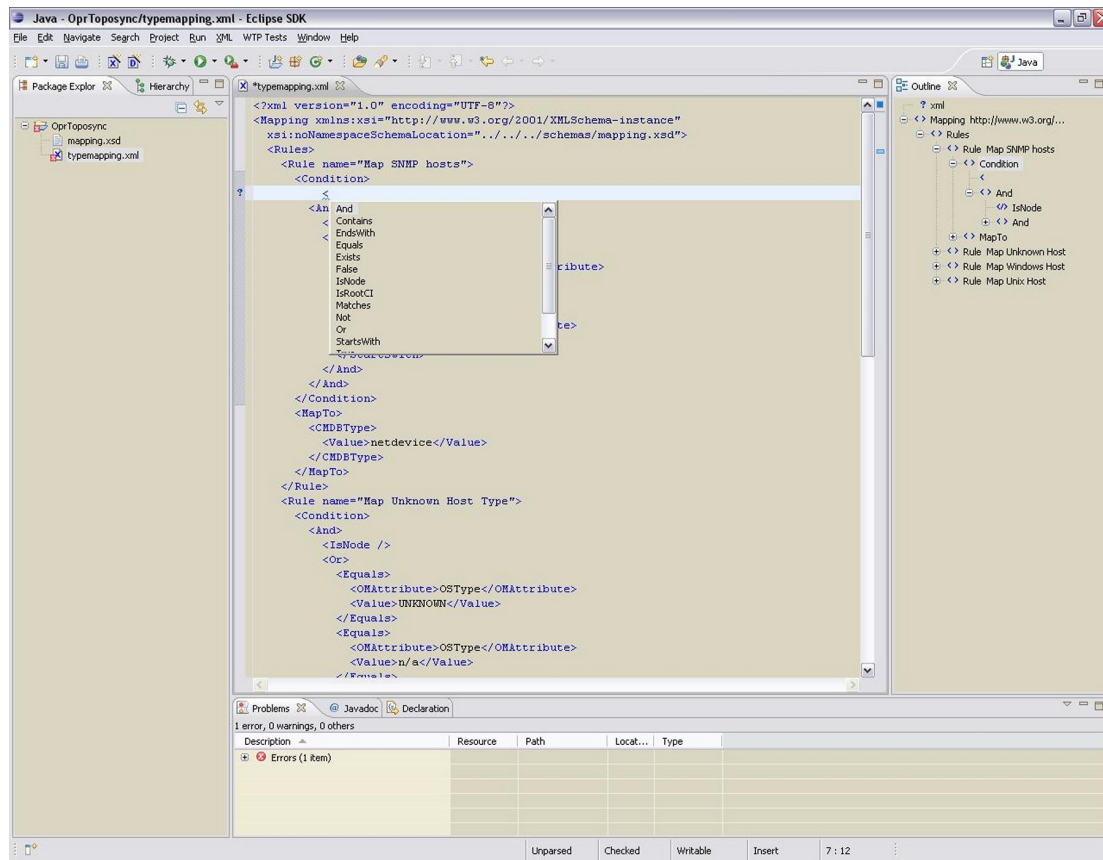
With a modern XML editor, you can validate a file against a schema. Eclipse, for example, can validate an XML file against a schema, if the top level element of the document contains a reference to an XSD file. To enable validation, add the following attributes to the top level element of an XML file:

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

Replace *<path or URL to schema file>* with the respective path or URL to the schema file against which you want to validate. For example, for a `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="<OMI_HOME>/conf/opr/topology-  
sync/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.



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 RTSM, and the data changed and added during the mapping.
- Create a dump file to check XPath expressions of rules.

Creating a Synchronization Data Dump

A synchronization data dump contains the synchronized topology data 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. Navigate to the OM Topology Synchronization settings in the Infrastructure Settings Manager:

Administration > Setup and Maintenance > Infrastructure Settings

Operations Management - HPOM Topology Synchronization Settings > Dump data

2. Change the value of **Dump data** to **true**.
3. Start the topology synchronization in OM with the following command:

- **OM for Windows.** Start the synchronization of topology data:

- i. In the console tree, select **Tools > HP Operations Manager Tools**.
- ii. Right-click **Synchronize Topology** and select **All Tasks > Launch Tool...**

The `startInitialSync.bat` tool is started and begins to send all the topology data to the target management servers.

- **OM for UNIX or Linux.** Type the following command to start the synchronization of topology data:

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

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>
    <CMDBId />
    <CMDBAttributes>
      <data_name>03a2f7b2-ec88-7539-0532-c5b07da188dd</data_name>
    </CMDBAttributes>
    <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>
      </CI>
    </Dependencies>
  </CI>
</Children>
```



```
</OMAttributes>  
<CMDBId />  
<CMDBAttributes>  
  <host_dnsname>met.deu.hp.com</host_dnsname>  
  <host_hostname>met</host_hostname>  
  <host_key>met.deu.hp.com</host_key>  
  <host_os>2000 (5.0)</host_os>  
</CMDBAttributes>  
<CMDBType>nt</CMDBType>  
<RootContainerId />  
</CI>  
</Dependencies>  
</CI>  
</Children>  
</CI>
```

Viewing a Synchronization Data Dump

To view synchronization data dumps, navigate to the directory:

```
<OMi_HOME>/opr/tmp/datadump
```

The directory contains the following subdirectories:

- pre-enrichment
Contains the synchronization data after the CI data structure has been normalized. The data reflects what has been loaded from OM to the RTSM (Run-time Service Model).
- post-enrichment
Contains the synchronization data after the mapping rules have been executed on the normalized data.
- ws-data
Contains raw data which was read from the OM web service. For each OM node, node group, and service, there is an XML file called `Caption_OMId.xml`.
Only in the case where writing to the RTSM failed, the XML file is written to the `post-ucmbd` directory.

Validating Mapping Rules

To validate mapping rules, complete the following steps:

1. Compare file differences.

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

2. Validate XPath expressions.

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

Note: 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.

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 84](#) for more information.

Avoiding Complex XPath Queries

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

Matching Against Existing Attributes Only

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

Avoiding Broad XPath Expressions

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

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

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

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

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

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).
- Log level 10 logs tracing information for debugging purposes, for example method parameters.

To change the log level of the service discovery server:

1. In a command prompt, run the following command:

```
ovconfchg -ovrg server -edit
```

You can see the log in a Notepad window.
2. Edit the file by adding `LOG_LEVEL=10` to the `[om.svcdiscserver]` namespace.

The service discovery server generates the following log file:

Windows: %0vShareDir%\server\log\0vSvcDiscServer.log

Linux: /var/opt/0V/shared/server/log/0vSvcDiscServer.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:

```
<OMi_HOME>/conf/core/Tools/log4j/wde/opr-svcdiscserver.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:

```
<OMi_HOME>/log/wde/opr-svcdiscserver.log
```

Troubleshooting, Common Issues, and Tips

The log files specified under "[Topology Synchronization File Locations](#)" on page 64 are a good starting point for troubleshooting.

The most common issues are listed in the following table. The issues apply to topology synchronization generally, unless otherwise specified.

Symptom	Cause	Solution
Topology synchronization fails.	<p>Required patches for OM for Windows were not installed.</p> <p>See the OMi Readme for details, including information about any required agent hotfixes or patches.</p>	<p>Operations Manager for Windows:</p> <p>Install Patches OMW_00138 or superseding and OMW_00123 for OM 8.1x for Windows.</p> <p>Install Patches OMW_00139 or superseding and OMW_00124 for OM 9.00 for Windows.</p> <p>See the <i>OMi 9.10 Release Notes</i> for details.</p> <p>Operations Manager for UNIX or Linux:</p> <p>Install Patch PHSS_42736 or superseding for OM 9.10 for HP-UX.</p> <p>Install Patch OML_00050 or superseding for OM 9.10 for Linux.</p> <p>Install Patch ITOSOL_00772 or superseding for OM 9.10 for Solaris.</p>
Topology synchronization fails.	Synchronization package was changed on disk, but not uploaded to the database.	Run the <code>opr-sdtool</code> command-line tool to upload changes to synchronization packages to the database (see " Managing Synchronization Packages " on page 65).
Result of <i>dynamic</i> topology synchronization is incomplete or empty.	Discovery policies (<code>DiscoverOMTypes</code> and <code>DiscoverOM</code>) are deployed prior to configuring the OMi instance as a target server in OM.	On the OM management server, run the command <code>ovagtrep -publish</code> . This command resends all topology data to the OM or OMi instances.
All of a sudden, no more node CIs are created and the synchronization fails.	The default synchronization package was removed from the Topology Synchronization settings.	Check if the default synchronization package was removed from the Topology Synchronization settings. The default package must always be present in the semicolon-separated list.

Symptom	Cause	Solution
Warnings in the log file.	Model-related issues.	No immediate action required, however topology synchronization performance can be affected.
You created your own synchronization package but you only get a cryptic RTSM exception in the log file.	Mapping-related issues.	Enable the data dump option and check if the file in the <code><OMi_HOME>/opr/tmp/datadump/post-enrichment</code> directory contains all expected attributes for the CIs of your synchronization package.

Limitations

This section describes some known limitations relating to topology synchronization.

Delta Detection Limitations

If an attribute of a service or node in OM changes, and that attribute or node is mapped to a key attribute in the RTSM, the delta detection does not delete and replace the old RTSM CI instance, but generates an additional new instance.

Here is an example to illustrate this. Consider a managed node in OM for Windows that has an agent ID of `aaaaa-bbbb-cccc-dddd`. This node maps to a host CI in the RTSM and an agent CI with the key `aaaaa-bbbb-cccc-dddd`.

In OM, a change is made to the agent ID, so that the agent ID is now `aaaaa-bbbb-cccc-eeee`. A new agent CI with the key attribute `aaaaa-bbbb-cccc-eeee` is created, but the old one is not deleted. So there are now two RTSM instances relating to the same (changed) managed node.

Workaround: To overcome this limitation, you need to manually delete the old RTSM instance.

Topology Synchronization Limitation

Event changes from the OM discovery server are registered with the WMI Listener, and forwarded through the WMI to the RTSM. Under certain circumstances, for example, due to high loads on the WMI, it could happen that not all events arrive in OMi and so are not reflected in the RTSM. As a result, this could lead to a temporary discrepancy between the status in OM and the RTSM. This discrepancy will be reconciled the next time the tool `startInitialSync.bat` runs.

Chapter 14: Mapping Engine and Syntax

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

- ["Common Mapping File Format" below](#)
- ["Mapping File Syntax" on the next page](#)
- ["XPath Navigation" on page 117](#)

Common Mapping File Format

Note: 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 Context="web server SPI">
    <Rule name="Apache Server">
      <Condition>
        <!-- ... Boolean operators ... -->
      </Condition>
      <MapTo>
        <!-- ... Target Mappings ... -->
      </MapTo>
    </Rule>
    <!-- ... More Rules ... -->
  </Rules>
  <!-- ... More Rule sets with different contexts ... -->
</Mapping>
```

The components of the mapping files are described in ["Mapping File Syntax" on the next page](#).

Mapping File Syntax

The following sections describe the valid syntax used in topology synchronization mapping files.

- ["Rules" below](#)
- ["Rule Conditions" below](#)
- ["Operator Elements" on page 99](#)
- ["Operand Elements" on page 102](#)
- ["Mapping Elements" on page 109](#)
- ["Filtering" on page 109](#)
- ["Type Mapping" on page 110](#)
- ["Attribute Mapping" on page 111](#)
- ["Relation Mapping" on page 115](#)

Rules

The `<Rules>` tag contains a set of rules. By using the optional `Context` attribute you can restrict these rules to a certain context. See ["Filtering" on page 109](#) for more information.

Rule Conditions

The `<Condition>` element of a rule contains a Boolean operator that specifies how the individual conditions relate to each other, and, for example, is similar to the `<condition>` task in Ant.

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>`.

Condition Examples

Check if the type of the current OM service is testtype.

Example:

```
<Condition>  
  <Equals>  
    <OMType/>  
    <Value>testtype</Value>  
  </Equals>  
</Condition>
```

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

Example:

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

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 OM.

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://docs.oracle.com/javase/7/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

OM Service ID

<OMId/>

Return type: String

Returns the OM ID string of the CI as stored in OMi. The OM ID returns different values as follows:

Services: OM ID is the service ID

Nodes: OM ID is the unique ID

Node Groups: OM ID is the node group ID

OM Type

<OMType/>

Return type: String

Returns the OM Type stored in OMi. For OM services, the OM Type is the service type definition. For nodes, the OM Type is set to the constant value "node".

CMDB Type

<CMDBType/>

Return type: String

Returns the CMDB CI Type ID string of the CI as it is stored in the RTSM. Initially this is returned as null because the CMDB type must initially be set in the Type Mapping. After this is set, the CMDB CI Type ID string should be available.

Caption

<Caption/>

Return type: String

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

OM Attribute

<OMAttribute>[Name]</OMAttribute>

Return type: String

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

CMDB Attribute

<CMDBAttribute>[Name]</CMDBAttribute>

Return type: String

Returns the value of the CMDDB attribute with the given name as it will be written to the RTSM. There are no attributes available until the attribute mapping has been performed.

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://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html>

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

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

```
</By>  
</Replace>
```

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

XPath Result

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

Return type: String

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

For more information on XPath, see ["XPath Navigation" on page 117](#).

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 117](#).

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

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

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

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

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

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

```
<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.

Mapping Elements

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

Mapping Examples

Map the OM service attribute Caption to the CI attribute display_label in the RTSM.

Example:

```
<MapTo>
  <CMDBAttribute>
    <Name>display_label</Name>
    <SetValue>
      <Caption />
    </SetValue>
  </CMDBAttribute>
</MapTo>
```

Map the OM service attribute OMID to the CI attribute data_name in the RTSM.

Example:

```
<MapTo>
  <CMDBAttribute>
    <Name>data_name</Name>
    <SetValue>
      <OMId />
    </SetValue>
  </CMDBAttribute>
</MapTo>
```

Filtering

Filtering allows to select interesting parts of topology data by assigning a context to these CIs. This context allows to selectively apply mapping rules to CIs of the same context. All CIs that have no context attached will not be synchronized.

Note: The <Rules> tag for filter mapping rules *must not* contain the Context attribute.

Context Mapping

```
<Context>[Context Name]</Context>
```

In the following example, all CIs that are assigned to the service type definition `exch_spi_std_server_role` and that are below a service with a service type definition `exch_spi_std_server` are assigned to the exchange context.

Example:

```
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="../mapping.xsd">
  <Rules>
    <Rule name="Exchange Server Role Filter">
      <Condition>
        <And>
          <Exists>
            <XPathResult>ancestor::ci[omType='exch_spi_std_server']
            </XPathResult>
          </Exists>
          <Equals>
            <OMType/>
            <Value>exch_spi_std_server_role</Value>
          </Equals>
        </And>
      </Condition>
      <MapTo>
        <Context>exchange</Context>
      </MapTo>
    </Rule>
  </Rules>
</Mapping>
```

Type Mapping

The service mapping maps the OM service type definitions to their CMDB types.

Mapping

Maps the CI to the specified CMDB type that is the concatenated string of the results of the operators. There must not be more than one `<CMDBType>` element in the `<MapTo>` section.

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

In the following example, all CIs that have an OM Type `Exch2k7_ByServer` and that have the context `exchange` assigned are mapped to the CMDB Type `exchangeserver`.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<Mapping>
```

```
<Rules context="exchange">
  </Rule>
  <Rule name="Map Exchange Server">
    <Condition>
      <Equals>
        <OMType/>
        <Value>Exch2k7_ByServer</Value>
      </Equals>
    </Condition>
    <MapTo>
      <CMDBType>
        <Value>exchangeserver</Value>
      </CMDBType>
    </MapTo>
  </Rule>
</Rules>
</Mapping>
```

In the following example, all nodes that have an attribute `OSType` that starts with the string `windows` are mapped to the `CMDB` type `nt`.

Example:

```
<Mapping>
  <Rules>
    <Rule name="Map Windows Host Type">
      <Condition>
        <And>
          <IsNode/>
          <StartsWith>
            <OMAttribute>OSType</OMAttribute>
            <Value>Windows</Value>
          </StartsWith>
        </And>
      </Condition>
      <MapTo>
        <CMDBType>
          <Value>nt</Value>
        </CMDBType>
      </MapTo>
    </Rule>
  </Rules>
</Mapping>
```

Attribute Mapping

The attribute mapping file `attributemapping.xml` defines the mapping between the attributes of a service in OM and the attributes of a CI in the RTSM.

Set 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 will be concatenated.

Mapping to String Values

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

Mapping to String Values of a Maximum Length

```
<CMDBAttribute>  
  <Name>[Attribute Name]</Name>  
  <SetValue Length="[IntegerValue]">  
    [Operands]  
  </SetValue>  
</CMDBAttribute>
```

Mapping to Integer Values

```
<CMDBAttribute>  
  <Name>[Attribute Name]</Name>  
  <SetIntValue>  
    [Operands]  
  </SetIntValue>  
</CMDBAttribute>
```

Mapping to Boolean Values

```
<CMDBAttribute>  
  <Name>[Attribute Name]</Name>  
  <SetBoolValue>  
    [Operands]  
  </SetBoolValue>  
</CMDBAttribute>
```

Mapping to Long Values

```
<CMDBAttribute>  
  <Name>[Attribute Name]</Name>  
  <SetLongValue>  
    [Operands]  
  </SetLongValue>
```



```
</CMDDBAttribute>
```

Mapping to Date Values

```
<CMDDBAttribute>  
  <Name>[Attribute Name]</Name>  
  <SetdateValue>  
    [Operands]  
  </SetdateValue>  
</CMDDBAttribute>
```

Mapping to Float Values

```
<CMDDBAttribute>  
  <Name>[Attribute Name]</Name>  
  <SetFloatValue>  
    [Operands]  
  </SetFloatValue>  
</CMDDBAttribute>
```

Mapping to Byte Values

```
<CMDDBAttribute>  
  <Name>[Attribute Name]</Name>  
  <SetByteValue>  
    [Operands]  
  </SetByteValue>  
</CMDDBAttribute>
```

Mapping to Double Values

```
<CMDDBAttribute>  
  <Name>[Attribute Name]</Name>  
  <SetDoubleValue>  
    [Operands]  
  </SetDoubleValue>  
</CMDDBAttribute>
```

Mapping to StringList Values

```
<CMDDBAttribute>  
  <Name>[Attribute Name]</Name>  
  <SetStringListValue>  
    [Operands] (comma-separated)
```

```
</SetStringListValue>  
</CMDBAttribute>
```

Mapping to IntList Values

```
<CMDBAttribute>  
  <Name>[Attribute Name]</Name>  
  <SetIntListValue>  
    [Operands] (comma-separated)  
  </SetIntListValue>  
</CMDBAttribute>
```

Attribute Mapping Example

For all CIs (no matter which context is assigned) the CMDB attribute `display_label` is set to the Caption of the OM CI. CIs that are assigned to the context `exchange` have `data_name` and for nodes the `host_key` attribute set to the OM ID.

```
<Mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:noNamespaceSchemaLocation="../../schemas/mapping.xsd">  
  <Rules>  
    <Rule name="Map Display Label">  
      <Condition>  
        <True/>  
      </Condition>  
      <MapTo>  
        <CMDBAttribute>  
          <Name>display_label</Name>  
          <SetValue>  
            <Caption/>  
          </SetValue>  
        </CMDBAttribute>  
      </MapTo>  
    </Rule>  
  </Rules>  
  <Rules Context="exchange">  
    <Rule name="Set data_name key attribute">  
      <Condition>  
        <True/>  
      </Condition>  
      <MapTo>  
        <CMDBAttribute>  
          <Name>data_name</Name>  
          <SetValue>  
            <OMId/>  
          </SetValue>  
        </MapTo>  
      </Rule>  
    </Rules>  
  </Mapping>
```

```
    </CMDBAttribute>
  </MapTo>
</Rule>
<Rule name="Set host_key key attribute for nodes">
  <Condition>
    <IsNode/>
  </Condition>
  <MapTo>
    <CMDBAttribute>
      <Name>host_key</Name>
      <SetValue>
        <OMId/>
      </SetValue>
    </CMDBAttribute>
  </MapTo>
</Rule>
</Rules>
</Mapping>
```

Relation Mapping

Using the relation mapping you can create relations between CIs. For topology synchronization, the OM associations are not synchronized as relations by default. You must explicitly define these relations.

```
<RelationTo>
  <To>
    [Operand]
  </To>
  <Type>[RelationType]</Type>
</RelationTo>
```

Define a relation from the current CI to the CI that is returned by the operand. The operand may either return a string, an instance of a CI, a list of CIs or a list of strings. String values must match the OM ID of the CI to which the relation is created. In the case of a list, a relation is created for each item (that is in turn either a string or a CI) contained in the list.

The relation has the type specified by [RelationType]. This type is the name of the relation, not the label.

```
<RelationFrom>
  <From>
    [Operand]
  </From>
  <Type>[RelationType]</Type>
</RelationFrom>
```

Works just as the previous mapping, but in the opposite direction.

Root Container Mapping

The CMDB model defines certain root container CIs that have to be created before the actual CI can be created. Topology synchronization must know such relations to be able to create the CIs in the correct order.

```
<RootContainer>  
  [Operand]  
</RootContainer>
```

The root container of the current CI is set to the CI specified by the return value of the operand. The return value may either be a String or a CI.

Message Alias Mapping for CI Resolution

```
<RedirectMessagesOf>  
  [Operand]  
</RedirectMessagesOf>
```

The aliases of the current CI is set to the OMId(s) specified by the return value of the operand. The return value may either be a String or a CI or a list of CIs or Strings.

In the following example, the CI with the STD Exch2k7_ByServer gets a relation of the type `is_impacted_from` to the node on which it is hosted and a relation of the type `deployed` to all descendant CIs with an OM Type that starts with `Exch2k7_Role_`.

The same node is also the root container CI.

Example:

```
<Mapping>  
  <Rules Context="exchange">  
    <Rule name="Create relation server to node">  
      <Condition>  
        <Equals>  
          <OMType/>  
          <Value>Exch2k7_ByServer</Value>  
        </Equals>  
      </Condition>  
      <MapTo>  
        <RelationTo>  
          <To>  
            <DependencyCI relationType="hosted_on">  
              <True/>  
            </DependencyCI>  
          </To>  
          <Type>is_impacted_from</Type>  
        </RelationTo>  
        <RelationTo>  
          <To>  
            <DescendantCIList>
```

```
        <StartsWith>
            <OMType>
                <Value>Exch2k7_Role_</Value>
            </StartsWith>
        </DependencyCI>
    </To>
    <Type>deployed</Type>
</RelationTo>
<RootContainer>
    <DependencyCI relationType="hosted_on">
        <True/>
    </DependencyCI>
</RootContainer>
<RedirectMessagesOf>
    <ChildCIList/>
</RedirectMessagesOf>
</MapTo>
</Rule>
</Rules>
</Mapping>
```

XPath Navigation

XPath is a syntax for defining parts of an XML document. XPath uses path expressions to navigate in XML documents.

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/xsl/xpath_intro.asp

Data Structure

The data structure that is exposed to the XPath expression matching used in mapping rules is shown in ["Example of an XPath-Navigated Data Structure" on page 120](#).

CI Data Structure

- **OMAttributes**

Contains a map of all original RTSM CI attributes. The key of this map is the name of the RTSM CI attribute that references the RTSM value of the RTSM CI attribute.

- **Caption**

Represents the name of the CI to be displayed in Service Navigator. Caption has the same value as the RTSM CI attribute `display_label`.

- **Children**

References a list of relations to CIs that have a containment relationship from the current CI to other CIs. Using this field, you can create complex XPath queries to retrieve values of children as well as parents using the `..` XPath selector.

- **Dependencies**

References a list of relations to dependent CIs. Similar to `Children`. However, referenced objects are contained in a different hierarchy.

- **OMid**

Unique ID of a CI.

- **Node**

Boolean value that indicates whether this is a node in OM.

- **Type**

Contains the service type of an OM service.

This is not the display label of the CI Type.

- **Service**

Boolean value that indicates whether this element is a service in OM.

Node groups have `Node` and `Service` set to `FALSE`.

Relationship Data Structure

- **CI**

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

• **RelationType**

Relationship type as stored in the RTSM.

This is not the display label of the CI Type.

For services:

Contains `container_f` if it is a containment relation in OM.

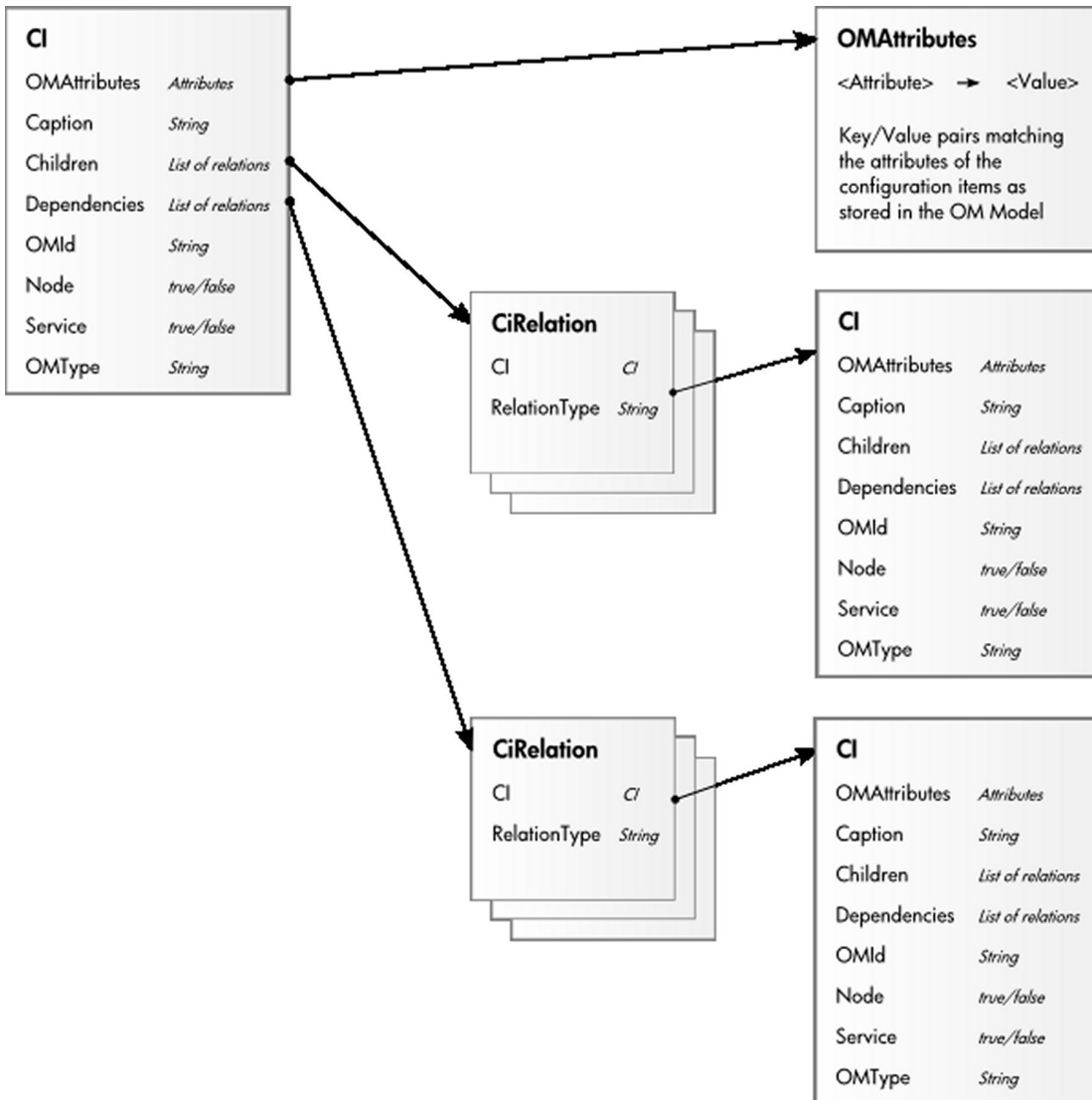
Contains `dependency` if it is a dependency relation in OM.

Contains `hosted_on` if it is a hosted on relation in OM.

For nodes:

Contains `container_node` or `dependency_node`.

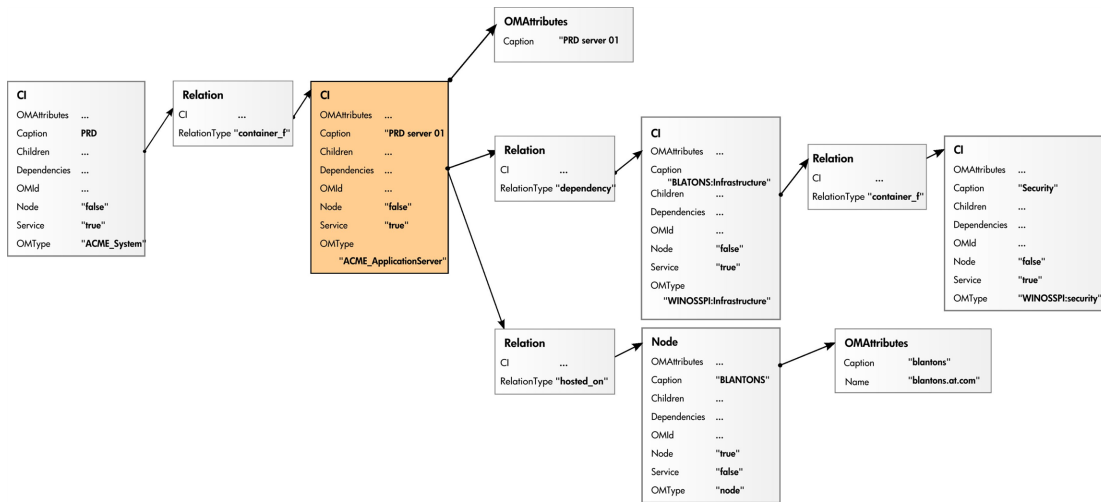
The following figure illustrates the data structure exposed to the navigation.



Example of an XPath-Navigated Data Structure

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

The following figure illustrates some XPath examples.



XPath Expressions and Example Values

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

XPath Expression	Meaning	Example
Caption	Caption from CI	PRD server 01
./Caption	Caption from CI	PRD server 01
/Caption	Caption of the root (database) CIs	PRD
.././Caption	Selects the caption from the parent of the parent CI	PRD
../RelationType	Selects the parent relation type	container_f
../../OMType	Selects the parent of the parent type	ACME_System
/OMType	Selects type of the root CI	ACME_System
//.[type='WINOSSPI:Infrastructure']/Caption	Selects the caption of all CIs of type WINOSSPI: Infrastructure	BLANTONS: Infrastructure
//Dependencies[type='hosted_on']/CI/Caption	Selects the caption of all CIs with a hosted_on dependency	BLANTONS

<code>//Dependencies/CI/Caption</code>	Selects the caption of all CIs that have dependencies	BLANTONS
--	---	----------

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

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

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

Part IV: Event Processing Interface

This section describes the role of event processing scripts and custom actions for modifying and enhancing events during event processing.

This section is structured as follows:

- ["Event Processing Interface" on page 123](#)
- ["Scripts For Custom Actions" on page 132](#)
- ["Creating EPI and Custom Action Scripts" on page 134](#)
- ["EPI Troubleshooting" on page 144](#)

Chapter 15: Event Processing Interface

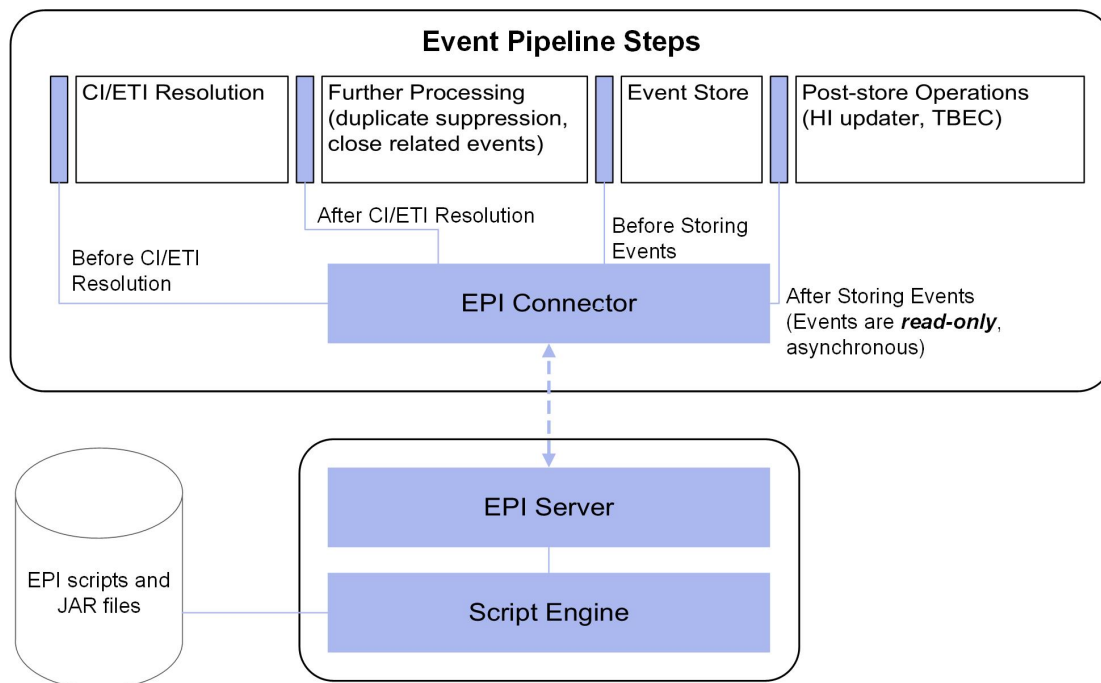
This section provides an overview of the Event Processing Interface (EPI), explaining the different steps in the event pipeline, and appropriate entry points for running scripts for modifying and enriching events.

Event Processing Interface and Scripts

The EPI enables you to run user-defined Groovy scripts during event processing. With these scripts, you can modify and enhance events with external data. For example, you could enrich events with additional data from an external SQL database, or an Excel list. For more information about developing and deploying Groovy scripts, see ["Groovy Scripts" on page 335](#).

["Event processing with pipeline entry points for scripts" below](#) provides an overview of the event processing, showing the event pipeline, and how the EPI script execution can be integrated into this process.

Event processing with pipeline entry points for scripts



Scripts are stored in the database. Any JAR files or libraries that are referenced or used by the scripts are also loaded into the database.

For each pipeline step, you can set up one or more scripts. There is no limit on the number of scripts that can be executed. However, be aware that the time it takes to process events in the pipeline increases with the number of scripts that are executed.

Note: You should design and execute scripts in the context of overall event processing. In other

words, be aware of the interaction of the scripts with other event processing settings in the Settings Manager, for example for duplicate events suppression and closing related events.

Entry Points for Running EPI Scripts

The event pipeline represents the various steps of event processing. There are four points within the event pipeline at which EPI scripts can be executed:

- **Before CI/ETI resolution.** Scripts can be executed directly before the event enters the event pipeline, so before the resolution of CIs and ETIs takes place.
For example, you may want to execute a script at this point that sets further hints that affect the resolution of CIs and ETIs. An entry point further along the event pipeline would be too late to influence the resolution of CIs and ETIs.
- **After CI/ETI resolution.** Scripts can be executed directly after CI/ETI resolution, but before further processing, such as duplicate events suppression and closing related events automatically.
For example, you may want to execute a script at this entry point in the event pipeline if you want to influence how duplicate events are handled. It could be that you have duplicate events suppression enabled in general, but you are interested in changing the duplicate events suppression setting for a particular type of event, while leaving it unchanged for all other event types. So you could execute a script at this entry point that disables duplicate events suppression for the specified event type. Any entry point further along the event pipeline would be too late to influence duplicate events suppression behavior.
- **Before the event is stored in the database.** Scripts can be executed after all event processing has taken place, but before the event is stored in the database.
For example, at this entry point to the event pipeline, you could execute a script that makes changes to some text, or inserts a link to a knowledge base, and so on, before the event gets stored in the database.
- **After the event is stored in the database.** Scripts can be executed after the event has been stored in the database. In this case, the scripts are all executed in read-only mode, since as soon as an event has been stored on the database, it can no longer be modified.
For example, you may want to execute a script at this entry point in the event pipeline to forward events of a particular type, that have already been stored in the database, to another application. Or you could execute a script that writes specified events stored in the database to an audit log.

Specifying a Script

You configure the EPI scripts in the following area of the configuration user interface:

Administration > Event Processing > Automation > Event Processing Customizations

You can create, copy, edit, and delete EPI scripts. You can also determine the order in which the scripts are executed.

For more details about creating scripts, see ["Creating EPI and Custom Action Scripts" on page 134](#).

For details about applying and managing scripts to event processing, see OMi Help.

EPI scripts can be defined in content packs and can be imported/exported using the Content Manager.

EPI Script Execution

The steps involved in script execution are as follows:

1. The EPI server reads the script and calls the `init()` function.
2. The EPI server calls the `process()` function, with the list of events specified as parameter. The script stays in memory, and the `process()` function is called when matching events arrive.
3. The EPI server calls the `destroy()` function if the script is unloaded, for example, if the script is disabled by the user, or at system shutdown.

The `destroy()` function is also called if the script is modified in the Event Processing Customizations manager. In this case the `destroy()` function is called on the old, unmodified script, followed by a call to the `init()` function on the new, modified script.

The call to the `process()` function of a script conforms to the ACID (Atomicity, Consistency, Isolation, Durability) principle that ensures that database transactions are processed reliably.

- **Atomicity**

If a script cannot be successfully executed (does not return without errors, and no exception is thrown), any changes made so far are rolled back. Subsequent events will get the event list as it was before the script in error was executed.

- **Consistency**

Values that violate the consistency of an event can be over-ridden by the system. For example, setting an user ID and a group ID where the user ID is not a member of the group ID is such a violation, and can be over-ridden.

- **Isolation**

Scripts with read/write access to events are not executed in parallel, but sequentially.

- **Durability**

Changes made to events by executing scripts are stored in the database.

EPI Script for Event Enrichment

It is possible to access CI-related data available in the RTSM for a resolved CI and use this information to enrich an event. An EPI groovy script in conjunction with the RTSM JAVA API forms the basis of this approach. See the ["Script Example" on page 129](#) for a working example.

The available data for event enrichment can be any attribute listed under:

Administration > RTSM Administration > Modeling > CI Type Manager

Select a CI type and open the Attributes tab.

Learn More

Basic Script Design

One of the requirements with regards to accessing the RTSM via the JAVA API is that a connection needs to be established first, including providing credentials for a valid user. Establishing the connection involves processing overhead and it is not practical to open and close a new connection for each event being processed. For EPI scripts, you need to minimize the processing time for each event being handled.

To minimize processing time, you can pre-load a simple script-internal HashMap containing CI ObjectIDs linked to a string containing the corresponding value of the CI attribute used for creating our event CA. The pre-loaded HashMap resides in memory, and enables fast retrieval of a small subset of RTSM information.

The `init()` method is used to establish a connection to the RTSM. A separate thread is started in which the HashMap is initialized. This thread is also synchronized with the main thread (via a synchronization object) so that the HashMap can be momentarily blocked and reloaded after a specified wait period (for example 10 minutes).

The `process()` method is used to:

- Retrieve the `ObjectID` of the event's related CI
- Access the HashMap to obtain the "owner" information
- Invoke `event.AddCustomAttributes` to create the CA

The `destroy()` method (called when the script is unloaded or de-activated) is used to release the memory used by the HashMap.

Building the HashMap

To retrieve a set of CIs from the RTSM, a TQL query is executed using the HPE UCMDB API. For full documentation on the available APIs, see HPE UCMDB API Reference. These files are located in the following folder:

```
<OMi_HOME_GW>/AppServer/webapps/site.war/amdocs/eng/API_docs/UCMDB_JavaAPI
```

The script illustrates how to execute a "named query" (a query that has already been defined in the RTSM).

After CIs are retrieved via the query, the HashMap must be initialized. For each CI in the query result, the `ObjectID` and the corresponding string value of the attribute (obtained via the `ci.getPropertyValue` method) are loaded. Any string attribute can be used.

Note: The query should return CIs which actually have the desired attribute defined (and ideally populated) in the RTSM.

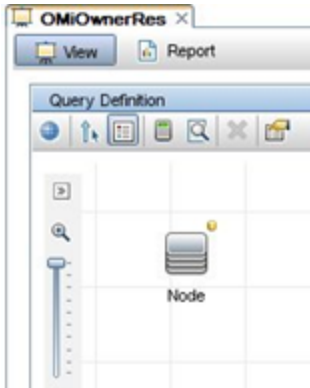
The background thread loads a parallel map (`newMap`) in the part of the code that is not synchronized with the main thread. When the thread gets the lock (is synchronized), it initializes the "run-time" HashMap (`ownerMap`) from the temporary one. This is a safe and effective approach to keeping the HashMap up to date to reflect ongoing changes in the RTSM topology

How to Use the OwnerResolver Script

The following procedure describes how to use a script to enrich events during event processing after the CI has been resolved with CI attributes from the RTSM:

1. Define a TQL query in the Modeling Studio that returns the CIs of interest:

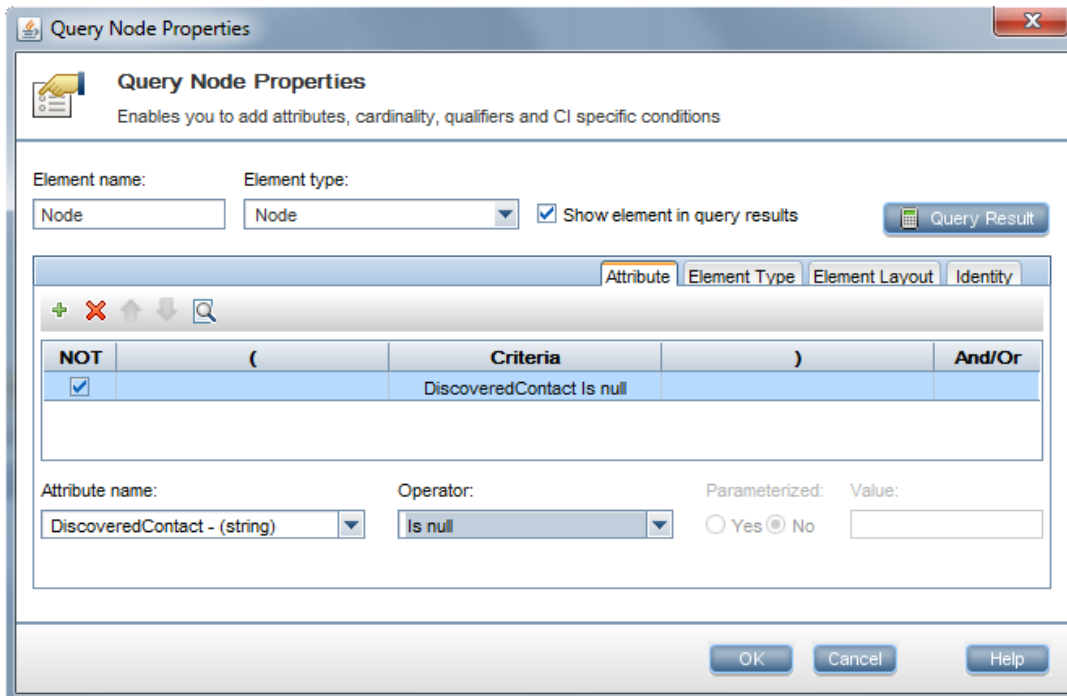
Administration > RTSM Administration > Modeling > Modeling Studio



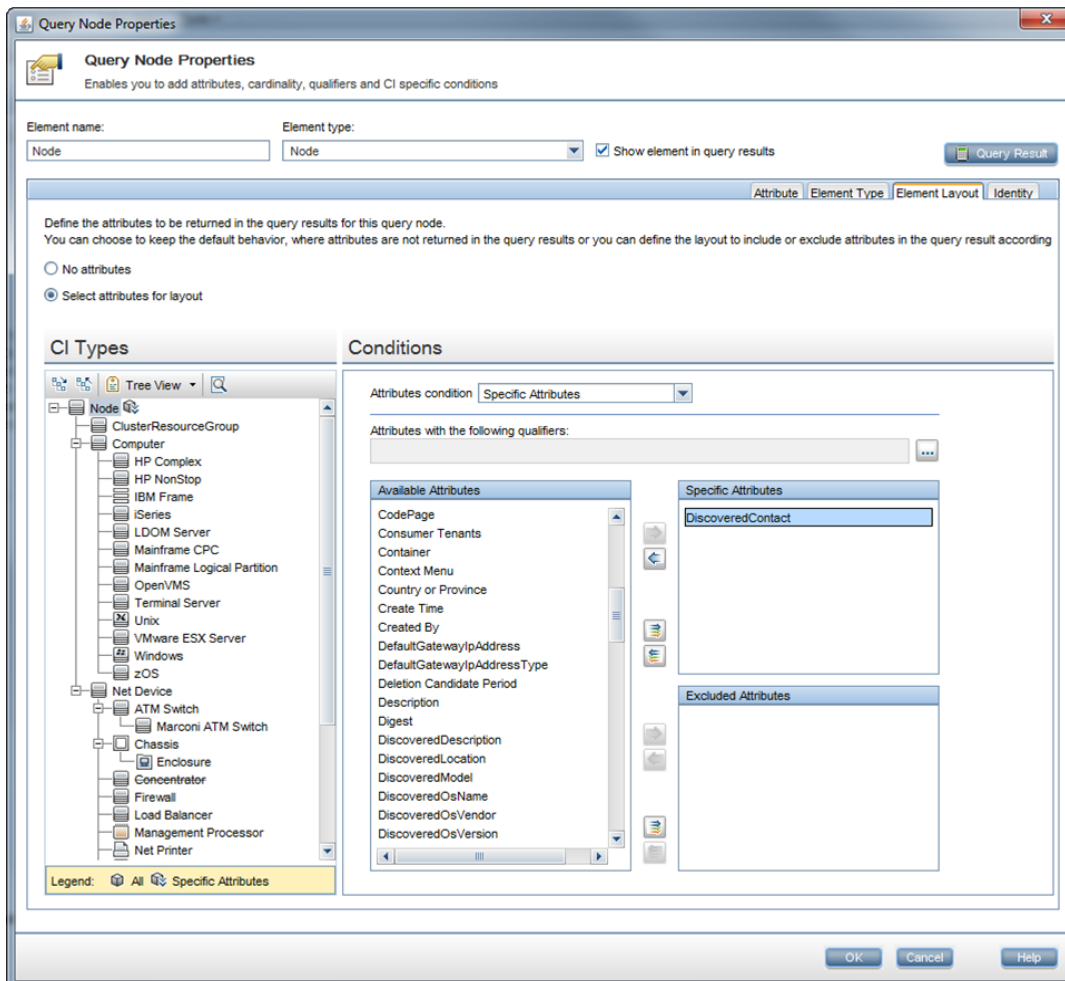
In this example, the OMiOwnerRes query returns CIs of type Node that have a value of NOT null for the attribute DiscoveredContact.

Note: DiscoveredContact is a standard attribute of the Node CIs type. You need to manually add values to the DiscoveredContact attribute for each node CI so that the example returns contact names in the Contact column of the Event Browser.

The Node CI Type conditions are set in the Query Node Properties dialog box as follows:



2. To make the `DiscoveredContact` attribute available to the RTSM JAVA API, you must mark it for calculation under “Element Layout”:



3. Install the script (in this case, with appropriate hostname, credentials, query name, and CI attribute name specified) in the **After CI/ETI Resolution** pipeline step.

Note: No special classpath is required.

4. Exposed the CONTACT CA for use in your browser by configuring the CA in the Available Custom Attributes Operations Management Infrastructure Settings:
 - a. Open Infrastructure Settings:
Administration > Setup and Maintenance > Infrastructure Settings
 - b. Select **Applications** and use the list to set the administration context to **Operations Management**.
 - c. Open the Available Custom Attributes item in the Custom Attributes Setting pane.
 - d. Add CONTACT as a new value.



5. Modify your browser columns to show the new CA (CONTACT).

Title	Related CI	State	CONTACT
Interface error rate exceeds threshold of 2 errors/sec.	11223344556a		
CPU exceeds 95 percent for 10 minutes.	tucana		Worf
Hello from ovlabd2 (CI attribute modified)	ovlabdt2 (OMW Svr)		Jean-Luc Picard
Hello from ovlabd2	ovlabdt2 (OMW Svr)		William Riker
Average query time has exceeded threshold of 10 seconds.	web_order		
Memory utilization has exceeded 90 percent for longer than 5 minutes!	orion		Geordi
SetEventKPIs	SA_7900_2		Data

The `DiscoveredContact` attribute from the RTSM is used to populate the CONTACT column. Some fields in the CONTACT column are empty because the related CI Type is not Node (or child CI Type of Node). As a result, there is nothing in the HashMap because this CI Type is not included in the TQL used to build the HashMap.

Script Example

To enrich events during event processing after the CI has been resolved with CI attributes from the RTSM, the following definition can be used in your EPI Groovy scripts:

```
def ucmdbServiceProvider
UcmdbService ucmdbServiceAccess = null
```

The following OwnerResolver script template illustrates how the definition can be used:

```
import com.hp.ucmdb.api.UcmdbService
import com.hp.ucmdb.api.UcmdbServiceProvider
import com.hp.ucmdb.api.topology.Topology
import com.hp.ucmdb.api.topology.TopologyQueryService
import com.hp.ucmdb.api.types.TopologyCI
import org.apache.commons.logging.Log
import org.apache.commons.logging.LogFactory

class OwnerResolver {
    private static Log s_log = LogFactory.getLog(OwnerResolver.class.canonicalName)

    def ucmdbService
    final String QUERY_NAME = "OMiOwnerRes"

    HashMap<String, String> ownerMap = new HashMap<String, String>()
    boolean running = true
    UcmdbServiceProvider provider = null
```

```
UcldbService service = null
final Object syncObject = new Object()

void init() {
    service = ucldbService.getCmdbService()

    Thread.start {
        while (running) {
            TopologyQueryService tq = service.getTopologyQueryService()
            Topology topology = tq.executeNamedQuery(QUERY_NAME)
            HashMap<String, String> newMap = new HashMap<String, String>()
            topology.getAllCIs().each { TopologyCI ci ->
                final String ciId = ci.id.getAsString()
                final String owner = ci.getPropertyValue("discovered_contact")
                if (owner)
                    newMap.put(ciId, owner)
                s_log.debug("CI with ID=${ciId} owned by ${owner}")
            }
            synchronized (syncObject) {
                ownerMap = newMap
                s_log.info("Owner map initialized with " + ownerMap.size() + " entries.")
                try {
                    syncObject.wait(600000)
                }
                catch (InterruptedException ignore) {
                    // ignore
                }
            }
        }
    }
}

void destroy() {
    ownerMap = null
    running = false
    synchronized (syncObject) {
        syncObject.notifyAll()
    }
}

void process(eventList) {
    synchronized (syncObject) {
        eventList.each { event ->
            def ciId = event.getRelatedCiId()
            if (!ciId) // Check for empty CI Id
                s_log.warn("Related CI ID is NULL or empty")
            else {
                s_log.info("Related CI ID = " + ciId)
                String owner = ownerMap.get(ciId)
                if (owner) {
                    s_log.info("CI Owner = " + owner)
                    event.addCustomAttributes('CONTACT', owner)
                } else {
                    s_log.info("Owner not found for CI with id=" + ciId + ". Owner
map has " + ownerMap.size() + " entries.")
                }
            }
        }
    }
}
}
```

Tips and Limitations

Some tips for using scripts for retrieving RTSM data:

- Use conditions in your query to limit the CIs returned to only those that have some value in the attribute you want to use. This avoids loading CIs with no corresponding attribute value.
- Be mindful of how many CIs the query returns. Do a **Calculate Query Result Count** in RTSM Administration before using the query within the script. The more CIs it returns, the more memory will be required for the HashMap.
- The script name must be the same as the class name specified in the script. The script name is saved under:

Administration > Event Processing > Automation > Event Processing Customizations

- Use the actual CI attribute name in the script, NOT the display name (in the `getPropertyValue` method)
- You can find the time taken by your script for each event from the following log file:

`<OMi_HOME>/log/opr-scripting-host/opr-scripting-host.log`

Creating Scripts

EPI scripts and custom actions scripts share the same script definition format.

For more details about creating scripts, see ["Creating EPI and Custom Action Scripts" on page 134](#).

Chapter 16: Scripts For Custom Actions

This section describes how to configure scripts for custom actions. Custom actions let you define your own actions to apply to events. You can configure Groovy scripts to make custom actions available in the Event Browser.

Note: Custom action scripts can be configured to change only the attributes that can be edited in the Event Browser.

For more information about developing and deploying Groovy scripts, see ["Groovy Scripts" on page 335](#).

- ["Specifying Custom Actions Scripts" below](#)
- ["Creating Scripts" on the next page](#)

Specifying Custom Actions Scripts

Note: To be able to specify or execute custom actions, users must have the appropriate permissions granted in the OMi User Management settings. For details about how to do this, see OMi Help.

The Custom Actions manager enables you to setup scripts to run custom actions on events. As a simple example, you can add a text string to certain events to make them easier to identify in the Event Browser.

You specify custom actions in Groovy scripts. You configure the scripts for custom actions in the following area of the configuration user interface:

Administration > Event Processing > Automation > Event Processing Customizations

Administration > Operations Console > Custom Actions

After a custom action is configured in OMi, the script is available in the list of scripts in the Scripts pane. The script can be triggered from an event from the context menu **custom action list**. The selected custom action is launched in the context of the CI associated with the selected event. If a custom action is run from a non-assigned event, that event is automatically assigned to the user that executed the custom action, and a corresponding entry is made in the Event History.

You can create, copy, edit, and delete scripts for custom actions. You can also stop the execution of a script.

For more details about creating scripts, see ["Creating EPI and Custom Action Scripts" on page 134](#).

For details about how to configure custom actions, see OMi Help.

Custom action scripts can be defined in content packs and can be imported/exported using the Content Manager.

Creating Scripts

EPI scripts and custom actions scripts share the same script definition format.

For more details about creating scripts, see ["Creating EPI and Custom Action Scripts" on page 134](#).

Chapter 17: Creating EPI and Custom Action Scripts

This section describes how to create EPI and custom action scripts.

This section is structured as follows:

- ["Script Definition Attributes" below](#)
- ["Groovy Script API" below](#)
- ["Script Definition Format" on the next page](#)
- ["EPI Groovy Script Samples" on the next page](#)
- ["Custom Actions Groovy Script Samples" on page 142](#)

Script Definition Attributes

A script requires a script definition, and for this you need to specify script definition attributes.

A script definition consists of the following attributes:

- **Name:** The internal script name (**not** the file name of the script).
- **Classpath / JAR files:** One or more JAR files can be uploaded together with the script. The content of the JAR file is available on the classpath during execution of the script. The order of the jar files on the classpath can be changed by moving jar files in the UI up or down.
- **Filter:** *Optional, EPI scripts only.* A filter can be referenced by name. Only events that match the filter are passed to the script.
- **Read-only:** *Optional.* Scripts specified with the read-only attribute do not modify events. These scripts are executed asynchronously to read/write scripts. This asynchronous execution speeds up overall event processing, so it is recommended as a best practice to set the read-only attribute for scripts that are not intended to modify events.
- **Active:** Enables or disables the script for execution. When a script is enabled, the `init()` function is called. Similarly, when a script is disabled, the `destroy()` function is called.
- **Timeout:** *Optional.* The maximum time for each script invocation. This is independent of the number of events. The default value for the timeout is 0, which means that the script execution will never timeout.

For synchronous scripts, if the timeout is reached, script execution is aborted, and all changes made to the events are rolled back.

For asynchronous scripts, if the timeout is reached, script execution is aborted.

Groovy Script API

If you intend to create your own EPI or custom action scripts, you must implement a Groovy script that at a minimum contains calls to the `init()`, `process()`, and `destroy()` functions. The Groovy script

API with full documentation of all arguments and types can be found in the Java API Documentation delivered with the product. For more information about developing and deploying Groovy scripts, see ["Groovy Scripts" on page 335](#).

The Java API Documentation includes the following information needed for creating scripts:

- Event class: this is the main interface
- Complete list of event attributes that are available for modification

You can find the Java API Documentation at the following location:

<OMi_HOME>/opr/api/doc/opr-external-api-javadoc.zip

Script Definition Format

The basic format of a script definition for EPI scripts and custom action scripts looks like this:

```
import com.hp.opr.api.scripting.Event;

def init()
{
    // This method is called when the script is loaded (for example,
    // when OMi is started, when the script is enabled, or when the
    // script has been modified).
}

def destroy()
{
    // This method is called when the script is unloaded (for example,
    // if it is disabled in the configuration user interface).
}

def process(List<EpiEvent> events)
{
    // This method is called when events are processed. The list is of
    // type java.util.List.

    // In this method, properties of the events can be changed. If the
    // script is in read-only mode, the UnsupportedOperationException is
    // thrown.

    // See the Java API documentation for opr-external-api.jar for a
    // list of event attributes that are available for modification.
}
```

EPI Groovy Script Samples

This section contains the sample EPI Groovy scripts shipped with the product.

You can find the EPI Groovy script examples in the following directory:

<OMi_HOME>/opr/examples/epi-scripts

SimpleExampleEPI.groovy

This is a simple example that sets all possible event attributes to some sample values.

```
import java.util.Date;
import java.util.List;

import com.hp.opr.api.scripting.Action;
import com.hp.opr.api.scripting.Event;
import com.hp.opr.api.scripting.EventActionFlag;
import com.hp.opr.api.scripting.LifecycleState;
import com.hp.opr.api.scripting.MatchInfo;
import com.hp.opr.api.scripting.NodeInfo;
import com.hp.opr.api.scripting.PolicyType;
import com.hp.opr.api.scripting.Priority;
import com.hp.opr.api.scripting.ResolutionHints;
import com.hp.opr.api.scripting.Severity;

/*
 * This example set all possible event attribute to some example values.
 */

class SimpleExample
{
    def init()
    {
    }

    def destroy()
    {
    }

    def process(List<Event> events)
    {
        events.each {
            event -> modifyEvent(event);
        }
    }

    def modifyEvent(Event event)
    {
        String application = event.getApplication();
        event.setApplication("Modified by EPI: " + application);

        def groupId = event.getAssignedGroupId();
        event.setAssignedGroupId(groupId);

        def assignedUserId = event.getAssignedUserId();
```



```
event.setAssignedUserId(assignedUserId);

Action autoAction = createSampleAction();
event.setAutoAction(autoAction);

String assignedGroupName = event.getAssignedGroupName();
event.addCustomAttributes("ASSIGNED_GROUP_NAME", assignedGroupName);

String assignedUserLogin = event.getAssignedUserLogin();
event.addCustomAttributes("ASSIGNED_USER_LOGIN", assignedUserLogin);

String category = event.getCategory();
event.setCategory("Modified by EPI: " + category);

String correlationKeyPattern = event.getCloseKeyPattern();
event.setCloseKeyPattern("Modified by EPI: " + correlationKeyPattern);

String description = event.getDescription();
event.setDescription("Modified by EPI: " + description);

String etiDisplayName = event.getEtiDisplayName();
event.addCustomAttributes("ETI_DISPLAY_NAME", etiDisplayName);

String etiName = event.getEtiName ();
event.addCustomAttributes ("ETI_NAME", etiName);

String etiStateDisplayName = event.getEtiStateDisplayName ();
event.addCustomAttributes("ETI_STATE_DISPLAY_NAME", etiStateDisplayName);

String etiStateName = event.getEtiStateName();
event.addCustomAttributes("ETI_STATE_NAME", etiStateName);

String etiInfo = event.getEtiHint();
event.setEtiHint(etiInfo);

String correlationKey = event.getKey();
event.setKey("Modified by EPI: " + correlationKey);

MatchInfo matchInfo = createSampleMatchInfo();
event.setMatchInfo(matchInfo);

event.setNoDedup(true);

ResolutionHints hints = createSampleResolutionHints();

event.setNodeHints(hints);

String object = event.getObject();
event.setObject("Modified by EPI: " + object);
```

```
String omServiceId = event.getOmServiceId();
event.setOmServiceId(omServiceId);

String omUser = event.getOmUser();
event.setOmUser(omUser);

String originalText = event.getOriginalData();
event.setOriginalData("Modified by EPI: " + originalText);

String originalId = event.getOriginalId();
event.setOriginalId(originalId);

event.setPriority(Priority.HIGHEST);

String ciInfo = event.getRelatedCiHint();
event.setRelatedCiHint("Modified by EPI: " + ciInfo);

event.setSeverity(Severity.CRITICAL);

String solution = event.getSolution();
event.setSolution("Modified by EPI: " + solution);

ResolutionHints sourceCiHints = createSampleResolutionHints();
event.setSourceCiHints(sourceCiHints);

event.setState(LifecycleState.IN_PROGRESS);

String subCategory = event.getSubCategory();
event.setSubCategory("Modified by EPI: " + subCategory);

event.setTimeReceived(new Date());

String title = event.getTitle();
event.setTitle("Modified by EPI: " + title);

String type = event.getType();
event.setType("Modified by EPI: " + type);

Action userAction = createSampleAction();
event.setUserAction(userAction);

/*
 * Calling isReceivedOnCiDowntime should be done after CI/ETI resolution,
 * at which point the CI is identified.
 */

Boolean receivedDuringDowntime = isReceivedOnCiDowntime();
event.addCustomAttributes("RECEIVED_ON_DOWNTIME", receivedDuringDowntime);
```

```
}

def ResolutionHints createSampleResolutionHints()
{
    ResolutionHints hints = new ResolutionHints(false);

    hints.setCoreId("CoreId");
    hints.setDnsName("mydqn.com");
    hints.setHint("My Hint");
    hints.setIpAddress("0.0.0.0");
    return hints;
}

def MatchInfo createSampleMatchInfo()
{
    MatchInfo matchInfo = new MatchInfo(false);

    matchInfo.setConditionId("conditionId");
    matchInfo.setPolicyName("policyName");
    matchInfo.setPolicyType(PolicyType.CONSOLE);
    return matchInfo;
}

def Action createSampleAction()
{
    NodeInfo actionNodeInfo = new NodeInfo(false);

    Action action = new Action(false);
    actionNodeInfo.setCoreId("CoreId");
    actionNodeInfo.setDnsName("myfqdn.com");
    actionNodeInfo.setIpAddress("0.0.0.0");

    action.setCall("Call");
    action.setNode(actionNodeInfo);
    action.setStatus(EventActionFlag.AVAILABLE);
    return action;
}
}
```

RegExample.groovy

This example script flips every second word with its previous word.

```
import java.util.Date;
import java.util.List;

import com.hp.opr.api.scripting.Action;
import com.hp.opr.api.scripting.Event;
```

```
import com.hp.opr.api.scripting.EventActionFlag;
import com.hp.opr.api.scripting.LifecycleState;
import com.hp.opr.api.scripting.MatchInfo;
import com.hp.opr.api.scripting.NodeInfo;
import com.hp.opr.api.scripting.PolicyType;
import com.hp.opr.api.scripting.Priority;
import com.hp.opr.api.scripting.ResolutionHints;
import com.hp.opr.api.scripting.Severity;

/*
 * This script flips every second word with its previous word.
 */
class RegExpExample
{
    def init()
    {
    }

    def destroy()
    {
    }

    def process(List<Event> events)
    {
        events.each {
            event -> event.setTitle(event.getTitle().replaceAll(/(\w+)\s+(\w+)/, '$2 $1'))
        }
    }
}
```

ResolveLocationFromDB.groovy

This script matches an IP address to a node name to resolve a location from the database.

To make this example work with a MS SQL Server, you must do the following:

1. Create a new DB asset_db (or adjust the name below).
2. Create a new table LocationMapping (or adjust the name below) with the following attributes:
 - ip (varchar(50), primary key)
 - location (varchar(50))
 - phone (varchar(50))
 - contact (varchar(50))
3. Add rows to the table for each IP address that matches the node hint of your events.
4. Adjust the database parameters in `Sql.newInstance` below.

5. Add this script as an EPI script. Upload a JTDS driver. You can find a JTDS driver at <http://jtds.sourceforge.net>.

```
import groovy.sql.Sql;

class ResolveLocationFromDB
{
    def connection;
    void init()
    {
        def properties = new Properties();

        // Create Properties object
        def properties = new Properties();
        // Set user ID for connection
        properties.put("user", "sa");
        // Set password for connection
        properties.put("password", "installed");

        def driver = new net.sourceforge.jtds.jdbc.Driver ();
        def conn = driver.connect("jdbc:jtds:sqlserver://localhost/asset_db",
properties);

        connection = Sql.newInstance(conn);
    }

    void process(eventList)
    {
        try {
            for (event in eventList) {
                def nodeHints = event.getNodeHints();
                def nodeName = nodeHints.getDnsName();
                if(nodeName != null) {
                    def ipaddress = InetAddress.getByName(nodeName).hostAddress
                    connection.eachRow('select * from LocationMapping', {
                        if (it.ip == ipaddress) {
                            if (event.getDescription() == null)
                                event.setDescription("CI located in building: " +
                                    it.location)
                            else
                                event.setDescription(event.getDescription() +
                                    "\nCI located in building: " +
                                    it.location)
                            event.addCustomAttribute('phone', it.phone)
                            event.addCustomAttribute('contact', it.contact)
                            event.addCustomAttribute('location', it.location)
                        }
                    });
                }
            }
        }
    }
}
```

```
    }  
  
    finally {  
    }  
}  
  
void destroy()  
{  
    connection.close();  
}  
}
```

Custom Actions Groovy Script Samples

This section contains a Groovy script example for custom actions.

You can find the custom actions Groovy script example in the following directory:

<OMI_HOME>/opr/examples/ca_scripts

SimpleExample.groovy

Here is a simple example of a custom actions script that modifies an event:

```
import com.hp.opr.api.scripting.Event;  
import com.hp.opr.api.scripting.Priority;  
import com.hp.opr.api.scripting.Severity;  
  
class SimpleExample  
{  
    def init()  
    {  
        // Nothing to initialize  
    }  
    def destroy()  
    {  
        // Nothing to destroy  
    }  
    def process(List<Event> events)  
    {  
        events.each {  
            event -> modifyEvent(event);  
        }  
    }  
    def modifyEvent(Event event)  
    {  
        event.addCustomAttributes("CA_SCRIPT", "MODIFIED");  
        event.setSeverity(Severity.CRITICAL);  
        event.setPriority Priority.HIGHEST;  
    }  
}
```

```
}  
  }
```

Chapter 18: EPI Troubleshooting

This section contains information to help you troubleshoot EPI and custom action script execution.

Log Files

A good starting point for troubleshooting EPI and custom action script execution is to look at the following log file:

```
<OMi_HOME>/log/opr-scripting-host/opr-scripting-host.log
```

The log file can provide helpful information about script execution. For example, you can see:

- When a script was loaded or shut down.
- When a script execution reached its timeout.
- Statistics about the execution of a script, for example, the time taken for script execution.

The following file contains log entries generated by the scripts themselves, for example runtime errors:

```
<OMi_HOME>/log/opr-scripting-host/scripts.log
```

Debugging

To perform a debug, do the following:

1. Go to the following location:

```
<OMi_HOME>/conf/core/Tools/log4j/opr-scripting-host/opr-scripting-host.properties
```
2. In the `opr-scripting-host.properties` file, set the `loglevel` to the desired value.
The debug `loglevel` (`loglevel=DEBUG`) is useful for finding problems.

Part V: Integrating the OMi UI with Other Applications

This section describes how to integrate parts of the OMi user interface with an external application using a drill-down URL launch.

This section is structured as follows:

- ["Specifying a URL Launch" on page 146](#)
- ["Parameters and Parameter Values" on page 147](#)
- ["Defining Columns" on page 149](#)
- ["Setting Filters" on page 151](#)
- ["URL launch of the Event Details" on page 156](#)

Chapter 19: URL Launch of the Event Browser

You can launch the Event Browser using a URL link. This is particularly interesting for integrators who want to integrate parts of the OMi user interface (UI) with an external application. So an operator of an external application with a graphical user interface can drill down into the OMi UI. For example, it is possible to have a portal application, where operators can launch the Event Browser and the Event Details in a browser within their application.

This section describes how to specify such a URL launch and includes the following sections.

- ["Specifying a URL Launch" below](#)
- ["Parameters and Parameter Values" on the next page](#)
- ["Defining Columns" on page 149](#)
- ["Setting Filters" on page 151](#)
- ["URL launch of the Event Details" on page 156](#)

Specifying a URL Launch

To perform a URL launch of the Event Browser, you can either use the default URL, or specify additional parameters. The optional parameters you specify in the URL enable you to define how you want the Event Browser to appear and behave.

Default URL Launch

When you use the default URL launch, the Event Browser opens with default OMi UI settings.

Note: In the Event Browser launched with the default URL, any changes you make (for instance, visible columns and columns widths) are *not* automatically saved.

To launch the Event Browser with default UI settings, enter the URL as follows:

```
https://<hostname:port>/opr-console/opr-evt-browser
```

Note: You must specify the port number if the port you are using is not the default HTTP port (80). If you are using the default port, you need not specify the port number.

Specifying Optional Parameters

You can specify additional parameters to define what you want to display in the Event Browser.

The available parameters, together with their possible values, are given in ["Parameters for the URL Launch of the Event Browser" on the next page](#).

You specify additional parameters for the URL launch in the following way:

```
https://<hostname:port>/opr-console/opr-evt-browser?<set_parameters_here>
```

Note: You must specify the port number if the port you are using is not the default HTTP port (80). If you are using the default port, you need not specify the port number.

Note: You place the character “?” in front of the first argument of the URL, and thereafter separate each argument by the “&” character.

Here is an example of a URL launch with optional parameters:

```
https://my.example.com:8080/opr-console/opr-evt-  
browser?sortField=severity&sortOrder=desc&filter_severities=critical,major,minor
```

Parameters and Parameter Values

"Parameters for the URL Launch of the Event Browser" below contains the parameters available for the URL launch of the Event Browser:

Parameters for the URL Launch of the Event Browser

Parameter	Description	Possible Values	Default Values
activeUpdates	Specifies whether the Event Browser gets updates periodically from the server. Without periodic updates, be aware that you may not get all data into the browser.	One of: <ul style="list-style-type: none"> • true • false 	true
checkParams	<p>Determines whether unknown URL launch parameters are validated.</p> <p>If you do not specify the checkParams parameter, unknown parameters (for example, misspelled parameters) are not validated; the Event Browser ignores unknown parameters and opens without error notifications. Known parameters are always validated.</p> <p>If you set the checkParams parameter to true, the Event Browser validates all parameters and reports unknown parameters as errors.</p>	One of: <ul style="list-style-type: none"> • true • false 	false
columns	Defines the columns to be displayed in the Event Browser.	One or more of: <Column key> (see "Fixed Columns Keys to Define Which Columns are Displayed" on page 149)	Default set of columns

Parameters for the URL Launch of the Event Browser, continued

Parameter	Description	Possible Values	Default Values
context	<p>Defines the context of the browser settings. Editing browser settings (for example, visible columns or column widths) when a context is selected, means that the settings are saved under the given context. The next time you open the browser with the same context, the browser settings are restored.</p> <p>If you do not specify the context parameter, the browser settings are not saved.</p> <p>Example: context=myOwnContext</p>	Any user-defined string	No default value set
headerText	Sets a header text for the Event Browser.	Any user-defined string	Event Browser
include_child_cis	Launches the Event Browser with events that are related to the given event's related CI and any of its child CIs.	One of: <ul style="list-style-type: none"> • true • false 	false
readOnly	<p>Launches the Event Browser in read-only mode.</p> <p>In read-only mode, most icons, buttons, and menus are hidden to prevent the user from performing actions on an event. Users can show and hide event details, refresh the browser, search events, and open OMi Help.</p>	One of: <ul style="list-style-type: none"> • true • false 	false
showClosed	Launches the Closed Events Browser configuration when the system starts.	One of: <ul style="list-style-type: none"> • true • false 	false
showDetails	Launches the Event Browser with embedded event details already open.	One of: <ul style="list-style-type: none"> • true • false 	false

Parameters for the URL Launch of the Event Browser, continued

Parameter	Description	Possible Values	Default Values
sortField	Defines the column by which the Event Browser is sorted.	One of: <Column key> (see "Fixed Columns Keys to Define Which Columns are Displayed" below)	timeReceived
sortOrder	Defines the sort order of the column specified by the sortField parameter, in either ascending or descending order.	One of: <ul style="list-style-type: none"> asc desc 	desc
symbol_id	The symbolic CI ID in a view. The symbolic ID is made up of the view root ID 1 followed by the IDs of the given CI's parent CIs. The IDs must be separated with two semicolons (; ;). Example: 1;;decafc0ffefacedbabef00ddeadbeef;;0019bbcaedc44a2eea884a4110a4c38a In this example, 1 is the ID of the view root, decafc0ffefacedbabef00ddeadbeef is the ID of the parent CI, 0019bbcaedc44a2eea884a4110a4c38a the ID of the given CI.	Symbolic CI ID representing the CI in a view	No default value set

Defining Columns

The fixed column keys for available columns are defined in ["Fixed Columns Keys to Define Which Columns are Displayed"](#) below. ["Custom Attribute Column Key"](#) on page 151 contains a special kind of column key, that unlike the other column keys, is customer configurable. All the column keys are case-insensitive, which means that for example "ID", "Id" and "id" all map to the ID column.

For more detailed explanations about the column keys, see OMi Help.

Fixed Columns Keys to Define Which Columns are Displayed

Column Key	Explanation
annotations	Annotations
application	Application

Fixed Columns Keys to Define Which Columns are Displayed, continued

Column Key	Explanation
automaticAction	Automatic action
category	Category
ciType	CI type
controlTransferred	Shows whether control of the event has been transferred
correlation	Correlation (symptom or cause)
description	Description
duplicateCount	Number of duplicates
eventAge	Age of the event based on time created
eti	Event type indicator value
externalId	External event ID
group	Assigned group
id	ID
node	Node
nodeHint	Node hint
object	Object
operatorAction	Operator action
originatingServer	Originating server
ownedInOM	Owned in OM.
priority	Priority
receivedOnCiDowntime	Received during CI downtime
relatedCi	Related CI
relatedCiHint	Related CI Hint
sendingServer	Sending server
severity	Severity
solution	Solution
sourceCi	Source CI
sourceCiHint	Source CI hint

Fixed Columns Keys to Define Which Columns are Displayed, continued

Column Key	Explanation
state	Lifecycle state
subCategory	Subcategory
timeCreated	Time when the event was created
timeFirstReceived	Time when the event was first received
timeReceived	Time when the (last duplicate) event was received
timeStateChanged	Time when lifecycle state of the event was last changed
title	Title
type	Type
user	Assigned user

Custom Attribute Column Key

Column Key	Explanation
ca_ <columnname>	Custom attribute column, where <columnname> is the name of the custom attribute. This is a configurable column key that you configure in the Available Custom Attributes setting in Custom Attribute Settings. For details, see OMi Help.

Setting Filters

You can narrow down the number of events in the Event Browser by using filters. You can specify filters for the following:

- String attributes
- Time properties
- Flag attributes
- Event priorities

"[Filter Parameters for the URL Launch of the Event Browser](#)" on the next page contains the filter parameters available for the URL launch of the Event Browser.

Filter Parameters for the URL Launch of the Event Browser

Filter Parameter	Description	Values
filter_assignment	Applies an assignment filter to the Event Browser.	One or more of: <ul style="list-style-type: none"> • me • my_workgroups • others • nobody
filter_severities	Applies a severity filter to the Event Browser.	One or more of: <ul style="list-style-type: none"> • unknown • normal • warning • minor • major • critical
filter_states	Applies a lifecycle state filter to the Event Browser.	One or more of: <ul style="list-style-type: none"> • open • in_progress, • resolved

Filtering by String Attributes

You can filter events by string attributes.

You specify filters to filter by properties of type string using the following format:

filter_<stringAttributeName>_<filterType>

Possible values for string attribute names and filter types are listed in "[Possible Filter Types and Values for String Attributes](#)" below.

Possible Filter Types and Values for String Attributes

Possible string attribute names	application
	ca_<customAttribute>
	category
	correlationKey
	description
	object

Possible Filter Types and Values for String Attributes, continued

	originalText
	relatedCiHint
	subCategory
	title
	type
Possible filter types	contains
	equals
	isEmpty
	isNotEmpty
	notContains
	notEquals

Here is an example of filtering by string attributes, where we are interested only in returning events that have a filter category called Network:

```
filter_category_equals=Network
```

Filtering by Time Properties

You can also filter events by time properties.

You can set the filter so that the Event Browser displays only those events for which the time attribute `timeAttributeName` is between `fromTime` and `toTime`.

You specify filters to filter by time using the following format:

```
filter_<timeAttributeName>=>fromTime-toTime
```

where:

<timeAttributeName> can be one of: `timeCreated`, `timeFirstReceived`, `timeReceived`, `timeStateChanged`.

You must specify the time in the following format:

```
<yyyyMMddHHmmss>
```

where:

- yyyy is the year
- MM is the month value (01-12)
- dd is the day value (01-31)
- HH is the hour value (00-23)

- mm is the minute values (00-59)
- ss is the second value (00-59)

Filtering by Priorities

You can also filter events by priorities.

You can set the filter so that Event Browser displays the events according to their priorities.

You specify filters to filter by priorities using the following format:

```
filter_priorities=<priority_level>
```

where:

<priority_level> can be one or more of the following in a comma-separated list: none, lowest, low, medium, high, highest.

As an example, if you want the Event Browser is to display only those events with the priorities highest and high, then you would specify the filter as follows:

```
filter_priorities=highest,high
```

Filtering by CIs and CI Types

You can set a filter so that the Event Browser displays only the events that are related to a given CI. To specify such a filter, use the following format:

```
filter_relatedCi_equals=<CI Id>
```

where:

<CI Id> is the ID of the CI.

Possible filter operations are: equals, isempty, notisempty

You can also set a filter so that the Event Browser displays only events where the CI type of the related CI matches the CI type specified. To specify such a filter, use the following format:

```
filter_ciType_<operator>=<type>
```

where <type> is the specified CI type, and <operator> can have one of the following values: equals, is_derived.

Filtering by Global CI ID

You can filter events by global CI ID. The global CI ID is the global ID of a CI in an external (non-OMi) database, such as a Content Management System (CMS) database.

When specifying the globalCiId parameter, the local (ODB) CI ID is looked up and a filter for that CI ID is set up.

To specify such a filter, use the following format:

```
filter_globalCiId_equals=<global id>
```

where:

<global id> is the global ID of the CI.

Possible filter operations are: equals, isempty, notisempty

The filter specification `filter_globalCiId_equals=<global id>` returns the same result as the filter specification `filter_relatedCi_equals=<CI Id>`.

Filtering by ETIs and ETI Values

You can set a filter so that the Event Browser displays only the events that match a given ETI.

To specify a filter by ETI, use the following format:

```
filter_eti_equals=<ETI Id>
```

where:

<ETI Id> is the UUID of the ETI.

Possible filter operations are: equals, isempty, notisempty, isoneof.

When using the `isoneof` filter operation, a comma-separated list of ETI UUIDs is expected, for example:

```
filter_eti_isoneof=<ETI Id1,ETI Id2,ETI Id3,...>
```

For the filter operations `isempty` and `notisempty`, no argument is expected.

You can also set a filter so that the Event Browser displays only events that set a specific ETI Value.

To specify such a filter, use the following format:

```
filter_etiValue_<operator>=<ETI value Id>
```

where:

<ETI value Id> is the UUID of the ETI Value.

Possible filter operations are: equals, isempty, notisempty, isoneof.

When using the `isoneof` filter operation, a comma-separated list of ETI value UUIDs, is expected, for example:

```
filter_etiValue_isoneof=<ETI value Id1,ETI value Id2,ETI value Id3,...>
```

For the filter operations `isempty` and `notisempty`, no argument is expected.

Filtering by Other Event Characteristics

You can also set a filter to group events that share the same characteristic. For example, you may want to see only those events that have duplicates.

To filter events to group them by event characteristics, there are a number of boolean flag attributes that you can specify. These boolean flag attributes specify whether the event possesses a particular characteristic, for example, whether it has symptoms, or annotations. You can set the filter so that the Event Browser displays only those events that match the boolean flag attribute for the specified characteristic.

To set a filter to display events that share a particular characteristic, you specify the boolean flag attribute for that characteristic using the following format:

`filter_<flagAttributeName>`

where:

`<flagAttributeName>` can have one of the following values: `hasSymptoms`, `hasCause`, `hasDuplicates`, `hasAnnotations`.

You can also specify a combination of characteristics, as in the following example where the filter is set to display only those events that have both duplicates and symptoms:

```
https://<my.example.com:8080>/opr-console/opr-evt-browser?filter_  
hasDuplicates&filter_hasSymptoms
```

URL launch of the Event Details

In a similar way to launching the Event Browser using a URL, you can also launch the Event Details using the following URL:

```
http://<hostname:port>/opr-console/opr-evt-details?eventId=<id_of_the_event>
```

where:

`<id_of_the_event>` is the ID of the event that you want to display in the Event Details.

Here is an example of the URL for a direct launch of the Event Details, showing the parameter `eventId` set to the ID of the event that you want to display:

```
http://my.example.com:8080/opr-console/opr-evt-details?eventId=e004e66b-cada-407f-  
84ac-32f2d613eec4
```

Part VI: Automating Operator Functions and Event Change Detection

This section provides integrators with information to allow them to programmatically automate operator functions and detect event changes. Most operations that an operator can do in the console while working on events can be done programmatically, to improve efficiency.

For information about integrating applications for event synchronization, see ["Forwarding Events and Synchronizing Event Changes" on page 203](#).

This section is structured as follows:

- ["Event Web Service Interface Reference" on page 158](#)
- ["Event Web Service Query Language" on page 171](#)
- ["RestWsUtil Command-Line Interface" on page 190](#)

Chapter 20: Event Web Service Interface

Reference

An interface is provided for integrators to integrate events into other applications, enabling them to programmatically automate operator functions and detect event changes. Most operations that an operator can do in the console while working on events can be done programmatically, to improve efficiency, and enable integration with external applications.

The interface used for integrating events into other applications, and automating operator functions, is the Event Web Service. This is a REST-based web service supporting the event model, that also provides subscription support through Atom feed functionality. You can read an Atom feed in your browser, where you can see a list of events, and you can also create and update events using the Atom service.

For convenience, the link to the URL for launching the Event Browser (for drilling down into the OMi user interface from an external application) is included in the event web service data. For details of how to specify and launch the drill-down URL, see ["URL Launch of the Event Browser" on page 146](#).

An integrator would typically be interested in:

- ["How to Access the Event Web Service" below](#)
- ["How to Detect New Events" on page 160](#)
- ["How to Detect Event Changes" on page 162](#)
- ["How to Modify Events" on page 162](#)
- ["How to Create New Events" on page 167](#)

For general information about using web service interfaces such as the Event Web Service Interface with OMi, see ["Reference Information For All Web Services" on page 248](#).

How to Access the Event Web Service

Your entry point to the event web service interface is the Service Document, using the following base URL:

- Secure environments:
`https://<server.example.com>/opr-web/rest/`
- Non-TLS environments:
`http://<server.example.com>/opr-web/rest/`

where:

`<server.example.com>` is the name of the gateway server.

For access to the Event Web Service, you need to be a valid user, and need to provide your user name and password as credentials for user authentication. Only authorized users can view events, change events, or run actions.

The Service Document lists the URLs of different OPR Event services. These services are listed in "[List of OPR Event Services in the Service Document](#)" below.

In the list of URLs in the Service Document, only two of them are actual links:

- Events Service:

`https://<server.example.com>/opr-web/rest/9.10/event_list`

9.10 is the current version of the OMi Event Web Service (not the current OMi version). If the version number is omitted, versions lower than 9.10 of the Event Web Service are addressed.

This shows a list of all events.

- Event Changes Service:

`https://<server.example.com>/opr-web/rest/event_change_list`

This shows a list of changes to events.

All the other URLs require a parameter to be specified (for instance, the event ID). In case of annotations, for the event for which you want the annotation, you need to specify the annotation ID. In "[List of OPR Event Services in the Service Document](#)" below, variables that you need to specify for each URL are shown in curly brackets, such as {event}, {annotation}, or {custom_attribute}. An example URL specifying the event with the ID 532d3674-684f-419f-a752-b8681ee01a72 would look like this:

`https://<server.example.com>/opr-web/rest/9.10/event_list/532d3674-684f-419f-a752-b8681ee01a72`

List of OPR Event Services in the Service Document

OPR Event Service	URL
Annotation Service	<code>https://<server.example.com>/opr-web/rest/9.10/event_list/{event}/annotation_list/{annotation}</code>
Annotations Service	<code>https://<server.example.com>/opr-web/rest/9.10/event_list/{event}/annotation_list</code>
Automatic Action Service	<code>https://<server.example.com>/opr-web/rest/9.10/event_list/{event}/auto_action</code>
Custom Attribute Service	<code>https://<server.example.com>/opr-web/rest/9.10/event_list/{event}/custom_attribute_list/{custom_attribute}</code>
Custom Attributes Service	<code>http://<server.example.com>/opr-web/rest/9.10/event_list/{event}/custom_attribute_list</code>
Event Change Service	<code>https://<server.example.com>/opr-web/rest/event_change_list/{event_change}</code>
Event Changes Service	<code>https://<server.example.com>/opr-web/rest/event_change_list</code>

OPR Event Service	URL
Event Service	https://<server.example.com>/opr-web/rest/9.10/event_list/{event}
Events Service	https://<server.example.com>/opr-web/rest/9.10/event_list
History Line Service	https://<server.example.com>/opr-web/rest/9.10/event_list/{event}/history_line_list/{history_line}
History Lines Service	https://<server.example.com>/opr-web/rest/9.10/event_list/{event}/history_line_list
Operator Action Service	https://<server.example.com>/opr-web/rest/9.10/event_list/{event}/user_action
Symptom Service	https://<server.example.com>/opr-web/rest/9.10/event_list/{event}/symptom_list/{symptom}
Symptoms Service	https://<server.example.com>/opr-web/rest/9.10/event_list/{event}/symptom_list

In general, for most of the OPR Event services, you can carry out the four supported operations (read, create, update and delete). Exceptions to this are:

- Automatic Action Service and Operator Action Service. For these services you can:
 - Get the state of an action by performing a read operation.
 - Start an action (if it is not already running) by performing a create operation.
 - Stop an action (if it is already in progress) by performing a delete operation.
- Services where only read operations are possible:
 - Event Change Service
 - Event Changes Service
 - History Line Service
 - History Lines Service

At first glance, the two services Event Change and Event Changes return very similar results to the services History Line and History Lines. However, the former services list all event changes, and are not specific to any particular events. The latter services are event-specific, and list changes to events relating to particular events. For more details about History Lines, see ["History Lines" on page 187](#).

How to Detect New Events

To return a list of all events, in the Service Document, click the URL for the event list:

https://<server.example.com>/opr-web/rest/9.10/event_list

By default, clicking this URL opens the event list in XML format. You can add HTTP query parameters to this base URL to change the way you want to view the data in the list:

- `format=atom`: This parameter presents the list of events in an Atom feed format. How the event data is displayed is determined by meta-data (for example, categories, author, and so on). For further details about the `format` media type, see ["Media Type" on page 178](#).
- `format=json`: This parameter presents the list of events in JSON format.
- `format=xml`: This parameter presents the list of events in XML format. This is the default for a web browser.

Receiving Events as Atom Feeds

Most browsers come with a very basic internal feed reader for Atom and RSS feeds. As the feed readers vary in their features and in the way they represent data, the same data may be represented differently by different readers (and you would typically also access the data in a different way, too).

The two most commonly available browsers with feed readers are Mozilla Firefox and Microsoft Internet Explorer. The following examples show how the Atom web service works with these browsers.

Event lists in an Atom feed are ordered as “last changed”, so a new event, or an event that was modified last would be at the top of the list.

To return a list of events in an atom feed format, follow these steps:

1. Enter the URL for the event list in your web browser, specifying the `format=atom` parameter as follows:
`https://<server.example.com>/opr-web/rest/9.10/event_list/?format=atom`
2. You now see a list of events. Firefox and Internet Explorer represent the data in a slightly different way.

- **In Firefox:**

You see a list of events where the title and description of each event are shown.

If you click on the title of an event in the atom feed, you start the URL launch of the Event Details for that event. For more information about the URL launch of the Event Browser and Event Details, see ["URL Launch of the Event Browser" on page 146](#).

Note: If you want to see a more detailed list of the events, right-click the page and select **View Page Source**. This displays the whole atom feed in XML format. Now you can see all other properties and information regarding the events that are listed.

- **In Internet Explorer:**

You see a list of events where the title and description of each event are shown. In addition to the list, you see a filter box on the right, where you can sort the list by date and title. You can also filter the events by specific categories.

If you click on the title of an event in the atom feed, you start the URL launch of the Event Details to drill down into the OMi user interface. For more information about the URL launch of the Event Browser and Event Details, see ["URL Launch of the Event Browser" on page 146](#).

Only basic details about the events are shown, and to see all the details, right-click on the page and select **View Source**. This opens the page in your XML editor.

Specifying Parameters to Filter the Event List

The Event Web Service provides a number of parameters for filtering the event list by specific criteria. For details about the URL query language, query filter, and parameters, see ["HTTP Query Parameters" on page 171](#).

How to Detect Event Changes

To return a list of event changes, in the Service Document, click the URL for the event change list:

```
https://<server.example.com>/opr-web/rest/event_change_list
```

You may want to return a list of all event changes since the last time you got a list of events. The following example URL returns a list of all event changes since 12:29:54 on March 10, 2010 in an atom feed format:

```
https://<server.example.com>/opr-web/rest/event_change_list?format=atom&watermark=2010-03-10T15:59:17%2B01:00
```

How to Modify Events

You can modify events (read, create, update and delete items) using a REST client within your web browser, or using the `RestWsUtil` command-line utility.

Note: With OMi 9.10 and higher, Event Web Service modify operations must be secured by setting the `X-Secure-Modify-Token` HTTP header on modify requests (PUT, POST, and DELETE). This header provides enhanced security protection against malicious exploits of web applications. For more information, see ["Securing Modify Operations" on page 250](#).

Modifying Events Using a REST Client

In general, when using a REST client to update an event, it is usual to do the following:

- Call the REST service by entering the URL of the event in question (you need to specify the event ID in the URL).
- Retrieve the event with an HTTP GET request.
- Edit the elements of the XML document you want to change.
- Send back the changed parts of the event using an HTTP PUT request.
- Reload the XML to see the changes.

Two examples of REST clients that can be used to modify events are the `RESTClient` and the Mozilla Firefox Poster Extension.

Modifying Events Using RESTClient

Here, we will look at how to use the `RESTClient` to modify events. The `RESTClient` is available as an

open source download. You can find information about where to download the RESTClient from this location:

<http://code.google.com/p/rest-client/>

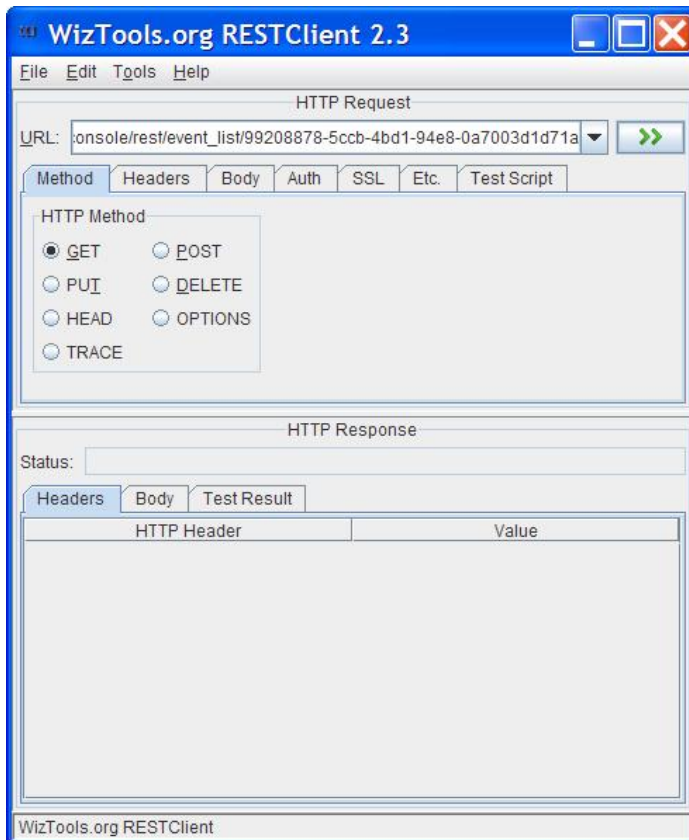
The RESTClient user interface is divided into two parts:


- HTTP Request: top half of the user interface, used for entering the URL for the event you want to modify, to retrieve the event with an HTTP GET request, and for sending the changed XML using an HTTP PUT request.
- HTTP Response: bottom half of the user interface, used for returning the response.


We will use an example of how to change the title and severity of an event using the RESTClient. The steps required are as follows:

1. Get the event ID of the event you want to modify from the event list feed.
2. In the URL field of the RESTClient user interface, enter the URL (specifying the event ID) of the event you want to modify. The syntax is as follows:

`https://<server.example.com>/opr-web/rest/9.10/event_list/<event_ID>`



3. Select the **GET** radio button in the HTTP Method box, and click the  button.
4. The result is returned in the lower, HTTP Response section. Click the **Body** tab to read the response XML for the event you want to edit.
5. Copy the XML. Click the **Body** tab in the upper, HTTP Request section, and paste the XML into the text box.

6. Edit the event properties you want to change to modify the event. It is not necessary to send the complete XML back to the server. As long as you preserve the XML structure, you can choose to send only those XML elements that you want to update for that event.
7. When you are done with your changes, select the **Method** tab again, and then click the **PUT** radio button.
8. Click the  button to submit the changes. When the changes have been applied, you will see an HTTP 200 OK message in the Response area. Check the Atom feed to verify the changes you made.

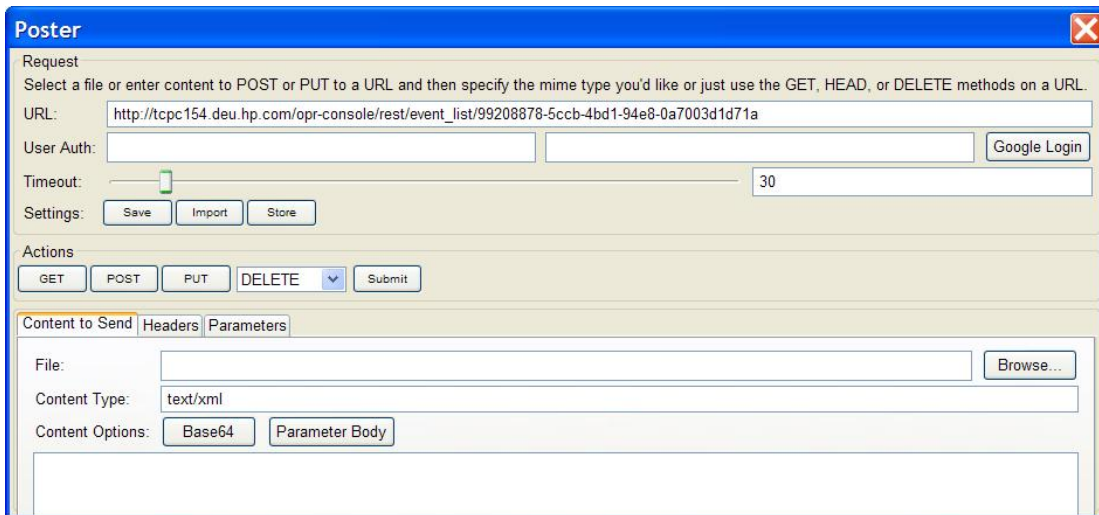
Modifying Events Using the Firefox Poster Extension

To illustrate how you can modify events using a REST client, Here we will look at how to use the Mozilla Firefox Poster Extension to modify events. The Poster Extension is a simple REST client that you first need to install as a plug-in for Firefox.

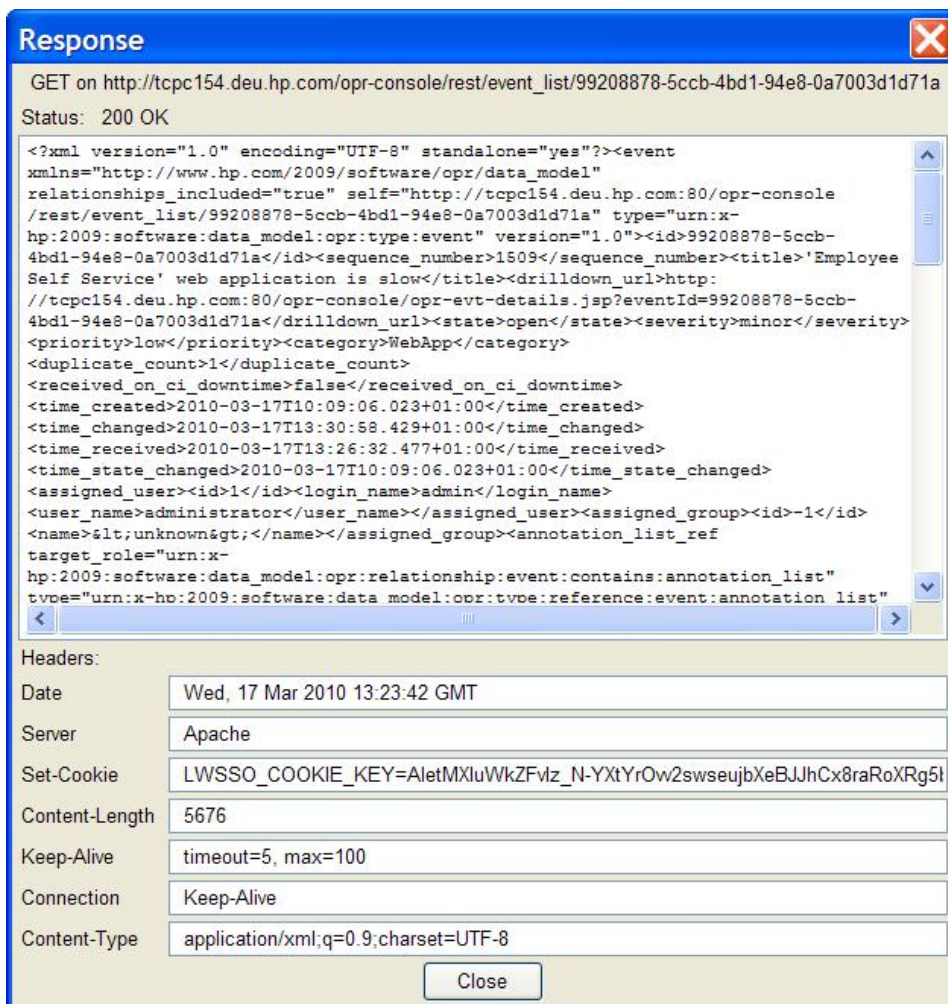
We will use an example of how to change the title and severity of an event using the Firefox Poster Extension. The steps required are as follows:

1. First install the Mozilla Firefox Poster Extension if it is not already installed.
2. Open a Firefox browser, and select **Tools > Poster**. This opens a Poster dialog box.
3. Get the event ID of the event you want to modify from the event list feed.
4. In the URL field of the Poster dialog box, enter the URL including the event ID of the event you want to modify. The syntax is as follows:

`https://<server.example.com>/opr-web/rest/9.10/event_list/<event_ID>`



5. Click **GET** to receive the event as XML. A window entitled Response opens, and if there are no errors, the status should be stated as 200 OK and you should be able to see the full XML of the event you want to modify.



6. Copy the full XML from the Response window and paste it into content text field of the Content to Send tab in the Poster dialog box. You no longer need the Response window, so you can close it now.
7. Next, you can edit the XML according to the changes you want to make to the event. It is not necessary to send the complete XML back to the server. As long as you preserve the XML structure, you can choose to send only those XML elements that you want to update for that event.

Note: Not all properties can be updated. For a list of editable properties, see ["Editable Properties" on page 186](#). Also check the latest Java API documentation, which is contained in a zip file that you can find in the following location:

```
<OMi_HOME>/opr/api/doc/opr-external-api-javadoc.zip
```

Unzip the contents of the zip file to a suitable location to see the API documentation.

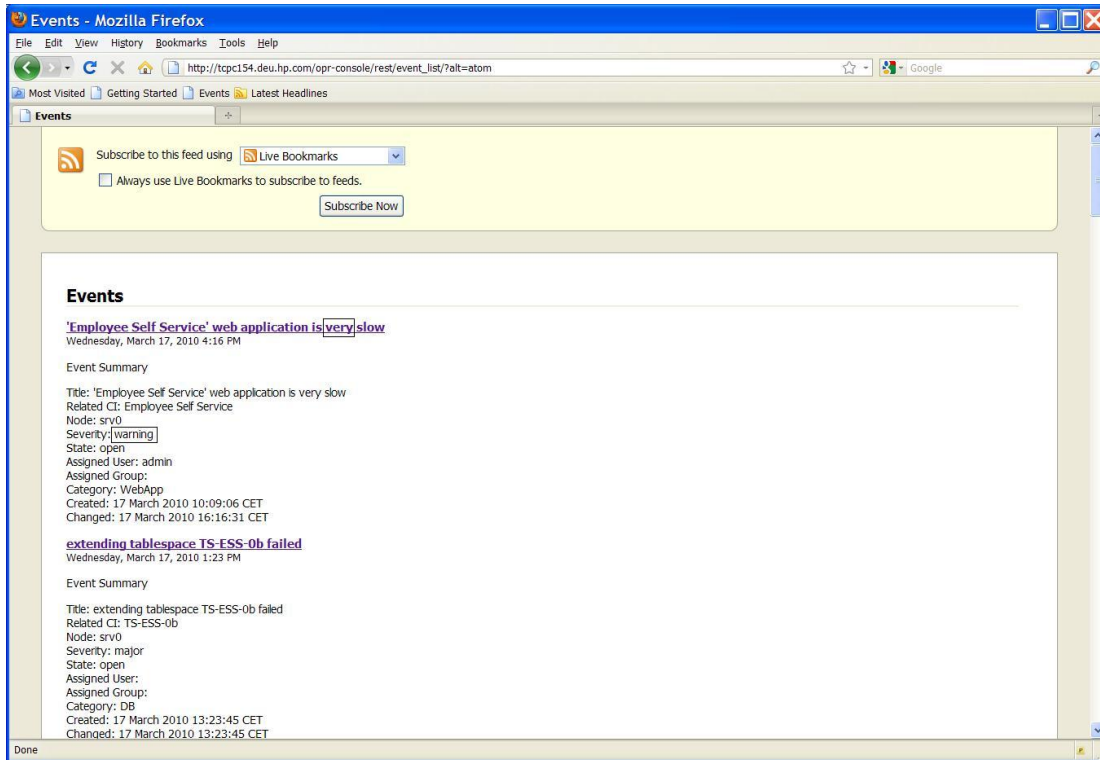
In this example, we will change the title of the event, and also the event severity, directly in the XML. For example, the event's original title was: "'Employee Self Service' web application is slow" and we want to edit the event title by inserting the word "very":

...<title>'Employee Self Service' web application is **very slow**</title>...

Similarly, we could change the event severity from minor to warning:

...<severity>**warning**</severity>...

- Once you have made your changes to the event, make sure that the Content Type field is set to **application/xml** (you must type this in). Then click **PUT** to save your changes. A Response window opens, and if there are no errors, you see an HTTP 200 OK message. Check the Atom feed to verify the changes you made.



You can find more information about Firefox Poster Extension at the following location:

<http://code.google.com/p/poster-extension/>

Modifying Events Using the RestWsUtil Utility

A REST web service utility is provided to allow you to execute REST web service operations against the event web service from the command-line. With this utility, you can execute one of the four following REST web service operations:

CRUD Operation	HTTP Method
Create	POST
Read	GET
Update	PUT
Delete	DELETE

You can find the utility in the following location:

```
<OMi_HOME>/opr/bin
```

You can modify events using the RestWsUtil command-line utility.

Example: How to Change an Event Title

To change the title of an event using the RestWsUtil utility, follow these steps:

1. Get the event ID of the event you want to modify.
2. Write the XML for the event you want to modify to an XML file (called, for example, update.xml) in the **<OMi_HOME>/opr/bin** directory.

The contents of the update.xml file look like this:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model"><title>  
New title goes here</title></event>
```

Edit the title.

3. To update and save the change, enter the following command in a command prompt:

```
<OMi_HOME>/opr/bin>RestWsUtil -update update.xml -username <login name> -  
password <password> -url "https://<server.example.com>/opr-web/rest/9.10/event_  
list/<event_ID>"
```

Where:

<OMi_HOME> is the directory where OMi is installed, <login name> is the user name required for authentication, and <event_ID> is the ID of the event you want to modify.

For further details and examples of how to use the RestWsUtil command-line utility, see ["RestWsUtil Command-Line Interface" on page 190](#).

How to Create New Events

You can create new events using the RestWsUtil command-line utility.

Note: To create events, users must have been granted the Event Submission permission in the OMi User Management settings. For details about how to do this, see OMi Help.

Example: How to Create a New Event

To create a new event using the RestWsUtil utility, follow these steps:

1. Create an XML file in the following folder (called, for example, create.xml):
<OMi_HOME>/opr/bin/create.xml
2. Define the properties of the new event by adding the following lines to the file:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <title>New event title</title>  
  <severity>normal</severity>  
  <priority>low</priority>  
  <state>open</state>  
</event>
```

3. Open a command prompt and type:

```
<OMi_HOME>/opr/bin>RestWsUtil -create create.xml -username <Login name> -  
password <password> -url "https://<server.example.com>/opr-web/rest/9.10/event_  
list"
```

Where:

<OMi_HOME> is the directory where OMi is installed, <Login name> is the user name required for authentication.

For further details and examples of how to use the RestWsUtil command-line utility, see ["RestWsUtil Command-Line Interface" on page 190](#).

Advanced Modification of Event Properties

An event has a selected number of list properties that can be modified by the OPR Event services listed in the Service Document, described in the section ["How to Access the Event Web Service" on page 158](#).

To generate an XML list of the custom attribute properties of the specified event, you call the following URL:

```
https://<server.example.com>/opr-web/rest/9.10/event_list/<event_ID>/custom-  
attribute-list/
```

Where:

<server.example.com> is the name of the gateway server, and <event_ID> is the ID of the event for which you want to list the custom attributes.

The REST response to such a URL call (in this case for an event with the ID 26629e00-2d8d-71dd-1aa2-1039228c0111) could look like this:

```
<custom_attribute_list  
  xmlns="http://www.hp.com/2009/software/opr/data_model"  
  self="http://<server.example.com>/ws/rest/event_list/26629e00-2d8d-71dd-1aa2-  
1039228c0111/custom_attribute_list"  
  type="urn:x-hp:2009:software:data_model:opr:type:event:custom_attribute_list"  
  version="1.0">  
  <custom_attribute  
    self="http://<server.example.com>/ws/rest/event_list/26629e00-2d8d-71dd-1aa2-  
1039228c0111/custom_attribute_list/drilldown.url.ci"  
    type="urn:x-hp:2009:software:data_model:opr:type:event:custom_attribute"  
    version="1.0">  
    <name>drilldown.url.ci</name>  
    <value>http://url.to/drill/down/ci</value>  
  </custom_attribute>  
  <custom_attribute  
    self="http://<server.example.com>/ws/rest/event_list/26629e00-2d8d-71dd-1aa2-  
1039228c0111/custom-attribute-list/drilldown.url.event"  
    type="urn:x-hp:2009:software:data_model:opr:type:event:custom_attribute"  
    version="1.0">  
    <name>drilldown.url.event</name>
```



```
<value>http://url.to/drill/down/event</value>
</custom_attribute>
</custom_attribute_list>
```

Using the Event Web Service, you can create and edit list items, and also delete items from the custom attributes list.

Note: You cannot create or delete an event itself using the RestWsUtil command-line utility.

To create a custom attribute item for a specified event, you send an HTTP POST request with the corresponding XML object to the event custom attribute list URL. When you call the custom attribute list URL for your specified event, you see that your new item has been added to the list of custom attributes for that event.

To edit an item in the list of custom attributes, you do something very similar. You send an HTTP PUT Request specifying an existing item name and a changed value for that item. The Event Web Service updates the value to the new value.

The HTTP response could look like this:

```
<custom_attribute
  xmlns="http://www.hpe.com/2009/software/opr/data_model"
  self="http://<server.example.com>/ws/rest/event_list/26629e00-2d8d-71dd-1aa2-
1039228c0111/custom_attribute_list/mynewattribute"
  type="urn:x-hp:2009:software:data_model:opr:type:event:custom_attribute"
  version="1.0">
  <name>mynewattribute</name>
  <value>Hello</value>
</custom_attribute>
```

Similarly, to delete an item from the list of custom attributes, you send an HTTP DELETE request to the following URL:

```
https://<server.example.com>/opr-web/rest/9.10/event_list/<event_ID>/custom_
attribute_list/<custom_attribute_name>
```

Where:

<custom_attribute_name> is the name of the custom attribute selected for deletion.

Bulk Update of Events

In addition to updating events individually with reference to the specific event ID, you can also update events in bulk. This is done not by addressing a specific event, but by addressing the event list with a query (see ["Event Web Service Query Language" on page 171](#)), and specifying the update in the payload of a PUT request.

For example:

URL: `https://server.example.com/opr-web/rest/9.10/event-list?query=title%20LIKE%20'%25db down%25'`

HTTP Method: PUT

```
Payload: <event xmlns="http://www.hp.com/2009/software/opr/data_model">
          <severity>major</severity>
        </event>
```

The above request sets all events with db down in the title to severity major.

HTTP Method: PUT

```
Payload: <event xmlns="http://www.hp.com/2009/software/opr/data_model">
          <state>closed</state>
        </event>
```

The above request closes all events with db down in the title.

Bulk Insert of Events

In addition to inserting individual events, a list of events may be inserted in a single Web Service call. This is done by executing a POST request with the following properties:

- An event_list rather than just a single event, for example:

```
<event_list xmlns="http://www.hp.com/2009/software/opr/data_model">
  <event>
    <title>Major Event</title>
    <severity>major</severity>
  </event>
  <event>
    <title>Minor Event</title>
    <severity>minor</severity>
  </event>
</event_list>
```

- A Content-Type of "application/xml; type=collection" or "text/xml; type=collection".
When using the RestWsUtil utility, you can use the -content_type option to specify the content type.

Chapter 21: Event Web Service Query Language

The Event Web Service uses standard query parameters for its URL query language.

Query parameters are used to modify the response from a resource in some way. For all resources, the response media type can be controlled. If a resource is a collection, the set of entries returned in the collection can be filtered in a variety of ways. Query parameters can be combined in interesting ways to further constrain a response.

The Event Web Service offers a number of parameters for filtering the event list by specific criteria. For example, you can filter the results by matching specific event parameters to a condition in a URL query parameter. It is also possible to reduce the list size by displaying only a certain number of list items and page through them. You can also send a time and date parameter which returns only those events that have been updated after the specified time and date.

HTTP Query Parameters

This section describes the HTTP query parameters that are supported by the Event Web Service.

These query parameters apply only to collection resources and are only meaningful when used with the HTTP GET method, or when updating events in bulk with the HTTP PUT method. They are specified within the HTTP query portion of the URL that addresses the resources.

A client can specify that the response feed should be filtered by only including resources where the meta-data or data for the resource matches certain criteria. This can be specified using the query parameter within the HTTP query portion of the URL.

["Query Filter Criteria Properties" on page 178](#) lists the available query filter criteria properties and supported operators.

This section includes:

- ["List of HTTP Query Parameters" on the next page](#)
- ["Filtering by Date and Time: watermark" on page 173](#)
- ["Filtering by Event Attributes: query" on page 174](#)
- ["Paging" on page 175](#)
- ["Ordering" on page 176](#)
- ["Data Inclusion" on page 177](#)
- ["Media Type" on page 178](#)

List of HTTP Query Parameters

The following are a list of parameters that may be specified in the HTTP query portion of the URL. Query parameters can be specified in combination, separated by the ampersand (&) operator.

Query Parameter	Description
watermark	Parameter for filtering by date and time. For details, see "Filtering by Date and Time: watermark" on the next page.
query	Parameter for filtering by event attributes. For details, see "Filtering by Event Attributes: query" on page 174.
start_index	Paging query parameter used to define the item from which to start the query. For details, see "Paging" on page 175.
page_size	Paging query parameter used to define how many items are displayed per page of an atom feed. For details, see "Paging" on page 175.
order_by	Ordering parameter used to specify that the response feed should be ordered by the indicated field. For details, see "Ordering" on page 176.
order_direction	Ordering parameter used to specify whether the response feed should be ordered in either descending or ascending order. For details, see "Ordering" on page 176.
include_closed	Parameter used to specify whether closed events are included when querying the Events Service (event_list). Default value: false For details, see "Data Inclusion" on page 177.
include_relationships	Parameter used to specify whether relationships are included when querying the Events Service (event_list). Default value: true For details, see "Data Inclusion" on page 177.
include_history	Parameter used to specify whether history lines are included in the events when querying the Events Service (event_list). Default value: false For details, see "Data Inclusion" on page 177.

Query Parameter	Description
include_cis	Parameter used to specify whether configuration item properties are included in the events for related CIs, nodes, and affected business services when querying the Events Service (event_list). Default value: true For details, see "Data Inclusion" on page 177 .
include_annotations	Parameter used to specify whether annotations are included in the events when querying the Events Service (event_list). Default value: false For details, see "Data Inclusion" on page 177 .

Filtering by Date and Time: watermark

A client can specify that the response feed should be filtered based on time. These query parameters apply only to collection resources and are only meaningful when used with the HTTP GET method.

You can query items by time and date using the `watermark` parameter. If the `watermark` parameter is specified, only events that have been created or updated after the specified time are returned. For example, when an application that is monitoring new and changed events using the web service shuts down, it will want to remember the time of the last changed event. In this way when it restarts it can query for only those new and changed events since this timestamp.

For instance, `watermark=2009-01-01T00:00:00Z` indicates that the only resources to be included in the response feed have been updated after the beginning of 2009.

Alternatively, you can get a list of events from a certain sequence number onwards by specifying the `sequence_number` parameter.

Time format requirements:

- **dateTime Format.** Like all times specified in a query, the value for the `watermark` parameter must be specified in the XML Schema `dateTime` format. If there are no resources in the collection that have been updated after the specified date, the response feed will be empty. A poorly formatted value for the query parameter (for example, one that does not conform to the XML Schema `dateTime` format) will cause a fault response.

For more details about XML Schema data types, refer to the XML schema document, which you can find in the following location:

<http://www.w3.org/TR/xmlschema-2>

- **Time Zones Encoding.** When specifying a time zone other than GMT using "Z", you must specify a "+" character. Because this is in the URL it must be URL encoded by replacing it with "%2B". An example would be:

```
query=time_created%20GT%202009-10-19T17:06:54.453%2B02:00
```

For a list of URL escape codes for characters that must be escaped, see ["URL Escape Codes for Characters that Must Be Escaped" on page 184](#).

For full details about URL encoding, refer to the Uniform Resource Locators (URL) Specification at the following location:

<http://www.rfc-editor.org/rfc/rfc1738.txt>

Examples:

- An example of a URL call where only events after 13:59:17 on March 19, 2010, are listed would look like this:

```
https://<server.example.com>/opr-web/rest/9.10/event_  
list?format=atom&watermark=2010-03-19T13:59:17%2B02:00
```

- The following example URL returns the first 40 currently open events since 15:59:17 on March 4, 2010 in an atom feed format.

```
https://<server.example.com>/opr-web/rest/9.10/event_  
list/?format=atom&watermark=2010-03-04T15:59:17%2B02:00&page_size=40&start_  
index=1
```

Filtering by Event Attributes: query

The query parameter facilitates filtering the request using specific event attribute values. The criteria for the query filter are:

- A filter property specifying an event attribute
- A supported operator
- A value for the property

Simple Filters

As a simple example, to filter all the queries where the severity property equals `critical`, the web service client would need to request the following URL:

```
https://<server.example.com>/opr-web/rest/9.10/event_  
list?format=atom&query=severity%20EQ%20"critical"
```

Compound Filters

You can compose multiple filters for more complex filter queries. Filter query parameters can be used in combination, separated by logical NOTs, ANDs, and ORs, which are resolved in the order specified. To achieve a different processing order of the query, use parentheses.

Examples:

- The following example shows two simple filters made into a compound filter query using the logical AND:

```
https://<server.example.com>/opr-web/rest/9.10/event_list?query=assigned_  
user%20EQ%20"admin"%20AND%20title%20EQ%20"My Title"
```

- The following example shows a compound filter query made up of a combination of a simple and a complex filter, using a logical AND:

```
https://<server.example.com>/opr-web/rest/9.10/event_list?query=assigned_
user%20EQ%20"admin"%20AND%20related_ci[target_
id%20EQ%20"ffca22eea15268533029f17b4c01b008"]
```

- The next example shows how you would compose a compound filter so that the logical ORs are resolved first.

```
https://<server.example.com>/opr-web/rest/9.10/event_list?query=assigned_
user%20EQ%20"admin"%20AND%20(title%20EQ%20"My
Title"%20OR%20state%20EQ%20"closed")
```

- The next example shows how you would use the NOT operator to negate an expression:

```
https://<server.example.com>/opr-web/rest/9.10/event_list?query=assigned_
user%20EQ%20"admin"%20AND%20(title%20EQ%20"My
Title"%20OR%20NOT%20state%20EQ%20"closed")
```

Paging

You can optimize the query result by specifying some additional URL parameters. Paging query parameters are used to filter the entries in a response feed to a subset of those that would be returned without the query parameters. Paging query parameters apply to collection resources and will have no affect on an individual resource. They are only meaningful when used with the HTTP GET method. Paging query parameters are applied to a response feed after considering all other filtering query parameters.

There are two query parameters that work together to provide an interface that a client can use to page through a feed with a large number of entries: `start_index`, and `page_size`.

start_index

The `start_index` query parameter is used to specify the index of the first entry returned in a response feed. So you can define the item from which to start the query with the `start_index` parameter. The first entry in a feed has an index of 1, the second entry an index of 2, and so on. The default value for `start_index` is always 1 if the query parameter is not specified. The request returns a fault if a value less than 1 is specified. A value greater than the number of entries in the collection will return an empty response feed.

page_size

The `page_size` query parameter is used to specify the number of entries returned at one time. So you can define how many items are displayed (on one page of an atom feed) with the `page_size` parameter. The default for the Event Web Service is set to 20 items.

The minimum value is 1 and the maximum value can be any number that the service is able to support. There is no default value that all applications are expected to use when the `page_size` parameter is not set. The Event Web Service returns an error if a value less than 1 is specified.

The following example returns a page with 30 items of query results filtered by the `severity` parameter set to equal `warning`:

```
https://<server.example.com>/opr-web/rest/9.10/event_
list?format=atom&query=severity%20EQ%20"warning"&page_size=30
```

Combining `start_index` and `page_size`

By combining the `start_index` and the `page_size` parameters, you can view the result distributed over more than one page. The value of `start_index` must be greater than 0.

For example, if you call the URL with a `page_size` of 5, but do not define the `start_index` value, the first five items (items 1 - 5) will be returned.

If you call the URL with a `page_size` of 5, and define the `start_index` value as 6, five items starting from the sixth item (items 6 - 10) will be returned on the second page. So the result of the query would be that 10 items are returned, distributed over two pages of an atom feed.

So, to return the first page (items 1 - 5) of query results filtered by the `severity` parameter set to equal warning, you would call the following URL:

```
https://<server.example.com>/opr-web/rest/9.10/event_
list?format=atom&query=severity%20EQ%20"warning"&page_size=5
```

To get the second page (items 6 - 10) of query results, you would call the following URL:

```
https://<server.example.com>/opr-web/rest/9.10/event_
list?format=atom&query=severity%20EQ%20"warning"&page_size=5&start_index=6
```

A client using paging can keep the page size constant for every request or can vary the page size for every request. The pages can also overlap. In the examples above with `start_index` set to 3 and `page_size` set to 5, the requested page will overlap with the first page. There is no guarantee that a client will not see duplicate entries, or that a client may miss some entries due to additions or deletions between page requests.

Links are set in all feeds to indicate how to get to the first or last page. When not all of the entries are returned due to paging, links also indicate how to get the next and previous pages. The links are described by setting the `rel` attribute to "first", "last", "next", or "previous". These links are described in RFC5005, which you can access at the following location:

<http://tools.ietf.org/html/rfc5005>

Ordering

A client can specify that a response feed should be returned with certain ordering criteria. These query parameters apply only to collection resources and are only meaningful when used with the HTTP GET method.

`order_by`

The `order_by` query parameter is used to specify that the response feed should be ordered by the indicated field. The field may be any simple data or meta-data property of the resource.

If the `order_by` query parameter is a time or the `sequence_number`, then the default ordering is descending, so that the newest entries appear first, otherwise it is ascending.

`order_direction`

The `order_direction` query parameter is used to specify that the response feed should have an order

direction as indicated. There are only two valid values for this query parameter: “ascending”, and “descending”. Any other value causes a fault response. The default value if none of the ordering query parameters is specified is descending (the APP specification indicates that a feed is ordered descending by update time).

Data Inclusion

With data inclusion query parameters, a client can control the amount of relevant data returned by the query, thereby directly influencing the performance of the response.

include_closed

The `include_closed` query parameter is used to specify whether closed events are included when querying the Events Service (`event_list`).

The default value of this parameter is `false`. If set to `true`, closed events are included in the query, otherwise only events that do not have a lifecycle state of `closed` are returned from the Event Web Service.

include_relationships

The `include_relationships` query parameter is used to specify whether relationships are included when querying the Events Service (`event_list`).

The default value of this parameter is `true`. If set to `false`, relationships are not included in the query. For example, if `related_ci` or `source_ci` is set in the event, and the `include_relationships` query parameter is set to `false`, only the CI IDs are returned from the Event Web Service. Key attributes, including the container CI (`part_of`), are not resolved or returned, in contrast to the case when the `include_relationships` parameter is set to the default value `true`.

include_history

The `include_history` query parameter is used to specify whether history lines are included in the events when querying the Events Service (`event_list`).

The default value of this parameter is `false`. If set to `true`, history lines are included in the query, otherwise only a reference to a history line list is included.

include_cis

The `include_cis` query parameter is used to specify whether configuration item (CI) properties of related CIs, nodes, and affected business services are included when querying the Events Service (`event_list`).

The default value of this parameter is `true`. If set to `false`, CI properties are not included in the query. For example, if `related_ci` or `source_ci` is set in the event, and the `include_cis` query parameter is set to `false`, only the CI IDs are returned from the Event Web Service. CI details such as `user_label` or `create_time` are not returned.

include_annotations

The `include_annotations` query parameter is used to specify whether annotations are included in the events when querying the Events Service (`event_list`).

The default value of this parameter is `false`. If set to `true`, annotations are included in the query.

Media Type

A client can request that a response be returned with a specified media type.

format

The `format` query parameter is used by a client to tell a server that it would like to have the response returned using the specified media type. This query parameter applies to all resources, including collections, and can be used with any HTTP method that returns a response. The value of the parameter is a media type for example, `application/atom+xml`, `application/json`, `application/xml`, and so on). Usually, a service will have a limited set media types that it is able to format a given resource. The list of supported media types is available from the Atom response format (for example, `format=atom`, `format=json`, `format=xml`), which is supported by most resources (see ["How to Detect New Events" on page 160](#)).

The `format` query parameter expresses the same client desire as the Accept HTTP header. The advantage of using the HTTP header is that many clients already have built-in support for the Accept header. The advantage of using the `format` query parameter is that a link with a specific response format can be passed around in email, chat, twitter, documents, etc. without also having to say that you need to set the Accept header to a certain value. When the service receives a message having both the Accept header and the `format` query parameter set, the `format` query parameter takes precedence.

Query Filter Criteria Properties

["Query Filter Criteria Properties" below](#) lists the available query filter criteria properties and supported operators.

Note: If you want to filter using complex (nested) attributes, you need to use brackets ([]) to show the nesting. When specifying complex attributes, the bracket characters ([]) must be escaped, for example, like this: `query=related_ci%5Btarget_id="ffc22eea15268533029f17b4c01b008"%5D`. For more information about filtering using complex attributes, see ["Complex Attributes" on page 185](#).

Query Filter Criteria Properties

Property	Supported Operators	Type of Value	Supports order_by	Default for order_direction
application	EQ, NE, LIKE, IS NULL, IN	String Literal	yes	ascending

Property	Supported Operators	Type of Value	Supports order_by	Default for order_direction
assigned_group	EQ, NE, IN	String Literal	yes	ascending by ID
assigned_group[id]	EQ, NE, IN	String Literal	yes	ascending by ID
assigned_group[name]	EQ, NE, IN	String Literal	yes	ascending by ID
assigned_user	EQ, NE, IN	String Literal	yes	ascending by ID
assigned_user[id]	EQ, NE, IN	String Literal	yes	ascending by ID
assigned_user[login_name]	EQ, NE, IN	String Literal	yes	ascending by ID
category	EQ, NE, LIKE, IS NULL, IN	String Literal	yes	ascending
cause[target_id]	EQ, IS NULL, IN	String Literal (UUID)		
close_key_pattern	EQ, NE, LIKE, IS NULL, IN	String Literal		
control_transferred_to[dns_name]	EQ, NE, LIKE	String Literal		
control_transferred_to[external_id]	EQ, NE, LIKE, IS NULL, IN	String Literal		
control_transferred_to[id]	EQ, NE, LIKE, IS NULL, IN	String Literal (UUID)		
control_transferred_to[state]	EQ, NE, IS NULL, IN	String Literal (UUID)		
control_transferred_to[state]	EQ, NE, IS NULL, IN	String Literal (UUID)		
ca[<CA name>]	EQ, NE, LIKE, IN	String Literal		
ca[<CA name>]	EQ, NE, LIKE, IN	String Literal		
custom_attribute_list[<CA name>]	EQ, NE, LIKE, IN	String Literal		
description	EQ, NE, LIKE, IS NULL, IN	String Literal	yes	ascending
duplicate_count	EQ, LT, GT, LTE, GTE, NE, IN	Integer	yes	ascending

Property	Supported Operators	Type of Value	Supports order_by	Default for order_direction
eti_hint	EQ, NE, LIKE, IS NULL, IN	String Literal		
eti_value_id	EQ, NE, LIKE, IS NULL, IN	String Literal		
id	EQ, NE, IN	String Literal (UUID)		
instruction_available	EQ	Boolean		
key	EQ, NE, LIKE, IS NULL, IN	String Literal		
log_only	EQ	Boolean		
node[target_global_id]	EQ, NE, IS NULL, IN	String Literal		
node[target_id]	EQ, NE, IS NULL, IN	String Literal		
node_hints[hint]	EQ, NE, LIKE, IS NULL, IN	String Literal		
node_hints[node_core_id]	EQ, NE, LIKE, IS NULL, IN	String Literal		
node_hints[node_dns_name]	EQ, NE, LIKE, IS NULL, IN	String Literal		
node_hints[node_ip_address]	EQ, NE, LIKE, IS NULL, IN	String Literal		
object	EQ, NE, LIKE, IS NULL, IN	String Literal	yes	ascending
om_service_id	EQ, NE, LIKE, IS NULL, IN	String Literal		
om_user	EQ, NE, LIKE, IS NULL, IN	String Literal	yes	ascending
original_data	EQ, NE, LIKE, IS NULL, IN	String Literal		
original_id	EQ, NE, LIKE, IS NULL, IN	String Literal		
priority	EQ, NE, LIKE, IS NULL, IN	String Literal	yes	ascending

Property	Supported Operators	Type of Value	Supports order_by	Default for order_direction
received_as_notify	EQ	Boolean		
received_on_ci_downtime	EQ	Boolean		
related_ci[sub_component]	EQ, NE, LIKE, IS NULL, IN	String Literal		
related_ci[sub_component]	EQ, NE, LIKE, IS NULL, IN	String Literal		
related_ci[target_global_id]	EQ, NE, IS NULL, IN	String Literal		
related_ci[target_id]	EQ, NE, LIKE, IS NULL, IN	String Literal		
related_ci_hints[hint]	EQ, NE, LIKE, IS NULL, IN	String Literal		
sequence_number	EQ, LT, GT, GTE, LTE, NE, IN	Integer	yes	descending
severity	EQ, NE, IS NULL, IN	String Literal	yes	ascending
skip_duplicate_suppression	EQ	Boolean		
solution	EQ, NE, LIKE, IS NULL, IN	String Literal	yes	ascending
source_ci[target_global_id]	EQ, NE, IS NULL, IN	String Literal		
source_ci[target_id]	EQ, NE, IS NULL, IN	String Literal		
source_ci_hints[hint]	EQ, NE, LIKE, IS NULL, IN	String Literal		
source_ci_hints[node_core_id]	EQ, NE, LIKE, IS NULL, IN	String Literal		
source_ci_hints[node_dns_name]	EQ, NE, LIKE, IS NULL, IN	String Literal		
source_ci_hints[node_ip_address]	EQ, NE, LIKE, IS NULL, IN	String Literal		

Property	Supported Operators	Type of Value	Supports order_by	Default for order_direction
sourced_from[dns_name]	EQ, NE, LIKE, IS NULL, IN	String Literal		
sourced_from[external_id]	EQ, NE, LIKE, IS NULL, IN	String Literal		
sourced_from[id]	EQ, NE, IS NULL, IN	String Literal (UUID)		
state	EQ, NE, IS NULL, IN	String Literal	yes	ascending
sub_category	EQ, NE, LIKE, IS NULL, IN	String Literal	yes	ascending
time_changed	LT, GT, GE, LE	Datetime	yes	descending
time_created	LT, GT, GE, LE	Datetime	yes	descending
time_first_received	LT, GT, GE, LE	Datetime	yes	descending
time_received	LT, GT, GE, LE	Datetime	yes	descending
time_state_changed	LT, GT, GE, LE	Datetime	yes	descending
title	EQ, NE, LIKE, IS NULL, IN	String Literal	yes	ascending
type	EQ, NE, LIKE, IS NULL, IN	String Literal	yes	ascending

Operator Aliases

"Operator Aliases" below lists the verbose aliases for operators.

Operator Aliases

Operator	Alias
=	EQ
!=	NE
<	LT
<=	LTE
>	GT
>=	GTE

Operator	Alias
LIKE	n/a

Value Types

The query filter criteria properties listed in ["Query Filter Criteria Properties" on page 178](#) can take the following types of values:

Value Types

Value Type	Description
String Literals	<p>If you want to filter using string literals, you need to surround them with double quotation marks ("). Comparisons of string literals are case sensitive.</p> <p>If you need to use one of the characters that must be escaped in a string literal (see "URL Escape Codes for Characters that Must Be Escaped" on the next page, you need to escape the character using the dollar sign (\$) instead of percent (%); for example, use <code>query=title%20EQ%20"\$3CMy title\$3E"</code> instead of <code>query=title%20EQ%20'%3CMy title%3E'</code>.</p> <p>The following query returns all events whose title contains the strings <code>cpu</code> and <code>value</code>. The URL encoding <code>\$25</code> represents the SQL wildcard <code>%</code>, which matches zero or more characters (similar to the regular expression <code>.*</code>):</p> <pre>query=title%20LIKE%20"\$25cpu\$25value\$25"</pre> <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <p>Note: OMi 9.0x and earlier only supports single quotation marks (') and the percent (%) escape codes. String literal escape codes using the dollar sign (\$) are not supported in OMi 9.0x. OMi 9.10 and later supports single (') and double (") quotation marks. It is recommended that Event Web Service clients use double quotation marks.</p> </div>
Datetime	<p>Must be specified in the XML Schema <code>dateTime</code> format. For more details about XML Schema data types, refer to the XML schema document, which you can find in the following location: http://www.w3.org/TR/xmlschema-2</p> <p>Additionally, when specifying a time zone other than GMT using "Z", you must specify a "+" character. Since this is in the URL it must be URL encoded. See "URL Escape Codes for Characters that Must Be Escaped" on the next page.</p>
Integer	Positive and negative natural numbers, including 0.

Query Using the POST Method

A limitation may exist for the maximum length of the URLs accepted by servers. If this length is exceeded, a 414 status is returned indicating `Request-URI Too Long`. It is more common for an intermediary to exhibit this type of limitation. When a client receives this status in a response, it is very

probably caused by a long query expression. The client should re-try the request with the following changes:

- Change the HTTP method from GET to POST
- Set the Content-Type header to `application/x-www-form-urlencoded`
- Remove the query query parameter from the URL and set the body of the request to the query query parameter. For example, the message body could be as follows: `query=severity='critical'`.
- Re-send the request

URL Escape Codes

"[URL Escape Codes for Characters that Must Be Escaped](#)" below lists the characters that must be escaped in URLs.

If you need to escape a character in a string literal, you need to use the dollar sign (\$) instead of percent (%); for example, use `query=title%20EQ%20"$3CMy title$3E"` instead of `query=title%20EQ%20'%3CMy title%3E'`.

URL Escape Codes for Characters that Must Be Escaped

Character	URL Escape Codes	String Literal Escape Code
SPACE	%20	\$20
<	%3C	\$3C
>	%3E	\$3E
#	%23	\$23
%	%25	\$25
+	%2B	\$2B
{	%7B	\$7B
}	%7D	\$7D
	%7C	\$7C
\	%5C	\$5C
^	%5E	\$5E
~	%7E	\$7E
[%5B	\$5B
]	%5D	\$5D
'	%60	\$60

URL Escape Codes for Characters that Must Be Escaped, continued

Character	URL Escape Codes	String Literal Escape Code
;	%3B	\$3B
/	%2F	\$2F
?	%3F	\$3F
:	%3A	\$3A
@	%40	\$40
=	%3D	\$3D
&	%26	\$26
\$	%24	\$24

White Spaces in URLs

You can use the following characters or strings to represent white space in the query portion of a URL:

- White space ()
- Plus sign (+)
- URL escape code (%20)
- String literal escape code (\$20)

Complex Attributes

If you want to filter using complex (nested) attributes, you need to use square brackets ([]) to show the nesting.

Complex attribute filters allow multiple nesting for subjacent attributes and can also be combined with other filters.

Here are examples of URL calls containing a complex attribute:

```
https://<server.example.com>/opr-web/rest/9.10/event_list?query=related_ci%5Btarget_id="ffca22eea15268533029f17b4c01b008"%5D
```

```
https://<server.example.com>/opr-web/rest/9.10/event_list?query=custom_attribute_list%5BMyCaName%20NE%20"%5D
```

Note: When specifying complex attributes, the bracket characters ([]) must be escaped. For more information about URL escape codes, see ["URL Escape Codes" on the previous page](#).

Editable Properties

Event properties that you can edit within the Event Web Service are listed in "[List of Editable Event Properties](#)" below.

Note: This is a summary of the com.hp.opr.ws.model.event Java API Documentation. For the latest information about editable properties, refer to the Java API Documentation.

See "[File Locations](#)" on page 189 for access information for the Java API Documentation.

List of Editable Event Properties

Name of Tag	Type	Description	Comment
title	String	Title of the event	
description	String	Description of the event	
solution	String	A solution	
state	String	Lifecycle state of the event	Value can be one of: closed, in_progress, open, resolved
severity	String	Severity of the event	Value can be one of: major, minor, critical, warning, normal, unknown
priority	Integer	Priority of the event	
assigned_user	User	User assigned to the event	Example: <name>User</name>
assigned_group	Group	User group assigned to the event	Example: <name>Users</name>
cause	ID	Identifier of the cause	
control_transferred_to	ID or name	ID or name of the connected server	Set in the control_transferred_to structure
symptom_list	List	List of symptoms	Symptoms can only be created, updated, or deleted using the Symptom web service

List of Editable Event Properties, continued

Name of Tag	Type	Description	Comment
custom_attribute_list	List	List of custom attributes	Custom attributes can only be created, updated, or deleted using the Custom Attribute web service
annotation_list	List	List of annotations	Annotations can only be created, updated, or deleted using the Annotation web service
auto_action	None	Automatic action	POST to run the action, DELETE to stop the action using the Automatic Action web service only. Note that the body of the XML is empty.
user_action	None	Operator action	POST to run the action, DELETE to stop the action using the Operator Action web service only. Note that the body of the XML is empty.

History Lines

History lines enable you to track changes for an event. History lines provide information about the types, names, and previous and current values of event properties that changed for a specific event. Concurrent changes are represented as a single history line. They are available in read-only mode, and are generated automatically.

Recorded Property Changes

Event properties that are recorded when they are changed in the form of a history line are listed in ["Recorded Property Changes" below](#). In general, the following properties are recorded for each change: property name, previous value, current value, and time of change. For details on each of these properties, see the appropriate Javadoc for the Change Type listed in the package `com.hp.opr.api.ws.model.event.property`.

Recorded Property Changes

Name	ChangeType
annotation	OprAnnotationPropertyChange
application	OprStringPropertyChange
assigned_group	OprGroupPropertyChange
assigned_user	OprUserPropertyChange
category	OprStringPropertyChange
cause	OprStringPropertyChange
control_transfer_to	OprControlTransferredToPropertyChange

Recorded Property Changes, continued

Name	ChangeType
correlation_rule	OprCorrelationRulePropChange
custom_attribute	OprCustomAttributePropertyChange
description	OprStringPropertyChange
duplicate_count	OprIntegerPropertyChange
eti_hint	OprStringPropertyChange
key	OprStringPropertyChange
key_pattern	OprStringPropertyChange
node_hint	OprStringPropertyChange
object	OprStringPropertyChange
om_service_id	OprStringPropertyChange
om_user	OprStringPropertyChange
original_data	OprStringPropertyChange
original_id	OprStringPropertyChange
priority	OprPriorityPropertyChange
related_ci_hint	OprStringPropertyChange
severity	OprSeverityPropertyChange
solution	OprStringPropertyChange
source_ci_hint	OprStringPropertyChange
state	OprStatePropertyChange
sub_category	OprStringPropertyChange
time_created	OprTimePropertyChange
time_received	OprTimePropertyChange
title	OprStringPropertyChange
type	OprStringPropertyChange

Event Changes List

History lines are provided for a specific event and no filtering support is provided in this web service endpoint.

The Event Changes List returns all history lines available. You can access the event changes list by calling the following URL:

```
https://<server.example.com>/opr-web/rest/event_changes_list
```

The change list can be filtered using the standard HTTP query parameters. This list is used to detect event changes as they occur.

In order to receive the history lines from a specific date, you must specify a watermark query. See ["Filtering by Date and Time: watermark" on page 173](#) for more details about watermarks and specifying date and time.

File Locations

File locations for reference material for the Event Web Service interface are as follows:

- Event Web Service Java API Documentation:
`<OMi_HOME>/opr/api/doc/opr-external-api-javadoc.zip`
- Event Web Service Schema:
`<OMi_HOME>/opr/api/schema/OprDataModel.xsd`

Chapter 22: RestWsUtil Command-Line Interface

The RestWsUtil command-line interface enables you to:

- Carry out simple testing of the Event Web Service.
- Perform the four basic operations: create, read, update and delete.

Create and update require an input file with the payload to send to the REST web service. Read and delete do not set the payload.

Location

<OMI_HOME>/opr/bin/RestWsUtil[.bat|.sh]

Synopsis

RestWsUtil -help | -version | -verbose | <CONNECTION_INFO> | <operation>

Options

Syntax for <CONNECTION_INFO>

-username <login name> [-password <password> | -smartcard | -winCrypto | -jks <keystore path> -jksPassword <keystore password>] [[-port <port>] [-server <gatewayserver>] [-ssl] | [-u <URL>]]

Note: If <CONNECTION_INFO> is omitted, the command is executed on the server to which you are logged on.

Option	Description
{-username -user} <login name>	Sets the login name of the user required to execute CLI operations on the target gateway server.
{-password -pw} <password>	Sets the password for the specified user. If using SSH on Cygwin, either enter the password in free text or use other communication methods, for example Java keystore, Windows keystore, or smart card authentication. Default value: empty string

Option	Description
{-smartcard -sc}	<p>Use certificate stored on smart card or security token for authentication. When OMi is configured to use CAC authentication, the CLI tools under <OMi_HOME>/opr/bin/ do not directly prompt users to enter the password for the smartcard connected to the system. Instead, users must specify that a smartcard authentication is to be run, using the option -sc or -smartcard. Users attempting to run a tool without the -smartcard option automatically receive an error message.</p> <p>For more information, see Using Command-Line Tools When Client Certificate Enforcement Is Used.</p>
{-winCrypto -wc}	<p>If OMi is configured for TLS mutual authentication, this option specifies to use the Windows certificate store for authentication. The certificate store must hold exactly one client certificate, which OMi will use to authenticate the user. This option is only available on Windows systems.</p> <p>For details, see Configuring Client Certificate or Smart Card Authentication.</p>
{-jks -j} <keystore path>	<p>If OMi is configured for TLS mutual authentication, this option can be used to specify the Java keystore to be used for authentication. The keystore must hold exactly one client certificate, which OMi will use to authenticate the user.</p> <div style="background-color: #f0f0f0; padding: 5px; margin: 10px 0;"> <p>Note: It is not necessary that the client certificate contains the flag "Smart Card Logon (1.3.6.1.4.1.311.20.2.2)" in the "Enhanced Key Usage" field.</p> </div> <p>For details, see Configuring Client Certificate or Smart Card Authentication.</p>
{-jksPassword -jp} <keystore password>	<p>Password for accessing the Java keystore.</p>
{-port -p} <port>	<p>Uses port <port> to connect to the target gateway server.</p> <p>Default: 80 (HTTP) 443 (HTTPS)</p>
-server <GatewayServer>	<p>Sets the target gateway server, using <GatewayServer> as the hostname or IP address to locate it.</p> <p>Default: FQDN of the OMi gateway server</p>
-ssl	<p>When this option is specified, the HTTPS protocol is used to connect to the target gateway server. If omitted, the HTTP protocol is used.</p> <p>Cannot be used in conjunction with the -url option.</p>

Option	Description
{-url -u} <url>	Sets the target gateway server, using <url> as the URL to locate it. Default: https://<OMi gateway FQDN>:80/opr-config-server

Syntax for <operation>

(-h | -version | (-read | -delete | ((-create | -update) <filename> [-content_type <type>])[-output <filename>] | -version)

Option	Description
{-c -create} <filename>	Create the resource in the specified XML document. HTTP POST operation.
-content_type <type>	HTTP header content-type value to set for create and update operations. Default: application/xml
{-d -delete}	Delete the specified resource. The URL directly addresses the resource to delete. HTTP DELETE operation.
{-h -help}	Print this message and exit.
{-o -output} <out_file>	Name of output file for text returned from the web service request. Default: stdout
{-r -read}	Read the specified resource. HTTP GET operation.
{-up -update} <filename>	Update the resource with the specified XML document changes. HTTP PUT operation.
{-v -verbose}	Print verbose output.
-version	Print the version information and exit.

Exit Status

Exit Status	Description	Output
0	Successful completion of the requested operation	No output.

Exit Status	Description	Output
1	Failure of the requested operation	An error message stating why the operation failed, followed by the tool's help text.
300-399	HTTP Redirection (300-399)	An error message stating the HTTP error number and description. For more information about HTTP error status values, see publicly available HTTP documentation.
400-499	HTTP Client Error (400-499)	
500-599	HTTP Internal Server Error (500-599)	

Restrictions

The utility uses basic authentication and has parameters to specify a username and password.

Note: The appropriate identity assurance software, such as ActivIdentity, must be installed to be able to authenticate using a smart card when executing the RestWsUtil CLI.

When connecting to OMi servers using Remote Desktop Connections, the client system must also have the appropriate identity assurance software, such as ActivIdentity, installed to be able to authenticate using a smart card attached to the client system when executing the RestWsUtil CLI remotely.

In addition, in the Remote Desktop Connections dialog box, select More **Show Options > Local Resources > More** to open the Local devices and resources dialog box. Ensure that the **Smart Cards** check box is selected.

Non-admin users also need the following file permissions to operate this command-line tool:

File	Windows Permissions	Linux Permissions
<OMi_HOME>/conf/TopazInfra.ini	read	r
<OMi_HOME>/log/opr-clis.log	full control	rw
<OMi_HOME>/log/opr-pgctl.log	full control	rw
<p>Note: This file is not available on gateway server systems.</p>		
<OMi_HOME>/conf/encryption.properties	read	r
<OMi_HOME>/conf/seed.properties	read	r

Examples

Some examples of usage of the RestWsUtil CLI follow.

Reading Events

Here is an example of using the RestWsUtil command-line utility to read events and to send the results to an output file called test.xml.

```
RestWsUtil -r -username admin -o test.xml -verbose
Password: *****
INFO: Read the resource located at: http://server.example.com/opr-web/rest/9.10/event_list
INFO: Operation successful.
```

Reading Custom Attributes of an Event

To read the custom attributes of an event, you need to specify the complete URL, including the ID of the event you want to read the attributes for.

The following example sends the output to stdout, which is the default.

```
RestWsUtil -r
-url "http://<fully qualified domain name of OMi gateway server>/opr-web/rest/9.10/event_
list/0695624b-93fa-40b1-8b0fc9b4ea07a4ec/custom_attribute_list"
-username admin -verbose
Password: *****
INFO: Read the resource located at: http://localhost/opr-web/rest/9.10/event_list/0695624b-
93fa-40b1-8b0f-c9b4ea07a4ec/custom_attribute_list

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<custom_attribute_list xmlns="http://www.hp.com/2009/software/opr/data_model"
  self="http://localhost:80/opr-web/rest/9.10/event_list/0695624b-93fa-40b1-8b0f-
c9b4ea07a4ec/custom_attribute_list"
  type="urn:x-hp:2009:software:data_model:opr:type:event:custom_attribute_
list" version="1.0">
  <custom_attribute
    self="http://localhost:80/opr-web/rest/9.10/event_list/0695624b-93fa-40b1-
8b0fc9b4ea07a4ec/custom_attribute_list/CiResolverSimilarityMetric"
    type="urn:x-hp:2009:software:data_model:opr:type:event:custom_attribute" version="1.0">
    <name>CiResolverSimilarityMetric</name>
    <value>100</value>
  </custom_attribute>
</custom_attribute_list>

INFO: Operation successful.
```

Reading Annotations of an Event

To read annotations of an event, you need to specify the complete URL, including the ID of the event you want to read the annotations for.

The following example sends the output to stdout, which is the default.

```
-username admin -verbose
Password: *****
INFO: Read the resource located at:
      http://<fully qualified domain name of OMi gateway server>/opr-web/rest/9.10/event_
```

```
list/10d9a54f-54b9-41d6-b933-ed84b4f5e43e/annotation_list

<?xml version="1.0" encoding="UTF-8"?>
<annotation_list xmlns="http://www.hp.com/2009/software/opr/data_model"
  self="http://mambo.mambo.net:80/opr-web/rest/9.10/event_list/10d9a54f-54b9-41d6-b933-
ed84b4f5e43e/annotation_list"
  type="urn:x-hp:2009:software:data_model:opr:type:event:annotation_list" version="1.0">
  <annotation
    self="http://mambo.mambo.net:80/opr-web/rest/9.10/event_list/10d9a54f-54b9-41d6-b933-
ed84b4f5e43e/annotation_list/74b869a8-399c-42cb-854c-03b9ced975c3"
    type="urn:x-hp:2009:software:data_model:opr:type:event:annotation" version="1.0">
    <id>74b869a8-399c-42cb-854c-03b9ced975c3</id>
    <event_ref
      target_role="urn:x-hp:2009:software:data_model:opr:relationship:event:is_related_
to:event"
      type="urn:x-hp:2009:software:data_model:opr:type:reference:event:event_ref"
version="1.0">
      <target_id>10d9a54f-54b9-41d6-b933-ed84b4f5e43e</target_id>
      <target_type>urn:x-hp:2009:software:data_model:opr:type:event</target_type>
      <target_href>http://mambo.mambo.net:80/opr-web/rest/9.10/event_list/10d9a54f-54b9-41d6-
b933-ed84b4f5e43e</target_href>
    </event_ref>
    <author>admin</author>
    <time_created>2010-07-13T14:18:14.860+02:00</time_created>
    <text>Some annotation text</text>
  </annotation>
</annotation_list>

INFO: Operation successful.
```

Creating a Custom Attribute

Here is an example showing how to create a new custom attribute called MyNewCA for a specified event.

```
RestWsUtil -c newca.xml
-url "http://<fully qualified domain name of OMi gateway server>/opr-web/rest/9.10/event_
list/0695624b-93fa-40b1-8b0fc9b4ea07a4ec/custom_attribute_list"
-username admin -verbose
Password: *****
INFO: Create the resource located at: newca.xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<custom_attribute xmlns="http://www.hp.com/2009/software/opr/data_model"
  self="http://localhost:80/opr-web/rest/9.10/event_list/0695624b-93fa-40b1-8b0f-
c9b4ea07a4ec/custom_attribute_list/MyNewCA"
  type="urn:xhp:2009:software:data_model:opr:type:event:custom_attribute" version="1.0">
  <name>MyNewCA</name>
  <value>100</value>
</custom_attribute>

INFO: Operation successful.
```

The new custom attribute and its value is written to stdout.

The contents of the newca.xml file look like this:

```
<custom_attribute xmlns="http://www.hp.com/2009/software/opr/data_model">
```

```
<name>MyNewCA</name>  
<value>100</value>  
</custom_attribute>
```

Creating an Annotation

Here is an example showing how to create an annotation. The new annotation is written to the newanno.xml file.

```
RestWsUtil -c newanno.xml  
-url "http://<fully qualified domain name of Omi gateway server>/opr-web/rest/9.10/event_  
list/10d9a54f-54b9-41d6-b933-ed84b4f5e43e/annotation_list"  
-username admin -verbose  
Password: *****  
INFO: Create the resource located at: newanno.xml  
  
<?xml version="1.0" encoding="UTF-8"?>  
  
<annotation  
  xmlns="http://www.hp.com/2009/software/opr/data_model"  
  self="http://mambo.mambo.net:80/opr-web/rest/9.10/event_list/10d9a54f-54b9-41d6-b933-  
ed84b4f5e43e/annotation_list/62f86310-38ce-4793-ab3f-23ecfb3f2a67"  
  type="urn:x-hp:2009:software:data_model:opr:type:event:annotation" version="1.0">  
  <id>62f86310-38ce-4793-ab3f-23ecfb3f2a67</id>  
  <event_ref target_role="urn:x-hp:2009:software:data_model:opr:relationship:event:is_  
related_to:event"  
    type="urn:x-hp:2009:software:data_model:opr:type:reference:event:event_ref"  
  version="1.0">  
    <target_id>10d9a54f-54b9-41d6-b933-ed84b4f5e43e</target_id>  
    <target_type>urn:x-hp:2009:software:data_model:opr:type:event</target_type>  
    <target_href>http://mambo.mambo.net:80/opr-web/rest/9.10/event_list/10d9a54f-54b9-41d6-  
b933-ed84b4f5e43e</target_href>  
  </event_ref>  
  <author>admin</author>  
  <time_created>2010-07-13T14:56:56.642+02:00</time_created>  
  <text>Some new annotation text</text>  
</annotation>  
  
INFO: Operation successful.
```

The contents of the newanno.xml file look like this:

```
<annotation xmlns="http://www.hp.com/2009/software/opr/data_model" >  
  <author>admin</author>  
  <text>Some new annotation text</text>  
</annotation>
```

Updating a Custom Attribute

Here is an example showing how to update a custom attribute called MyNewCA with a new value for a specified event.

```
RestWsUtil -update updateca.xml  
-url "http://<fully qualified domain name of Omi gateway server>/opr-web/rest/9.10/event_  
list/0695624b-93fa-40b1-8b0fc9b4ea07a4ec/custom_attribute_list/MyNewCa"
```

```
-username admin -verbose
Password: *****
INFO: Update the resource with changes located at: updateca.xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<custom_attribute xmlns="http://www.hp.com/2009/software/opr/data_model"
  self="http://localhost:80/opr-web/rest/9.10/event_list/0695624b-93fa-40b1-8b0f-
c9b4ea07a4ec/custom_attribute_list/MyNewCA"
  type="urn:x-hp:2009:software:data_model:opr:type:event:custom_attribute" version="1.0">
  <name>MyNewCA</name>
  <value>999</value>
</custom_attribute>

INFO: Operation successful.
```

The updated value for the custom attribute is written to stdout.

The contents of the updateca.xml file look like this:

```
<custom_attribute xmlns="http://www.hp.com/2009/software/opr/data_model">
  <value>999</value>
</custom_attribute>
```

Updating an Annotation

Here is an example showing how to update an annotation. The updated annotation is written to the updateanno.xml file.

```
-url "http://<fully qualified domain name of OMi gateway server>/opr-web/rest/9.10/event_
list/10d9a54f-54b9-41d6-b933-ed84b4f5e43e/annotation_list/582f0488-15ad-40ac-907f-
fec21041b5c0"
-username admin -verbose
Password: *****
INFO: Update the resource with changes located at: updateanno.xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<annotation xmlns="http://www.hp.com/2009/software/opr/data_model"
  self="http://mambo.mambo.net:80/opr-web/rest/9.10/event_list/10d9a54f-54b9-41d6-b933-
ed84b4f5e43e/annotation_list/582f0488-15ad-40ac-907f-fec21041b5c0"
  type="urn:x-hp:2009:software:data_model:opr:type:event:annotation" version="1.0">
  <id>582f0488-15ad-40ac-907f-fec21041b5c0</id>
  <event_ref
    target_role="urn:x-hp:2009:software:data_model:opr:relationship:event:is_related_
to:event"
    type="urn:x-hp:2009:software:data_model:opr:type:reference:event:event_ref"
    version="1.0">
    <target_id>10d9a54f-54b9-41d6-b933-ed84b4f5e43e</target_id>
    <target_type>urn:x-hp:2009:software:data_model:opr:type:event</target_type>
    <target_href>http://mambo.mambo.net:80/opr-web/rest/9.10/event_list/10d9a54f-54b9-41d6-
b933-ed84b4f5e43e</target_href>
  </event_ref>
  <author>admin</author>
  <time_created>2010-07-13T15:56:31.220+02:00</time_created>
  <text>Updated annotation text</text>
</annotation>
```

```
INFO: Operation successful.
```

The contents of the `updateanno.xml` file look like this:

```
<annotation xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <text>Updated annotation text</text>  
</annotation>
```

Updating the Title of an Event

Here is an example showing how to modify the title of an event. The modified title is written to the `update.xml` file.

```
RestWsUtil -update update.xml -username admin  
-url "http://<fully qualified domain name of OMi gateway server>/opr-web/rest/9.10/event_  
list/d36a157e-7312-4302-a2da-e5b7230b0e21"  
Password: *****  
  
INFO: Operation successful.
```

The contents of the `update.xml` file look like this:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <title>New title goes here</title>  
</event>
```

Updating the Lifecycle State of an Event

Here is an example showing how to modify the lifecycle state of an event. This works in the same way as modifying the title of an event. The modified lifecycle state is written to the `update.xml` file.

```
RestWsUtil -update update.xml -username admin  
-url "http://<fully qualified domain name of OMi gateway server>/opr-web/rest/9.10/event_  
list/d36a157e-7312-4302-a2da-e5b7230b0e21"  
Password: *****  
  
INFO: Operation successful.
```

The contents of the `update.xml` file look like this:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <state>in_progress</state>  
</event>
```

Updating the Lifecycle State and the Severity of an Event

Here is an example showing how to modify both the lifecycle state and the severity of an event. This works in the same way as any other updates to an event. The modified lifecycle state and severity is written to the `update.xml` file.

```
RestWsUtil -update update.xml -username admin  
-url "http://<fully qualified domain name of OMi gateway server>/opr-web/rest/9.10/event_  
list/d36a157e-7312-4302-a2da-e5b7230b0e21"
```

```
Password: *****
```

```
INFO: Operation successful.
```

The contents of the `update.xml` file look like this:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <state>in_progress</state>  
  <severity>critical</severity>  
</event>
```

Updating the Event with Transfer Control to Connected Server Information

Here is an example showing how to update an event with the information that control of the event has been transferred to a connected server. This information is written to the `update.xml` file.

```
RestWsUtil -update update.xml -username admin  
-url "http://<fully qualified domain name of OMi gateway server>/opr-web/rest/9.10/event_  
list/d36a157e-7312-4302-a2da-e5b7230b0e21"  
Password: *****  
INFO: Operation successful.
```

The contents of the `update.xml` file look like this:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <control_transferred_to>  
    <name>logger</name>  
  </control_transferred_to>  
</event>
```

Bulk Event Update: Updating the State and the Severity of an Event

Here is an example where all events with the title set to "DB down" have their severity set to `critical` and their state set to `in_progress`. A list of the updated events is returned to the caller.

```
RestWsUtil -update update.xml -username admin  
-url "http://<fully qualified domain name of OMi gateway server>/opr-web/rest/9.10/event_  
list?query=title='DB down'"  
Password: *****  
INFO: Operation successful.
```

The contents of the `update.xml` file look like this:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <state>in_progress</state>  
  <severity>critical</severity>  
</event>
```

Deleting a Custom Attribute

Here is an example of how to delete a custom attribute called `MyNewCA` from the list of custom attributes for a selected event.

```
RestWsUtil -d
```

```
-url "http://<fully qualified domain name of OMi gateway server>/opr-web/rest/9.10/event_
list/0695624b-93fa-40b18b0fc9b4ea07a4ec/custom_attribute_list/MyNewCa" -username admin -
verbose
Password: *****
INFO: Deleting resource located at: http://localhost/opr-web/rest/9.10/event_list/0695624b-
93fa-40b1-8b0f-c9b4ea07a4ec/custom_attribute_list/MyNewCa

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<custom_attribute xmlns="http://www.hp.com/2009/software/opr/data_model"
  self="http://localhost:80/opr-web/rest/9.10/event_list/0695624b-93fa-40b1-8b0f-
c9b4ea07a4ec/custom_attribute_list/MyNewCa"
  type="urn:x-hp:2009:software:data_model:opr:type:event:custom_attribute" version="1.0">
  <name>MyNewCa</name>
  <value>999</value>
</custom_attribute>

INFO: Operation successful.
```

Deleting an Annotation

Here is an example of how to delete an annotation from the list of annotations for a selected event.

```
RestWsUtil -d
-url "http://<fully qualified domain name of OMi gateway server>/opr-web/rest/9.10/event_
list/10d9a54f-54b9-41d6-b933-ed84b4f5e43e/annotation_list/582f0488-15ad-40ac-907f-
fec21041b5c0"
-username admin -verbose
Password: *****
INFO: Deleting resource located at: http://mambo.mambo.net/opr-web/rest/9.10/event_
list/10d9a54f-54b9-41d6-b933-ed84b4f5e43e/annotation_list/582f0488-15ad-40ac-907f-
fec21041b5c0

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<annotation xmlns="http://www.hp.com/2009/software/opr/data_model"
  self="http://mambo.mambo.net:80/opr-web/rest/9.10/event_list/10d9a54f-54b9-41d6-b933-
ed84b4f5e43e/annotation_list/582f0488-15ad-40ac-907f-fec21041b5c0"
  type="urn:x-hp:2009:software:data_model:opr:type:event:annotation" version="1.0">
  <id>582f0488-15ad-40ac-907f-fec21041b5c0</id>
  <event_ref
    target_role="urn:x-hp:2009:software:data_model:opr:relationship:event:is_related_
to:event"
    type="urn:x-hp:2009:software:data_model:opr:type:reference:event:event_ref"
  version="1.0">
    <target_id>10d9a54f-54b9-41d6-b933-ed84b4f5e43e</target_id>
    <target_type>urn:x-hp:2009:software:data_model:opr:type:event</target_type>
    <target_href>http://mambo.mambo.net:80/opr-web/rest/9.10/event_list/10d9a54f-54b9-41d6-
b933-ed84b4f5e43e</target_href>
  </event_ref>
  <author>admin</author>
  <time_created>2010-07-13T15:56:31.220+02:00</time_created>
  <text>Updated annotation text</text>
</annotation>

INFO: Operation successful.
```


Creating an Event

Here is an example of how to create a new event.

```
RestWsUtil -create create.xml -username admin -password ***** -url  
"http://<server.example.com>/opr-web/rest/9.10/event_list"
```

The contents of the create.xml file look like this:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <title>New event title</title>  
  <severity>normal</severity>  
  <priority>low</priority>  
  <state>open</state>  
</event>
```

Creating a List of Events

Here is an example of how to create a list of events.

```
RestWsUtil -create createlist.xml -content_type "application/xml; type=collection" -username  
admin -password ***** -url "http://<server.example.com>/opr-web/rest/9.10/event_list"
```

The contents of the createlist.xml file look like this:

```
<event_list xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <event>  
    <title>Major Event</title>  
    <severity>major</severity>  
  </event>  
  <event>  
    <title>Minor Event</title>  
    <severity>minor</severity>  
  </event>  
</event_list>
```

Note: The HTTP response code from this call will be 202 Accepted. This means that the POST operation looks OK and the request will be submitted, but the resource has not yet been created. New events must be submitted through the event pipeline, and may or may not be eventually stored in the event database. For example, the submitted event may be de-duplicated in the event pipeline. Submission of events through this interface is subject to all the standard restrictions and must be processed through the event pipeline.

Part VII: Integrating External Event Processes

The Event Web Service allows integrators to interface with external incident management applications such as HPE Service Manager or BMC Remedy Service Desk. The web service supports the following functionality:

- Receiving notifications of forwarded events.
- Receiving notifications of event changes.

This section is structured as follows:

- ["Forwarding Events and Synchronizing Event Changes" on page 203](#)
- ["Integrating External Event Processes Using Groovy Scripts" on page 208](#)
- ["Event Synchronization Web Service Interface" on page 227](#)
- ["Integrating External Event Processes: Frequently Asked Questions" on page 228](#)
- ["Integrating an External Event Processing Service Defined by a WSDL" on page 240](#)
- ["Error Handling" on page 244](#)

Chapter 23: Forwarding Events and Synchronizing Event Changes

This section describes how events are forwarded to an external event processing application, such as an incident manager like HPE Service Manager or BMC Remedy Service Desk, and how forwarded events and subsequent event changes are synchronized back from the external application.

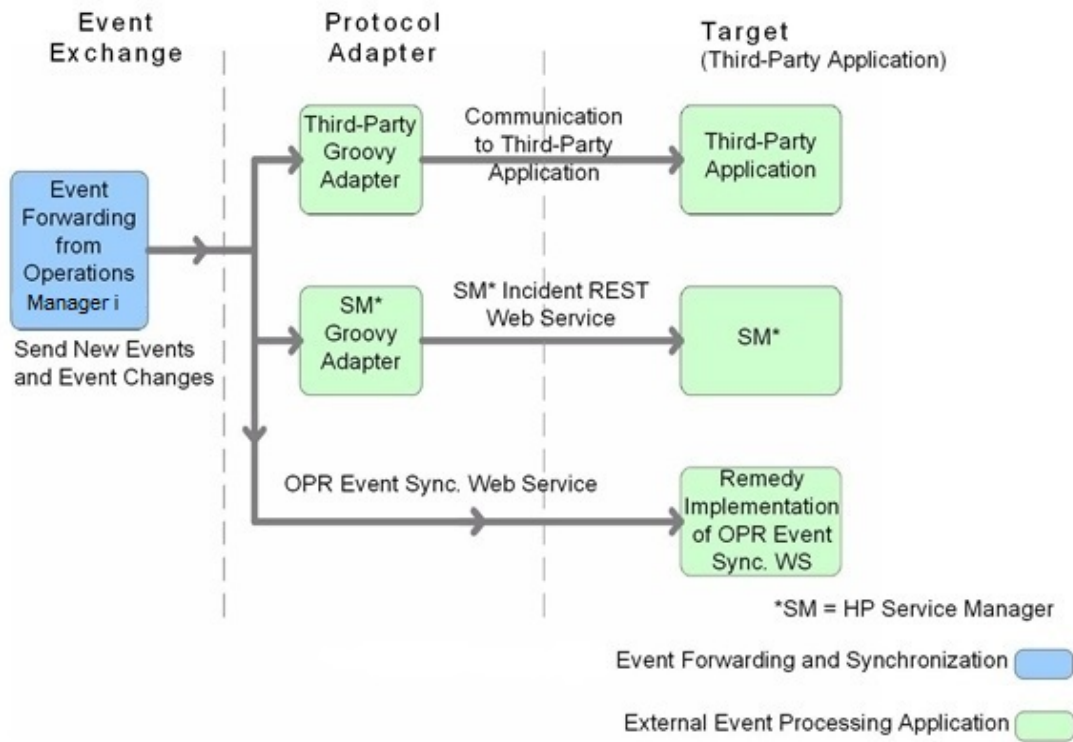
Event Forwarding and Synchronization Process

Synchronizing events and event changes between applications depends on:

- Forwarding events and subsequent changes to an external event process
- Synchronizing event changes back from an external process

Forwarding Events and Event Changes to an External Event Process

The following graphic shows an overview of the event forwarding architecture, and the various routes that the forwarded event can take to reach the incident manager.



The target servers that are to receive the forwarded events and subsequent event changes must be configured using the Connected Servers manager. You can also configure which events to forward based upon a filter, and which connected server to forward the events to. You can configure filters in the Event Forwarding manager.

For details about how to configure connected servers and forwarding rules, see OMi Help.

Events matching the filter are pushed to the target connected server, together with any subsequent changes to the event. The following forwarding modes are supported:

- **Notify** — Target server receives original events but no further updates
- **Notify and Update** — Target server receives original events and all further updates
- **Synchronize** — Target server receives original events and all further updates and sends back all updates
- **Synchronize and Transfer Control** — Target server receives original events and updates and sends back all updates. Ownership of the event is transferred to the other server. Only OMi users with special permission are allowed to close the event after control is transferred, for example, an Administrator.

This option is available only if Enable Synchronize and Transfer Control is enabled on the selected connected server. An operator may manually transfer control via the context menu of the Event Browser.

Delivery of forwarded events and subsequent event changes is guaranteed. If the target connected server is down when the event is forwarded or changes occur, the request is queued and delivered when the target connected server becomes available again.

You can integrate an external event process using the following ways:

Using a Groovy script adapter

You can write and modify Groovy scripts so that you can customize out-of-the-box adapters, and create new adapters. For more information about developing and deploying Groovy scripts, see ["Groovy Scripts" on page 335](#).

The following Groovy script adapters are provided out-of-the-box:

- Service Manager Adapter
- Sample Logfile Adapter (see ["Sample Groovy Script: Logfile Adapter" on page 208](#))

If a Groovy script is configured for the connected server, then a call is made to the Groovy script to forward matching events, and subsequent changes to those events, to the target connected server. The script is so designed that it uses the API that is best suited to deliver the event and event changes to the target connected server. For example, Groovy scripts used for the Service Manager Adapter use the Apache Wink REST client API to execute REST web service calls to HPE Service Manager.

For more details about how to use Groovy scripts for integrations, see ["Integrating External Event Processes Using Groovy Scripts" on page 208](#).

Using the Event Synchronization Web Service

Instead of implementing a Groovy script, an integrator may implement an Event Synchronization Web Service endpoint to receive event forwarding requests and subsequent updates directly from OMi. If the connected server is configured in OMi to call an event REST web service rather than a Groovy script,

then it is also expected that an OPR Event-compliant REST web service is implemented by the target server and available at the connected server endpoint.

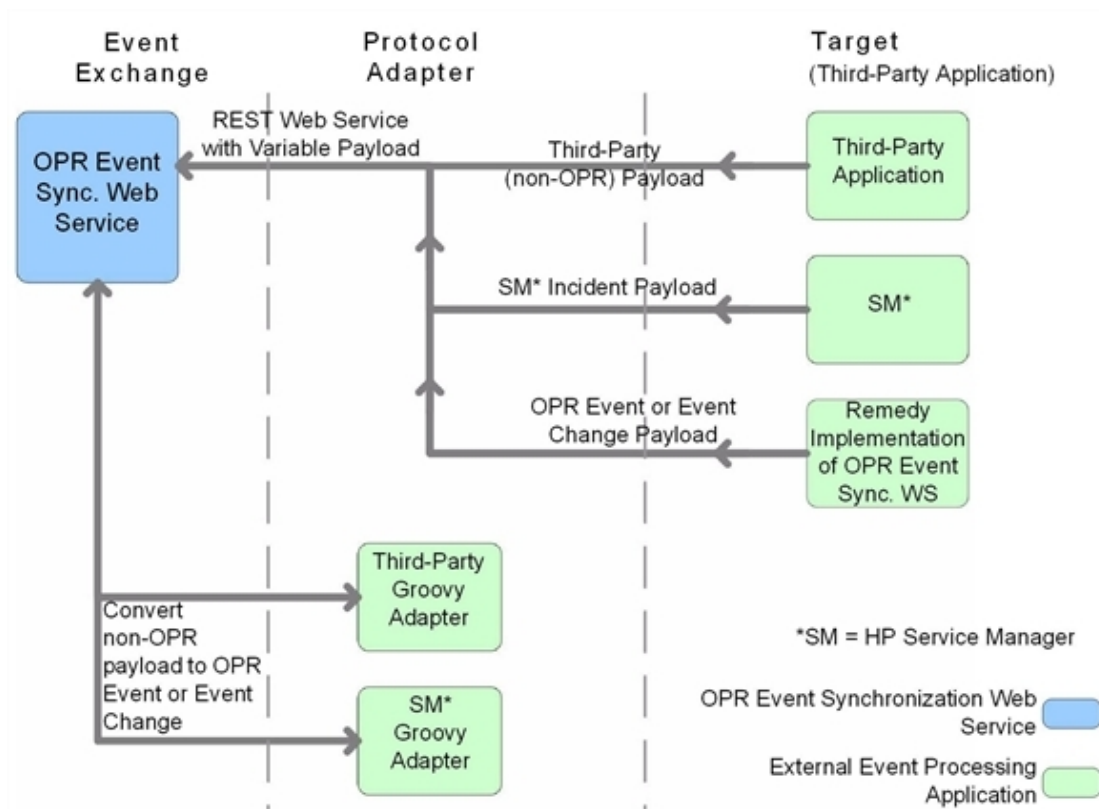
To forward events and their subsequent changes, the following standard REST web service HTTP method calls are made by OMi:

- **POST.** A POST call forwards an event with the payload of an OPR event (the payload is the body of the request). The base URL configured for the target connected server, appended with the `/event` parameter, is used to address the endpoint. The new event created is expected in the response payload.
- **POST.** A POST call forwards a list of events with the payload of an OPR event list. The base URL configured for the target connected server, appended with the `/event` parameter is used to address the endpoint. This endpoint is optional. If provided, the user may select **Supports Bulk Transfer** in the Connected Servers configuration. Each event in the list has a unique `sequence_number`. The expected response payload is an OPR event list containing the events that are created. The corresponding events must have the `sequence_number` set to identify which events were created. Events that are not identified in the response are retried.
- **POST.** A POST call forwards event changes with the payload of an OPR event change. The base URL configured for the target connected server, appended with the `/event_change/<external_event_ID>` parameter, is used to address the endpoint. The expected response payload is an OPR event change object.
- **POST.** A POST call receives bulk event changes with the payload of OPR event change list. This endpoint is optional. It is only required if the option **Supports Bulk Transfer** is selected in the Connected Servers configuration. The base URL configured for the target connected server, appended with the `/event_change` parameter, is used to address the endpoint. The expected response payload is an OPR event change list. Each `OprEventChange` item in the list has an event reference with the event ref with the `target_id` set to the OPR event ID and the `global_target_id` set to the external event ID. Each `OprEventChange` item also has a sequence number set to its sequence in the list. The response must set the sequence number to denote which event changes have been successfully applied. Event changes missing from the list (identified by the corresponding sequence number) are retried.
- **GET.** A GET call is used to get the current state of the external event. The base URL configured for the target connected server, appended with the parameters `/event/<external_event_ID>`, is used to address the endpoint. The expected response payload is an OPR event object.
- **HEAD.** A HEAD call is used to ping the service. This is used by the Connected Servers manager to check the web service credentials specified by the end user. The base URL configured for the target connected server is used to address the endpoint.

For details about how to use the Event Synchronization Web Service, see ["Event Synchronization Web Service Interface" on page 227](#).

Receiving Event Changes Back from an External Event Process

The following graphic shows an overview of how the target application synchronizes changes back to the OPR Event Synchronization Web Service.



When configuring forwarding types for connected servers, you need to consider whether the target server should synchronize back any changes to OMi. If so, you need to specify either the Synchronize forwarding type, or the Synchronize and Transfer Control forwarding type when configuring a forwarding rule.

You can configure a target connected server to support transfer of control. If Supports Synchronize and Transfer of Control is selected during the configuration of the target server, the server is available in the Event Browser context menu, enabling an operator to transfer control of the event to this target connected server. If transfer of control is not selected, then the target server does not appear in the context menu.

If it is expected that the target connected server should synchronize back any changes, the Event Synchronization Web Service is available to receive these changes. You can customize the payload of the web service using a Groovy script adapter, in the same way as you can when configuring event forwarding.

OMi additionally provides the following WS PUT method endpoints for use by external applications to synchronize back external event changes.

PUT. A PUT call is used to bulk update external events. The URL used by OMi to address the endpoint for bulk updating external events is:

`https://gatewayhost/opr-gateway/rest/synchronization/event/`

With a Groovy script configured for the Connected Server, the payload is defined by the Groovy script `receiveChanges()` method.

Without a Groovy script configured for the Connected Server, the payload must be an `OprEventList` object.

For details about how to use the Event Synchronization Web Service, see ["Event Synchronization Web Service Interface" on page 227](#).

Performing a URL Launch of the Event Browser from an External Application

Any operator who needs to drill down into the OMi user interface from an external application using a URL launch of the Event Browser must:

- Be a valid OMi user.
- Have the same user name set up in OMi as the user name of the operator logged on to the calling application and performing the call.

If Single Sign-On (SSO) authentication is configured, set up the operator in OMi with the *same* user name that is used by operator to log onto the calling application and to perform the URL call. (The password of the OMi operator can be empty or any string.) After successfully logging into the calling application, the operator can launch the Event Browser without further authentication.

If the calling application is not configured to use SSO authentication, set up the operator in OMi with the *same* user name that is used by the operator of the calling application and specify a valid password. The operator is required to enter user name and password when launching the Event Browser.

- Have the permission `Events assigned to user` including the required actions granted. You can optionally grant the permission to view events not assigned to the user.

Without this valid user name, or if a user does not have the required viewing rights, an attempt to perform a URL launch of the Event Browser from the calling application results in an empty browser window.

For details about how you would set up a URL launch of the Event Browser, see ["URL Launch of the Event Browser" on page 146](#).

Chapter 24: Integrating External Event Processes Using Groovy Scripts

Groovy scripting is supported. You can use a Groovy script adapter to forward events and event changes to target servers. This section provides information about the main integration points you need to consider when using and developing Groovy scripts for integrations with external event processing applications.

For more information about developing and deploying Groovy scripts, see ["Groovy Scripts" on page 335](#).

You can also find answers to frequently asked questions in the section ["Groovy Scripts and Programming" on page 231](#).

Sample Groovy Script: Logfile Adapter

A sample Groovy script is provided out-of-the-box for use as a template. It is called `LogfileAdapter.groovy` and can be found at:

Administration > Setup and Maintenance > Connected Servers

Click **Manage Scripts**.

You can configure a target connected server in the Connected Servers manager using the Logfile Adapter as the Groovy script adapter. You can use any hostname and port number for the configuration when using the LogfileAdapter, but note that you cannot use the DNS name of the Operations Manager i server. You must use another name, for example, `localhost`. Events and event changes forwarded to this adapter are logged to this log file:

```
<OMi_HOME>/log/opr/integration/LogfileAdapter.log
```

Developers and integrators can use this adapter for testing, and as a template to create other adapters.

A template groovy script for use as a starting point when developing new groovy scripts for external event integration on OMi is available from the following location:

Note: `<OMi_HOME>/opr/examples/external-event-adapter` contains

There are clearly marked sections with "TODO" and steps to take when developing a new integration.

Configure a Connected Server using the Logfile Adapter


Synchronizing events and event changes between an OMi instance and a third-party event process depends upon OMi forwarding events to the external event processing application, with these events and event changes being sent back from the external application. The first step to achieve this is to configure a target connected server in the Connected Servers manager.

For full details about how to configure a connected server, see OMi Help.

To configure a target connected server using the Logfile Adapter as Groovy script adapter, perform the following steps:

1. Navigate to the Connected Servers manager:

Administration > Setup and Maintenance > Connected Servers

2. Click the New  button to open the Create New Server Connection dialog box.
3. In the **Display Name** field, enter a name for the target connected server. The Name field is filled automatically. For example, if you enter `Logger_Example` as the Display Name for the target HPE Service Manager server, `Logger_Example` is automatically inserted in the Name field.

Note: Make a note of the name of the new target server (in this example, `Logger_Example`). You need to provide it later on as the username when configuring the HPE Service Manager server to communicate with the server hosting OMi.

Optional: Enter a description for the new target server.

Make sure that you check the Active check box.

Click **Next**.

4. Select **External Event Processing** to choose the server type suitable for an external event processing application.

Click **Next**.

5. Enter the Fully Qualified DNS Name of the Logfile target server, for example, `localhost`

Click **Next**.

6. Next, you need to establish the type of integration. In the Integration Type dialog box, you can choose between using a Groovy script adapter, or the Event Synchronization Web Service. In this example, we want to choose the a Groovy script adapter.

- a. Select **Call Groovy Script Adapter**.

- b. In the External Event Processing Type field, select **sample**.

- c. In the Groovy Script File Name field, select **LogfileAdapter.groovy**.

(Leave the Groovy Classpath field blank, as in this case, no external resources are required.)

- d. Click **Next**.

7. The Outgoing Connection dialog box is for providing the credentials (user name, password, port number) to enable connection to the target server, and to forward events to that server. In the case of the Logfile Adapter, you do not need to provide the user name, password, or port number, and you do not have to choose the HTTP setting. You can leave these fields blank for this exercise.

In the Outgoing Connection dialog box, do the following:

- a. Make sure that the Enable Synchronize and Transfer Control checkbox is checked. When the Synchronize and Transfer Control flag is set, an OMi operator is then able to transfer ownership of the event to the target connected server. If the flag is not set, then the option Synchronize and Transfer Control does not appear in the list of forwarding types when configuring forwarding rules.

- b. Click **Next**.

8. The Event Drilldown dialog opens. If, in addition to purely forwarding events to the target connected server, you also want to be able to drill-down into the external event processing application, you need to specify the fully qualified DNS name, and port of the OMi system where

you want to perform event drill-down. For this example, enter the following:

- Fully qualified DNS name: **test.host.com**
 - Port: **80**
 - Click **Next**.
9. The next thing to do is to enable event changes to be received back from the connected server. For this you need to provide credentials for the connected server to access the server hosting OMi.
- a. In the Incoming Connection dialog box, enter a password that the external application requires to connect to the server hosting OMi, `HPpasswd1_` in this example.
 - b. Click **Finish**.




The target Logger Example server appears in the list of Connected Servers.

Configure an Event Forwarding Rule

The next step is to configure an event forwarding rule that determines which events are forwarded to the `Logger Example` server.

See OMi Help for full details about configuring filters.

To configure a forwarding rule, carry out the following steps:

1. Navigate to the Event Forwarding manager:
Administration > Event Processing > Automation > Event Forwarding
2. Click the New  button to open the Create New Forwarding Rule dialog box.
3. In the Display Name field, enter a name for the forwarding rule, in this example `Forward Major (Sync and Transfer Control)`.
Optional. Enter a description for the forwarding rule you are creating.
Make sure the Active check box is checked. A rule must be active in order for its status to be available in the target connected server.
4. Click the browse button next to the Event Filter field. The Select an Event Filter dialog opens.
5. In the Select an Event Filter dialog, click the New  button to open the Filter Configuration dialog.
6. In the Filter Display Name field, enter a name for the new filter, in this example, `FilterMajor`.
Deselect the checkboxes for all severity levels except for the severity Major.
Click **OK**.
7. You should see your new filter in the Select an Event Filter dialog (select it, if it is not already highlighted).
Click **OK**.
8. Under Target Servers, select the connected target server you configured in the section "[Sample Groovy Script: Logfile Adapter](#)" on page 208. In this example, this is `Logger Example`.
Click the Add  button next to the target servers selection field. You can now see the

connected server's details.

Click **OK**.

The new forwarding rule is now available.

Groovy Script Interface

If you intend to use Groovy scripts to integrate with external event processes, such as an integration with an incident manager, then you must implement a Groovy script that implements the methods defined by the following interface:

```
com.hp.opr.api.ws.adapter.ExternalProcessAdapter
```

This is provided in the JAR file `opr-external-api.jar`.

The Groovy script interface with full documentation of all arguments and types can be found in the Javadoc documentation delivered with the product. For the location of the Javadoc API Documentation, see "[Javadoc API Documentation](#)" below. For more information about developing and deploying Groovy scripts, see "[Groovy Scripts](#)" on page 335.

API Library

The API library contains Java Architecture for XML Binding (JAXB) annotated classes for the OPR Event and Event Change objects. If you are programming in Groovy or Java, you can use these classes directly. Otherwise, you may need to use the OPR Event Schema (see "[OPR Event Schema](#)" below).

You can find the API library in the following installation location:

```
<OMi_HOME>/lib/opr-external-api.jar
```

Javadoc API Documentation

The external API interface is documented in detail in the Javadoc API Documentation. You can find it in the following installation location:

```
<OMi_HOME>/opr/api/doc/opr-external-api-javadoc.zip
```

OPR Event Schema

If you are using a programming language other than Java, you may need to use the OPR Event Schema to generate classes to marshal and unmarshal Event and Event Change objects.


You can find the OPR Event Schema in the following installation location:

```
<OMi_HOME>/opr/api/schema/OprDataModel.xsd
```

Event Integration Groovy Scripts

Event integration Groovy scripts are stored in the database and accessed from the following location:

Administration > Setup and Maintenance > Connected Servers

In the Connected Servers pane, click the  button to open the Event Forwarding Scripts Configuration dialog box from which you can select an existing script or create a new one.

The out-of-the-box scripts are:

Logfile Adapter, type: `sample:LogfileAdapter`

Service Manager Adapter, type: `sm:ServiceManagerAdapter`

Upgrade Adapter, type: `upgrade:UpgradeAdapter`

Groovy Script Methods

If you are using a Groovy script to integrate external event processes, that Groovy script must implement the methods defined by the `ExternalProcessAdapter` interface.

The methods that must be implemented by the Groovy script are listed in this section.

- [init, destroy, and ping Methods](#) 212
- [Methods for Forwarding Events to a Connected Server](#) 214
- [Methods for Receiving Synchronization Data from a Connected Server](#) 220
- [Additional Methods](#) 222
- [Methods for Retrieving Data from a Connected Server](#) 224
- [Method for Supplying Data to a Connected Server](#) 225
- [Management of Groovy Script Methods](#) 226

init, destroy, and ping Methods

The methods for initializing and closing the Groovy script, and verifying the Groovy script parameters are listed in this section.

init

The `init(def args)` method is called when the Groovy script is first loaded. The loaded script is cached and called many times. It is only reloaded at startup and whenever the script or the configuration is changed in the Connected Servers manager. A separate instance is created for each connected server that uses this script.

The `init(def args)` method has one argument, and the properties listed in "[Properties for init\(\) Method](#)" below.

Properties for init() Method

Property	Description
Read Properties	
<code>installDir</code>	OMi root directory.

Property	Description
logger	Type <code>org.apache.commons.logging.Log</code> . Used to access the logs.
connectedServerId	ID of the target connected server.
connectedServerName	Name of the target connected server.
connectedServerDisplayName	Display Name of the target connected server.
node	DNS name of the target connected server.
port	Type Integer. Port number of the web service, if any, for the target connected server.
nodeSsl	Type Boolean. Indicates if the target connected server web service requires TLS.
drilldownNode	DNS name of the target connected server where drill-down may be done.
drilldownPort	Type Integer. Drill-down port, if any, for the drilldown service on the target connected server drill-down node.
drilldownNodeSsl	Type Boolean. Indicates if the drill-down service requires TLS.
maxTimeout	Number of milliseconds to use for timeout on connections. You configure this in the Connected Servers manager.
Write Properties	
None	There are no write properties for this method.

destroy

The `destroy()` method is called when the Groovy script is no longer needed. It has no arguments.

ping

The `ping(def args)` method is called by the Connected Servers manager to determine whether the configuration parameters are correct. If the connected server is reachable using the specified read properties, the `ping` method returns `true`, otherwise it returns `false`.

"[Properties for ping\(\) Method](#)" below lists the properties for the `ping()` method:

Properties for ping() Method

Property	Description
Read Properties	
installDir	OMi root directory.

Property	Description
logger	Type <code>org.apache.commons.logging.Log</code> . Used to access the logs.
connectedServerName	Name of the target connected server.
connectedServerDisplayName	Display Name of the target connected server.
node	DNS name of the target connected server.
port	Type Integer. Port number of the web service, if any, for the target connected server.
nodeSsl	Type Boolean. Indicates if the target connected server web service requires TLS.
credentials	Type <code>java.net.PasswordAuthentication</code> Credentials, if any, for the target connected server.
locale	Type: Locale The desired locale of the properties that are set.
Write Properties	
outputDetail	Detailed results of the ping operation. This is displayed in the Connected Servers configuration dialog box, and should be in the desired locale.

Methods for Forwarding Events to a Connected Server

The methods for forwarding events to an external event processing application, configured as a target connected server, are listed in this section.

forwardEvent

The `forwardEvent(def args)` method is called to forward the event to the target connected server. This method has one argument, with the properties listed in "[Properties for forwardEvent\(\) Method](#)" below:

Properties for forwardEvent() Method

Property	Description
Read Properties	

Property	Description
credentials	Type: java.net.PasswordAuthentication Credentials, if any, for the target connected server.
event	Type: com.hp.opr.api.ws.model.event.OprEvent This is the event to be forwarded.
info	Type: com.hp.opr.api.ws.model.event.ci.OprForwardingInfo Holds information regarding the type of forwarding, that is: Notify, Notify and Update, Synchronize, or Synchronize and Transfer Control.
causeExternalRefId	If this is a symptom, the cause is set, and the cause has been forwarded to the target server, this property holds the external reference ID of the cause, otherwise null is returned.
Read Methods	
credentials	Type: java.net.PasswordAuthentication Credentials, if any, for the target connected server.
getEvent(id, includeRefs)	Enables getting a specified event. If the event exists and the application has access to it, it is returned, otherwise null is returned. If the caller wishes the references to be resolved, set includeRefs to true, otherwise false. These references include history lines, all properties of related CI, and event relations such as symptom events.
getEvent(id, includeRefs, includeAnnotations, includeCis)	Enables getting a specified event. If the event exists and the application has access to it, it is returned, otherwise null is returned. If the caller wishes the references to be resolved, set includeRefs to true, otherwise false. These references include history lines, all properties of related CI, and event relations such as symptom events. Set the includeAnnotations flag to false to exclude annotations. Set the includeCis flag to false to avoid pulling related CI information from the RTSM. This will also help to improve forwarding performance.
getCi(id)	Obtains details of a configuration item, including all properties. Arguments are of type String. The method returns the specified configuration item, including all properties. Return value is of type OprConfigurationItem.
Write Properties	
drilldownUrlPath	<i>Optional.</i> If set, enables drill-down into the external application in the Event Browser. This is the base path of the URL, and does not include node or port

Property	Description
externalRefId	Must be set if forward is successful. Can be any string value.

forwardEvents

The `forwardEvents(def args)` method is called to forward a list of events to the target connected server. This method is passed a single argument, that contains a list of events to forward, as well as some utility methods. There is also a method to set the forward result of each event in the list. If the forward result is not set for one of the events it will remain in the forwarding queue and will be retried on the next call to the groovy script. The result for each individual event may be set to success or failed. This method is optional.

The method `forwardEvent()` has utility methods to enable the groovy script to easily obtain the `OprForwardingInfo`. `BulkForwardEventArgs` does not directly provide this utility method. To obtain the `OprForwardingInfo` for each event, the you can call the method `getForwardingInfo(final String serverId)` on each event. The `serverId` is the ID of this connected server and was passed to the groovy script in the `InitArgs`.

This method has one argument (`BulkForwardEventArgs`), with the properties listed in "[Properties for forwardEvents\(\) Method](#)" below:

Properties for forwardEvents() Method

Property	Description
Read Properties	
credentials	Type: <code>java.net.PasswordAuthentication</code> Credentials, if any, for the target connected server.
events	Type: <code>com.hp.opr.api.ws.model.event.OprEventList</code> This is the list of events to be forwarded.
Read Methods	
<code>getEvent(id, includeRefs)</code>	Enables getting a specified event. If the event exists and the application has access to it, it is returned, otherwise null is returned. If the caller wishes the references to be resolved, set <code>includeRefs</code> to true, otherwise false. These references include history lines, all properties of related CI, and event relations such as symptom events.

Property	Description
<code>getEvent(id, includeRefs, includeAnnotations, includeCis)</code>	Enables getting a specified event. If the event exists and the application has access to it, it is returned, otherwise null is returned. If the caller wishes the references to be resolved, set <code>includeRefs</code> to true, otherwise false. These references include history lines, all properties of related CI, and event relations such as symptom events. Set the <code>includeAnnotations</code> flag to false to exclude annotations. Set the <code>includeCis</code> flag to false to avoid pulling related CI information from the RTSM. This will also help to improve forwarding performance.
<code>getCi(id)</code>	Obtains details of a configuration item, including all properties. Arguments are of type String. The method returns the specified configuration item, including all properties. Return value is of type <code>OprConfigurationItem</code> .
<code>getCauseExternalRefId(symptomId)</code>	Obtains an external reference ID. Arguments are of type String. The method returns the <code>cause external reference ID</code> , if any exists for the specified symptom.
Write Methods	
<code>setForwardSuccess(eventId, externalRefId, crossLaunchUrlPath)</code>	This method enables setting the success for each individual event, along with the external reference ID and any cross launch URL for that external event.
<code>setForwardFailed(reason)</code>	If the bulk forward fails, this method can be called, passing a <code>Throwable</code> which describes the reason for the failure to deliver. The reason may be null. The entire list is marked as failed and not retried.
<code>setForwardFailed(eventId, reason)</code>	Sets an individual event in the list to failed. This event is not retried. An optional reason may be specified or set to null. If neither <code>setSuccess()</code> or <code>setFailed()</code> is called for a particular event, the event is retried later.

forwardChange

The `forwardChange(def args)` method is called to forward the event changes to the target connected server. This method has one argument, with the properties listed in "[Properties for forwardChange\(\) Method](#)" below:

Properties for forwardChange() Method

Property	Description
Read Properties	

Property	Description
changes	Type: com.hp.opr.api.ws.model.event.OprEventChange The event changes to be forwarded.
credentials	Type: java.net.PasswordAuthentication Credentials, if any, for the target connected server.
externalRefId	Type: String The ID of the external event that should be updated.
info	Type: com.hp.opr.api.ws.model.event.ci.OprForwardingInfo Holds information regarding the type of forwarding, that is: Notify, Notify and Update, Synchronize, or Synchronize and Transfer Control.
event	Returns a copy of the event in its current state. The attributes of the event reflect the current state of the event at the time this call was made, and not the state when the change occurred.
causeExternalRefId	If this is a symptom, the cause is set, and the cause has been forwarded to the target server, this property holds the external reference ID of the cause, otherwise null is returned.
Read Methods	
getEvent(id, includeRefs)	Enables getting a specified event. If the event exists and the application has access to it, it is returned, otherwise null is returned. If the caller wishes the references to be resolved, set includeRefs to true, otherwise false. These references include history lines, all properties of related CI, and event relations such as symptom events.
getEvent(id, includeRefs, includeAnnotations, includeCis)	Enables getting a specified event. If the event exists and the application has access to it, it is returned, otherwise null is returned. If the caller wishes the references to be resolved, set includeRefs to true, otherwise false. These references include history lines, all properties of related CI, and event relations such as symptom events. Set the includeAnnotations flag to false to exclude annotations. Set the includeCis flag to false to avoid pulling related CI information from the RTSM. This will also help to improve forwarding performance.
getCi(id)	Obtains details of a configuration item, including all properties. Arguments are of type String. The method returns the specified configuration item, including all properties. Return value is of type OprConfigurationItem.
Write Properties	
None	There are no write properties for this method.

forwardChanges

The `forwardChanges(def args)` method is called to forward the event changes to the target connected server. This method is passed a single argument, that contains a list of event changes to forward, as well as some utility methods. There is also a set of methods to set the forward result of each event change item in the list. If the forward result is not set for one of the event changes it will remain in the forwarding queue and will be retried on the next call. The result for each individual event change may be set to success or failed. This method is optional.

This method has one argument (`BulkForwardChangeArgs`), with the properties listed in ["Properties for forwardChanges\(\) Method"](#) below:

Properties for forwardChanges() Method

Property	Description
Read Properties	
changes	Type: com.hp.opr.api.ws.model.event.OprEventChangeList The event changes to be forwarded.
credentials	Type: java.net.PasswordAuthentication Credentials, if any, for the target connected server.
Read Methods	
getCi(id)	Obtains details of a configuration item, including all properties. Arguments are of type String. The method returns the specified configuration item, including all properties. Return value is of type <code>OprForwardingInfo</code> .
getEvent(id, includeRefs)	Enables getting a specified event. If the event exists and the application has access to it, it is returned, otherwise null is returned. If the caller wishes the references to be resolved, set <code>includeRefs</code> to true, otherwise false. These references include history lines, all properties of related CI, and event relations such as symptom events.
getEvent(id, includeRefs, includeAnnotations, includeCis)	Enables getting a specified event. If the event exists and the application has access to it, it is returned, otherwise null is returned. If the caller wishes the references to be resolved, set <code>includeRefs</code> to true, otherwise false. These references include history lines, all properties of related CI, and event relations such as symptom events. Set the <code>includeAnnotations</code> flag to false to exclude annotations. Set the <code>includeCis</code> flag to false to avoid pulling related CI information from the RTSM. This will also help to improve forwarding performance.
getCauseExternalRefId(symptomId)	Obtains an external reference ID. Arguments are of type String. The method returns the <code>cause external</code> reference ID, if any exists for the specified symptom.

Property	Description
OprForwardingInfo getForwardingInfo (change)	Obtains details of the forwarding information, including all properties. Arguments are of type String. This method gets the OprForwardingInfo for the specified event change. The external reference ID can be obtained from this forwarding info.
Write Methods	
setForwardSuccess()	Sets success for all changes in the list.
setForwardSuccess (changesId)	Sets success for the specified change in the list.
setForwardFailed (reason)	Sets all changes in the list to failed. A reason may be specified in the form of an Exception, or null may be specified. If set to failed, the changes are not retried.
setForwardFailed (changeId, reason)	Set the specified change to failed. This change is not retried. If neither setSuccess() or setFailed() is called for a particular change, the change is retried later.

Methods for Receiving Synchronization Data from a Connected Server

The methods for receiving synchronization data from an external event processing application, configured as a target connected server, are listed in this section.

Note: These methods are only needed if the connected server supports the synchronizing back of event changes from the connected server.

receiveChange

The `receiveChange(def args)` method is called when the connected server sends an event change to the Event Synchronization Web Service. This method has one argument with the properties listed in ["Properties for receiveChange\(\) Method"](#) below:

Properties for receiveChange() Method

Property	Description
Read Properties	
event	Type: OprEvent A read only copy of the event in its current state, before any changes are applied. This is for read-only. To update the event set one of the writable properties of the arguments.

Property	Description
externalEventChange	Type: String The payload of the web service call (PUT or POST) received from the connected server when a change is synchronized back from the connected server.
info	Type: com.hp.opr.api.ws.model.event.ci.OprForwardingInfo Holds information regarding the type of forwarding, that is: Notify, Notify and Update, Synchronize, or Synchronize and Transfer Control.
locale	Type: Locale The desired locale of the web service caller.
externalRefId	Type: String The ID of the external event that should be updated.
causeExternalRefId	If this is a symptom, the cause is set, and the cause has been forwarded to the target server, this property holds the external reference ID of the cause, otherwise null is returned.
Read Methods	
getEvent(id, includeRefs)	Enables getting a specified event. If the event exists and the application has access to it, it is returned, otherwise null is returned. If the caller wishes the references to be resolved, set includeRefs to true, otherwise false. These references include history lines, all properties of related CI, and event relations such as symptom events.
getEvent(id, includeRefs, includeAnnotations, includeCis)	Enables getting a specified event. If the event exists and the application has access to it, it is returned, otherwise null is returned. If the caller wishes the references to be resolved, set includeRefs to true, otherwise false. These references include history lines, all properties of related CI, and event relations such as symptom events. Set the includeAnnotations flag to false to exclude annotations. Set the includeCis flag to false to avoid pulling related CI information from the RTSM. This will also help to improve forwarding performance.
getCi(id)	Obtains details of a configuration item, including all properties. Arguments are of type String. The method returns the specified configuration item, including all properties. Return value is of type OprConfigurationItem.
Write Properties	
eventId	ID of the event.
title	Title of the event.

Property	Description
description	Description of the event.
solution	Solution of the event.
severity	Severity of the event.
priority	Priority of the event.
state	State of the event. If set to <code>closed</code> or <code>resolved</code> , the state of the event is set to <code>closed</code> . Setting the event to any other value sets the state of the event to <code>open</code> .
cause	ID of the cause event.
assignedUser	Type: <code>com.hp.opr.api.ws.model.event.OprUser</code> Name of the user who is responsible for solving the event's underlying problem.
assignedGroup	Type: <code>com.hp.opr.api.ws.model.event.OprGroup</code> Name of the group to which the event's assigned user belongs.

Additional Methods

Additional methods are provided for transferring the control of an event, annotations, custom attributes, configuration items, and for accessing the HTTP requests and responses.

Control Transfer

The methods available for transferring control of an event are listed here.

- `requestControl`
The `requestControl` method is a request to take control of the event. Once a request has been made, it is not possible for the event to be owned by another server. The request is queued, and when completed, the caller receives a change notification for the `control_transferred_to` event property.
- `returnControl`
The `returnControl` method returns the control of the event back from the external event process to OMi. The caller must have control of the event in order to return control. The request is queued, and when completed, the caller receives a change notification for the `control_transferred_to` event property.

Annotations

The methods available for annotations are listed here. Annotations may be added or updated, but not

deleted.

- `addAnnotation(def text, def author)`
The `addAnnotation(def text, def author)` method adds an annotation. Arguments are of type `String`.
- `updateAnnotations(def id, def text, def author)`
The `updateAnnotation(def id, def text, def author)` method updates an annotation. Arguments are of type `String`.

Custom Attributes

The methods available for custom attributes are listed here.

- `addCustomAttribute(def name, def value)`
The `addCustomAttribute(def name, def value)` method adds a custom attribute. Arguments are of type `String`.
- `updateCustomAttribute(def name, def value)`
The `updateCustomAttribute(def name, def value)` method updates a custom attribute. Arguments are of type `String`.

HTTP Requests and Responses

Three methods are provided for accessing the HTTP request or response.

- `getHttpRequestHeader(def name)`
The `getHttpRequestHeader(def name)` method returns a list of header values. Arguments are of type `String`.
- `setHttpResponseStatus(def httpResponseCode, def httpResponseMessage)`
The `setHttpResponseStatus(def httpResponseCode, def httpResponseMessage)` method can be called to control the response status and payload. Default is 202 and no payload. The code is of type `Integer`, message is of type `String`.
- `setHttpResponseHeader(def name, def value)`
The `setHttpResponseHeader(def name, def value)` method enables setting of the specified HTTP response header. Arguments are of type `String`.

receiveChanges

The `receiveChanges(def args)` method is called when the connected server sends multiple event changes to the Event Synchronization Web Service. This method is passed a single argument, that contains the payload sent by the external processing server to the Event Synchronization WS endpoint `/opr-gateway/rest/9.10/event_change`, as well as some utility methods. It is expected the payload contains a list of changes. It is up to the groovy script method to interpret the list of changes and apply them to an event. This is done through a utility method to create a `ReceiveChangeArgs` object for each event that is to be changed and setting the changes in that object.

This method has one argument (`BulkReceiveChangeArgs`) with the properties listed in "[Properties for receiveChanges\(\) Method](#)" below:

Properties for receiveChanges() Method

Property	Description
Read Properties	
externalEventChanges	Type: String The payload of the web service call (PUT or POST) received from the connected server when one or more changes are synchronized back from the connected server.
locale	Type: Locale The desired locale of the web service caller.
Read Methods	
getEvent(id, includeRefs)	Enables getting a specified event. If the event exists and the application has access to it, it is returned, otherwise null is returned. If the caller wishes the references to be resolved, set <code>includeRefs</code> to true, otherwise false. These references include history lines, all properties of related CI, and event relations such as symptom events.
getEvent(id, includeRefs, includeAnnotations, includeCis)	Enables getting a specified event. If the event exists and the application has access to it, it is returned, otherwise null is returned. If the caller wishes the references to be resolved, set <code>includeRefs</code> to true, otherwise false. These references include history lines, all properties of related CI, and event relations such as symptom events. Set the <code>includeAnnotations</code> flag to false to exclude annotations. Set the <code>includeCis</code> flag to false to avoid pulling related CI information from the RTSM. This will also help to improve forwarding performance.
getCi(id)	Obtains details of a configuration item, including all properties. Arguments are of type String. The method returns the specified configuration item, including all properties. Return value is of type <code>OprConfigurationItem</code> .
getHttpRequestHeader(name)	Obtains the specified header. Header may be specified more than once. All instances specified are returned.
getReceiveChangeArgs(id)	Gets the <code>ReceiveChangeArgs</code> object for the specified event. It is automatically added to the list of events to be changed. The methods provided by <code>ReceiveChangeArgs</code> can be called to apply the appropriate changes to the specified event.

Methods for Retrieving Data from a Connected Server

The methods for retrieving data from a connected server are listed in this section.

getExternalEvent

The `getExternalEvent()` method is called when the operator requests the current representation of the external event. When the operator opens the Event Details in the Event Browser and selects the

External Info tab, or uses the context menu in the Event Browser, the `getExternalEvent()` method is called to get the latest copy of the external event, and display selected fields in the Event Browser. This method has one argument with the properties listed in ["Properties for getExternalEvent\(\) Method" below](#):

Properties for `getExternalEvent()` Method

Property	Description
Read Properties	
<code>externalRefId</code>	The external reference ID, identifying the external event to get.
<code>credentials</code>	Type: <code>java.net.PasswordAuthentication</code> Credentials, if any, for the target connected server.
<code>locale</code>	Type: <code>Locale</code> The desired locale of the properties that are set.
Write Properties (All are of type String)	
<code>title</code>	<i>Optional.</i> Title of the external event.
<code>description</code>	<i>Optional.</i> Description of the external event.
<code>assignedUser</code>	<i>Optional.</i> User assigned to the external event.
<code>assignedGroup</code>	<i>Optional.</i> User group assigned to the external event.
<code>severity</code>	<i>Optional.</i> Severity of the event.
<code>priority</code>	<i>Optional.</i> Priority of the event.
<code>state</code>	<i>Optional.</i> State of the event.

Method for Supplying Data to a Connected Server

There is also a method for supplying data to a target connected server.

`toExternalEvent`

The `toExternalEvent()` method converts an OPR Event into an external event object. This method is called when a connected server queries the Event Synchronization Web Service for the latest copy of the OPR Event. This enables an integrator to convert the event to an external event representation, that is, to create the response payload of the GET from `(.../event/<event id>)`.

This method has one input parameter: `com.hp.opr.api.ws.model.event.OprEvent`

The return value of this method is a string. This is the payload that is returned to the application that called the web service.

Management of Groovy Script Methods

The Groovy script methods run under a separate Java Security Manager. The following permissions are denied by the Java Security Manager:

- access to the `System.exit()` call
- `java.lang.RuntimePermission` only for target `exitVM`:
<http://java.sun.com/javase/6/docs/api/java/lang/RuntimePermission.html>

Chapter 25: Event Synchronization Web Service Interface

The Event Synchronization Web Service interface can be used to forward events and event changes from the OMi OPR client to the third-party application, and to synchronize back from the third-party client events and event changes that the external application makes to the events.

For detailed reference information about the web service, see ["Event Synchronization Web Service Interface" on page 275](#).

Chapter 26: Integrating External Event Processes: Frequently Asked Questions

This section contains a collection of frequently asked questions (FAQs) related to integrating external event processes. The FAQs are intended to assist developers writing scripts for a connected server configured for external event processing.

This section is structured as follows:

- [Getting Started](#) 228
- [Groovy Scripts and Programming](#) 231
- [Integration Script Methods](#) 233
- [Event Properties](#) 237
- [Troubleshooting](#) 238
- [Logging](#) 239

Getting Started

This section contains FAQs related to basic information required to get started.

Is there any documentation available?

Yes. Read this guide.

- For information about the event forwarding and synchronization process, see "[Forwarding Events and Synchronizing Event Changes](#)" on page 203.
- For information about the Event Synchronization Web Service, see "[Event Synchronization Web Service Interface](#)" on page 227.
- For information about integrations using Groovy scripts, see "[Integrating External Event Processes Using Groovy Scripts](#)" on page 208.
- For an introduction to using Groovy scripts, see "[Groovy Scripts](#)" on page 335.

Is there any Javadoc for the API?

Yes. You can find the Javadoc here:

`<OMi_HOME>/opr/api/doc/opr-external-api-javadoc.zip`

You need to unzip this file.

Is there a schema available for the OprEvent and OprEventChange object?

Yes. You can find the schema here:

`<OMi_HOME>/opr/api/schema/OprDataModel.xsd`

Will I need a schema?

If you integrate an external application using a Groovy script, you most likely do *not* need the `OprEvent` schema. The schema is mainly needed if you intend your integration to communicate directly with the Event Synchronization Web Service endpoints using `OprEvent` and `OprEventChange` objects with no intervention by a Groovy script. For example, if you implement a REST web service that will accept the HTTP POST method call of an `OprEvent` object (as described in the schema), and a REST client that either performs HTTP POST methods calls of `OprEventChange` objects, or performs HTTP PUT method calls for `OprEvent` objects to submit those objects with changes to the Event Synchronization Web Service. In this case, the connected server would be configured for a web service integration and not a Groovy script integration.

Additionally, if your client or service is written in Java, you can directly use the Java Architecture XML Binding (JAXB) annotated objects in the `opr-external-api.jar` file to marshal and unmarshal the XML. Only if you are programming in a different language should you need to use the schema directly.

Is there a Web Services Description Language (WSDL) available?

No. The Event Synchronization Web Service is a simple REST-based web service. In general, REST-based web services do not have a WSDL.

Where can I find out more about REST?

A good place to start is the following Wikipedia entry:

http://en.wikipedia.org/wiki/Representational_State_Transfer

Are there any good toolkits to use for creating a REST client or server?

Use the Apache Wink toolkit, which you can get here:


<http://incubator.apache.org/wink/index.html>

Are there JAXB annotated classes already available for `OprEvent` and `OprEventChange`?

Yes. The `<OMi_HOME>/lib/opr-external-api.jar` file contains JAXB annotated classes. If you are programming in Java, you can use these classes directly instead of generating classes from the schema. See the Javadoc API Documentation for details about the classes provided.

Are there any sample implementations?

Yes. Take a look at the `LogfileAdatper` sample located here:

1. Open the Connected Servers manager:
Administration > Setup and Maintenance > Connected Servers
2. In the Connected Servers pane, click the  button to open the Event Forwarding Scripts Configuration dialog box.
3. Open the `sample.LogfileAdatper` Groovy script.

This is a sample Groovy script adapter that accepts forward requests and writes them to a log file.

How do I configure and use the Logfile Adapter sample?

When configuring this adapter, configure the connected server with the following parameters:

- node - localhost (do not use the DNS name here)
- port - 80 for standard http, 443 if TLS is enabled
- drilldown node - *<DNS name of your OMi node>*
- drilldown port - 80 for standard http, 443 if TLS is enabled
- **Enable Synchronize and Transfer Control** is selected

After you configure the connected server, you can forward events manually from the event browser to the Logfile Adapter. Just select **Transfer Control to ...** from the Event Browser context menu. You can also configure an automatic forwarding rule using the Event Forwarding manager.

Once an event has been forwarded, you should see a new tab, called External Info, in the Event Details of the for the forwarded event. Selecting this tab calls the `sample.LogfileAdapter` script `getExternalInfo()` method to fetch the external information for this event and then display it in the tab. If the call succeeds, the fields on the left will be populated with data.

I wrote a client to access the Event Synchronization Web Service to update an event, but I always receive an HTTP 401, Unauthorized error message. How do I authenticate?

Make sure you have configured a password for your connected server. This is the password on the last page of the wizard, or the “Incoming Connection” page. You can also find the user name on this page.

Make sure the event you are updating has first been successfully forwarded to the connected server you are authenticating.

I wrote a Groovy integration script. How do I install and test it?

- Create your new script under:
Administration > Setup and Maintenance > Connected Servers
Click **Manage Scripts**.
- Configure a new external process connected server and select your script from the drop-down menu.
- Create a forwarding rule with the title “my_forward_test” to forward events automatically to your target connected server.
- Use the `sendEvent` tool to create a test event:

```
<OMi_HOME>/opr/support/sendEvent.bat -t “my_forward_test”
```
- Check to see if the event was delivered to your server.

Alternatively, you can select **Enable Synchronize and Transfer Control** in the Connected Server Outgoing Connection screen. You can then manually forward the event from the Event Browser. You can then also check the external event by selecting the External Info tab.

I automatically forwarded an event to my external server, but I cannot see the External Info tab in the Event Browser. What happened?

The External Info tab is only for events that have transferred control to an external server. If you forwarded the event using one of the other types of forwarding, for example, Notify, Notify and Update, or Synchronize, then you will not be able to get the external status on the External Info tab. Currently, there is no interface in the Event Browser to view other types of forwarding.

Can I forward an event to more than one external system?

Yes. You can forward an event to as many systems as you like, but you can transfer control to only one system.

I transferred control to an external system. Can I take it back?

This is currently not possible.

I transferred control to an external system. Can that system give it back?

Yes. The external system must generate an update using the Event Synchronization Web Service. When the Groovy script method `receiveChange()` is called, you can set the `args` property as like this:

```
args.returnControl
```

This returns control back to the local OMi instance, assuming the connected server had control and is logged in.

Groovy Scripts and Programming

This section contains frequently asked questions related to Groovy scripts and programming.

Where can I find out more about Groovy programming?

There is a beginners tutorial that you can access here:

<http://groovy.codehaus.org/Beginners+Tutorial>

I see in the Logfile Adapter sample that the "?" character is sometimes used. What does it mean?

Consider this line of code:

```
def username = args.credentials?.userName
```

If the value of `credentials` is null, there will be no attempt at runtime to de-reference it. Null will simply be assigned to `username`. If the value of `credentials` is not null, `userName` will be assigned to `username`.

The payload being sent by my connected server is XML. How can I parse it from the Groovy script?

A simple way to parse the XML from Groovy is to use the `XmlSlurper`. See the sample code below and the `LogfileAdapter.receiveChange()` method.

The following sample assumes a payload with an OPR event (XML). This payload will vary depending upon the connected server sending the update.

```
def receiveChange(def args) {
    def timestamp = new Date()
    def externalEvent = args.externalEventChange
    def msg = """"### ${timestamp.toString()}: receiveChange() called ###
parameter externalEvent: ${externalEvent}\n\n""
    m_logfile.append(msg)
    if ((externalEvent == null) || (externalEvent.length() == 0))
        return false;
    // check if this is an event or event_change
    def xmlNode = new XmlSlurper().parseText(externalEvent);
    if (xmlNode.name().equals("event"))
        return handleEvent(args, xmlNode)
    else if (xmlNode.name().equals("event_change"))
        return handleEventChange(args, xmlNode)
    else {
        def err = "Unexpected object type: ${obj.getClass().canonicalName}"
        m_logfile.append("${err}\n\n")
        m_logger.error(err);
        return false
    }
}

def handleEvent(def args, def event) {
    m_logger.debug("Change request received with ${EVENT_TAG} record.")
    // Update the event properties if present in XML
    if (event."title".size())
        args.title = event."title".text()
    if (event."description".size())
        args.description = event."description".text()
    if (event."solution".size())
        args.solution = event."solution".text()
    if (event."severity".size()) {
        def text = event."severity".text()
        def severity = severityMap."${text}"
        if (severity)
            args.severity = severity
        else {
            args.setHttpResponseStatus(400, "Invalid severity: ${text}")
            return false
        }
    }
}
```



```
    }  
    ...
```

I want to map my external process severity to an OMi event severity. Is there an easy way to do that in Groovy?

Use a map as follows:

```
// Map for severity mapping  
static def severityMap = ["0": OprSeverity.unknown,  
                          "1": OprSeverity.normal,  
                          "2": OprSeverity.warning,  
                          "3": OprSeverity.minor,  
                          "4": OprSeverity.major,  
                          "5": OprSeverity.critical]  
  
...  
def externalSeverity = "2"  
def oprSeverity = severityMap."${externalSeverity}"
```

Integration Script Methods

This sections contains frequently asked questions related to integration script methods.

What script methods must I implement?

You must implement the following methods in any integration script:

- `init()`
- `destroy()`
- `forwardEvent()`

For more information about script methods, see ["Groovy Script Methods" on page 212](#).

Which script methods are optional?

The following script methods are optional:

- `forwardChange()`: Only needed if your script adapter supports the following forwarding modes: Notify and Update, Synchronize, or Synchronize and Transfer Control.
- `receiveChange()`: Only needed if your adapter supports the following forwarding modes: Synchronize or Synchronize and Transfer Control.
- `getExternalEvent()`: Only needed if your script adapter supports populating the External Info tab in the Event Browser.
- `toExternalEvent()`: Only needed if your connected server needs to perform a GET HTTP method call at the Event Synchronization Web Service, to retrieve the current properties of the event that originated in OMi.

Must my class implement the EventProcessAdapter interface?

No. Your script must only implement the required methods. The interface is provided for documentation and for those that wish to use an integrated development environment (IDE) such as Eclipse or IntelliJ.

When do init() and destroy() get called?

The `init()` method gets called when the script is first loaded. There is one instance loaded per connected server and done so on the first access to the connected server. It remains loaded until the connected server configuration is updated, or the script is changed. At that time, the `destroy()` method is called, and the script is reloaded. Once loaded, the script may maintain state between calls, for example, it may leave connections open to a remote server and reuse those connections on subsequent calls.

What properties are available in the init() method argument?

For available properties for the `init()` method, see ["Properties for init\(\) Method" on page 212](#).

Example:

```
def init(def args) {
    m_logger = args.logger
    m_initArgs = args
    def logfileDir = new File("${args.installDir}${File.separator}${LOG_DIR_REL}")
    if (!logfileDir.exists())
        logfileDir.mkdirs()
    m_logfile = new File(logfileDir, LOGFILE_NAME)
    if (!m_logfile.exists())
        m_logfile.createNewFile()
    m_logger.debug("Logfile Adapter initialized. INSTALL_DIR=${args.installDir}")
    def timestamp = new Date()
    def msg = """"### ${timestamp.toString(): init() called ###
parameter connected server ID: ${m_initArgs.connectedServerId}
parameter connected server name: ${m_initArgs.connectedServerName}
parameter connected server display name: ${m_
initArgs.connectedServerDisplayName}
parameter node: ${m_initArgs.node}
parameter port: ${m_initArgs.port}
parameter ssl: ${m_initArgs.nodeSsl}
parameter drilldown node: ${m_initArgs.drilldownNode}
parameter drilldown port: ${m_initArgs.drilldownPort}
parameter drilldown ssl: ${m_initArgs.drilldownNodeSsl}\n\n""""
    m_logfile.append(msg)
}
```

For details, see the `com.hp.opr.api.ws.adapter.InitArgs` Javadoc.

What properties are available in the ping() method argument?

For available properties for the `ping()` method, see ["Properties for ping\(\) Method" on page 213](#).

Example:

```
def ping(def args) {
    args.outputDetail = "success"
    return true
}
```

For details, see the `com.hp.opr.api.ws.adapter.PingArgs` Javadoc.

What properties are available in the `forwardEvent()` method argument?

For available properties for the `forwardEvent()` method, see ["Properties for forwardEvent\(\) Method" on page 214](#).

Example:

```
def forwardEvent(def args) {
    def timestamp = new Date()
    def extId = "urn:uuid:${args.event.getId()}"
    def msg = """"### ${timestamp.toString(): forwardEvent() called ###
event.id: ${args.event.id}
event.title: ${args.event.title}
event.state: ${args.event.state}
event.external.id: ${extId}\n\n""""
    m_logfile.append(msg)
    // Set the external reference ID
    args.externalRefId = extId
    // Make a drilldown to the original event as an example
    args.drilldownUrl =
        new URL("http://${m_initArgs.drilldownNode}:${m_initArgs.drilldownPort}
${ROOT_DRILLDOWN_PATH}${args.event.getId()}")
    return true
}
```

For more details on the argument passed to the method `forwardEvent()`, see the `com.hp.opr.api.ws.adapter.ForwardEventArgs` Javadoc.

What properties are available in the `forwardChange()` method argument?

For available properties for the `forwardChange()` method, see ["Properties for forwardChange\(\) Method" on page 217](#).

Example:

```
def forwardChange(def args) {
    def timestamp = new Date()
    StringBuffer buff = new StringBuffer()
    buff.append("### ${timestamp.toString(): forwardChange() called ###\n")
    buff.append("    parameter externalRefId: ${args.externalRefId}\n")
    buff.append("    change headline: ${args.changes.headline}\n")
    args.changes.changedProperties.each {
        def propertyChange ->
```

```

        buff.append("    changed property: ${propertyChange.propertyName}
=${propertyChange.currentValue}\n")
    }
    buff.append("\n")
    m_logfile.append(buff.toString())
    return true
}

```

For more details on the argument passed to the method `forwardChange()`, see the `com.hp.opr.api.ws.adapter.ForwardChangeArgs` Javadoc.

What properties are available in the `receiveChange()` method argument?

For available properties for the `receiveChange()` method, see ["Properties for receiveChange\(\) Method" on page 220](#).

Example:

```

def receiveChange(def args) {
    def timestamp = new Date()
    def msg = """"### ${timestamp.toString(): receiveChange() called ###
parameter externalEvent: ${args.getExternalEventChange()}\n\n""""
    m_logfile.append(msg)
    def jc = javax.xml.bind.JAXBContext.newInstance(
com.hp.opr.api.ws.model.event.OprEvent.class)
    def event = jc.createUnmarshaller().unmarshal(
new CharArrayReader(args.getExternalEventChange().toCharArray()))
    if (event instanceof com.hp.opr.api.ws.model.event.OprEvent) {
        if (event.titleUpdated)
            args.title = event.title
        if (event.descriptionUpdated)
            args.description = event.description
        if (event.solutionUpdated)
            args.solution = event.solution
        return true
    } else {
        def err = "Unexpected object type: ${obj.getClass().canonicalName}"
        m_logfile.append("${err}\n\n")
        m_logger.error(err);
        return false
    }
}

```

For more details on the argument passed to the method `receiveChange()`, see the `com.hp.opr.api.ws.adapter.ReceiveChangeArgs` Javadoc.

When processing `receiveChange()`, I would like to send back a particular response to the web service caller. How can I do that?

The `args` has a method to allow you to control the response:

```
args.setHttpResponseStatus(400, "My response message")
```

You can set the HTTP status and payload to anything you wish. If the value is less than 300, the payload is processed after the `receiveChange()` method is called. Then the status and message are returned to the web service caller, otherwise the HTTP status and message are returned immediately to the we service caller.

What properties are available in the `getExternalEvent()` method argument?

For available properties for the `getExternalEvent()` method, see ["Properties for getExternalEvent\(\) Method" on page 225](#).

Example:

```
def getExternalEvent(def args) {
    def timestamp = new Date()
    def msg = ""### ${timestamp.toString()}: getExternalEvent() called ###\n\n""
    m_logfile.append(msg)
    args.assignedUser = "logger"
    args.assignedGroup = "logging group"
    args.state = "open"
    args.severity = "normal"
    args.priority = "none"
    return true
}
```

For more details on the argument passed to the method `getExternalEvent()` see the `com.hp.opr.api.ws.adapter.GetExternalEventArgs` Javadoc.

Event Properties

This sections contains frequently asked questions related to event properties.

What event properties exist?

All properties available in the Event Browser, or that can be found in the Event Web Service are available in the `OprEvent` object. Details can be found in the Javadoc for `OprEvent`.

For an example go to the Event Web Service and list the events:

```
https://<server.example.com>/opr-web/rest/9.10/event_list
```

The XML output will give you a good idea of the properties that are available. This XML output is directly generated from the `OprEvent` object.

I want to see what is the related CI for this event. How can I get this information?

`event.relatedCi` returns an object with properties describing the related CI. It is of type `OprRelatedCi`. `event.relatedCi.configurationItem` contains the key properties of the CI, and, if the CI is part of another CI, it contains the CI it is part of:

`event.relatedCi.configurationItem.partOf`. "partOf" is of type `OprRelatedCi`, so this will continue until there are no more parts. This should provide you with enough details to identify the CI in

an external system.

An OprConfigurationItem object does not define all key properties for all CIs.

How do I get the other key properties?

Use the `OprConfigurationItem` utility method to get the other properties: `getProperty(name)`, or for a map of all properties call `getProperties()`.

The OprConfigurationItem in the event only has the key properties. How do I get the other properties?

Call the utility method `getCi(id)` for the CI you want. All properties will be set in the CI that is returned from this method. This utility method is available in the args for `forwardEvent()`, `forwardChange()`, and `receiveChanges()`.

What are the possible class types that can be returned in the JAXBElements returned from OprConfigurationItem.getAny()?

The possible class types are:

- String
- Boolean
- Integer
- Long
- Float
- Double
- Date

OprConfigurationItem.getAny() returns multiple objects with the same name.

How can this happen?

If the CI property is a list, you will get multiple entries.

Troubleshooting

This section contains frequently asked questions related to troubleshooting the connected server.

I configured a connected server, but it does not show up in the Event Browser context menu item "Transfer Control to". Why?

Check the following:

- Make sure the connected server is "Active". Found on "General" tab.
- Make sure the connected server supports "Ownership Transfer". Found on "Outgoing Connection" tab.

I have forwarded an event to my external connected server. How can I tell if my script was called?

Try the following:

- Switch logging to debug level.
- Check the logfile.

Logging

This section contains frequently asked questions related to log files.

Where can I view the log file for script execution?

You can view the log file at this location:

```
<OMi_HOME>/log/opr-event-sync-adapter.log
```

How can I change the logging level?

For `getExternalEvent()` method calls, you need to edit the following properties file:

```
<OMi_HOME>/conf/core/Tools/log4j/jboss/opr-event-sync-adapter.properties
```

For all other method calls, edit the following properties files:

```
<OMi_HOME>/conf/core/Tools/log4j/wde/opr-event-sync-adapter.properties
```

```
<OMi_HOME>/conf/core/Tools/log4j/opr-ctxm-server/opr-event-syncadapter.properties
```

You need to set the `logLevel` parameter towards the top of the file. The file contains possible values.

How can I log from my script?

The `args` passed to the `init()` method has a property called `logger`. Use this `logger` for logging. For example:

```
def logger = args.logger
logger.info("This is an info log")
logger.warn("This is a warning log")
logger.error("This is a error log")
logger.debug("This is a debug log")
logger.error("This is a error log with an exception", exception)
```

Chapter 27: Integrating an External Event Processing Service Defined by a WSDL

You can integrate an external event processing system that exposes its interfaces using a standard web service and a WSDL (Web Services Description Language) description using the steps described in this section.

HPE recommends that you implement the integration in stages that correspond to the four defined forwarding types. Each stage builds upon the previous to create a more fully functional integration:

- **All forwarding types.** You must implement the Groovy script `init()` method for all forwarding types. The method is called whenever the Groovy script is initialized.
- **Notify.** This is the minimum and requires implementing the `forwardEvent()` method in the Groovy script to forward events to the target server.
- **Notify and Update.** This requires implementing the `forwardChange()` method in the Groovy script to forward changes to the target server.
- **Synchronize.** This requires implementing the `receiveChange()` method in the Groovy script to receive changes from the target server.

The target server must be able to call the OPR Event Synchronization Web Service to post the changes to OMi when a change occurs on the target server. The payload of the web service request is passed to the Groovy script `receiveChange()` method for interpretation. The web service call may be of any type, for example SOAP or REST based.

- **Synchronize and Transfer Control.** No additional Groovy script methods are required for this implementation.
- *Optional. Ping support.* This requires implementing the Groovy script `ping()` method.

For more information about Groovy script methods, see ["Groovy Script Methods" on page 212](#).

The following configuration steps are required to implement an integration for the notify forwarding type:

1. ["Generate Java Code from WSDL" below](#)
2. ["Configure the External Event Processing Application as a Connected Server" on page 242](#)
3. ["Test the External Event Creation" on page 243](#)

To support the other forwarding types, the appropriate methods in the Groovy script must be implemented in a similar fashion as the `forwardEvent()` method.

Generate Java Code from WSDL

The following example uses Apache Axis to generate Java code from a WSDL file.

1. Download and install the following prerequisites:
 - Java JDK 1.7 or later
 - Apache Axis2
 - Apache Ant 1.7

2. Create a working directory, for example `integration`.
3. Copy the WSDL file to that directory, for example `integration/service.wsdl`.
4. Create a directory for the generated code, for example `integration/gen`.
5. Change to the `gen` directory.
6. Run Apache Axis2 and generate the code from the WSDL file:

```
wsdl2java -uri file:../service.wsdl
```
7. Edit the generated `build.xml` file to correctly set the `classpath` in the manifest of the JAR file:
 - a. Open the `build.xml` file in a text editor.
 - b. Locate the Ant target `jar.client`.
 - c. Add the following Ant path declaration just after the path declaration `axis2.class.path` at the beginning of the file:

```
<path id="axis2.client.class.path">
  <fileset dir="${axis2.home}">
    <include name="lib/*.jar"/>
  </fileset>
</path>
<pathconvert property="axis2.client.class.path.string"
  pathsep=" ">
  <path refid="axis2.client.class.path" />
  <flattenmapper />
</pathconvert>
```

- d. In the `jar` task, add the following manifest directive to specify the `classpath` in the JAR file manifest.

As there are many Axis2 JAR files needed at execution time, it is simpler if the manifest of the generated JAR file is able to resolve the dependencies at runtime.

```
<jar destfile="${lib}/${name}-test-client.jar">
  <fileset dir="{classes}">
    <exclude name="**/META-INF/*.*/>
    <exclude name="**/lib/*.*/>
    <exclude name="**/*MessageReceiver.class"/>
    <exclude name="**/*Skeleton.class"/>
  </fileset>
  <manifest>
    <attribute name="Created-By"
      value="Developer name goes here" />
    <attribute name="Class-Path"
      value="${axis2.client.class.path.string}" />
  </manifest>
</jar>
```

- e. Save the changed `build.xml` file.
- f. Run `ant` at the command line. This builds the JAR file needed to access the service. The output is available the `build/lib` directory.

Configure the External Event Processing Application as a Connected Server


Synchronizing events and event changes between OMi and the external event processing application depends on a server hosting OMi forwarding events to the external event processing application. To achieve this you must configure the external event processing application as a target connected server in the Connected Servers manager.

When you configure the connected server, you also create the Groovy script to access the service.

For full details about how to configure a connected server, see the *OMi Administration Guide*.

1. Navigate to the Connected Servers manager:

Administration > Setup and Maintenance > Connected Servers

2. Click the  **New** button to open the Create New Server Connection dialog.
3. In the **Display Name** field, enter a name for the external event processing application server. By default, the display name is filled automatically.

Optional: Enter a description for the new target server.

Make sure the **Active** check box is selected.

Click **Next**.


4. Select **External Event Processing** to choose the server type suitable for an external event processing application.

Click **Next**.

5. Enter the **Fully Qualified DNS Name** of the external event processing application server.

Click **Next**.

6. Select the **Integration Type** used to establish the connection to the external server:

- a. Select **Call Script Adapter**.
- b. In **Script Name**, click **Manage Scripts** to open the Event Forwarding Scripts Configuration dialog box.
- c. Select the `sample:LogfileAdapter` Groovy script and click  **Duplicate Item** to create a copy of the script.
- d. In the General tab of the script editor, rename the copy to `TestAdapter`.
- e. In the Script tab, add calls to the classes in the Axis2 generated JAR file:
 - You must first call the `forwardEvent()` script method. You need to create a target object for a given event.
 - The Axis2 code generator generated a stub class. You need to construct this class and then, using the event passed in the args to `forwardEvent()`, create an external event.
 - Each service will be different, so there are no specifics to supply here. It is expected that once the external event is successfully created the `externalID` is set in the args before the method is exited.
- f. In the Advanced tab, add all JAR files required by the `TestAdapter` script.

- g. Click **OK** to save the TestAdapter script and then close the Event Forwarding Scripts Configuration dialog box.
 - h. Click **Next** in the Integration Type page of the Connected Server wizard.
7. In the Outgoing Connection dialog, provide the credentials (user name, password, and port number) to connect to the external event processing application target server and to forward events to that server.

Select **Enable Synchronize and Transfer Control** for initial testing. When the Enable Synchronize and Transfer Control flag is set, an OMi operator is then able to transfer ownership of the event to the target connected server. If the flag is not set, then the option Synchronize and Transfer Control does not appear in the list of forwarding types when configuring forwarding rules.

If the Enable Synchronize and Transfer Control flag is not set for any target connected server, the Transfer Control to option does not appear at all in the Event Browser context menu.

If a specific server is configured without the Enable Synchronize and Transfer Control flag set, then that server is not available in the Event Browser context menu as a server to which you can transfer ownership.

Click **Next**.

8. Complete the remaining dialogs and then click **Finish**.
The target external event processing application server appears in the list of Connected Servers.

Test the External Event Creation

1. Open the Event Browser on the system running OMi.
2. Select an event.
3. Right-click the event and select **Transfer Control > <external event processing application target server>**.
4. Verify that the event appears in the external event processing application target server.

Chapter 28: Error Handling

The OMi event synchronization interface that forwards events and subsequent changes uses the error handling algorithms described in this section to determine if a forward request should be retried or discarded.

Error handling is different depending on the type of integration:

Groovy Script Integration

If you implement a Groovy script to integrate your application, error handling is as follows.

For `forwardEvent()` and `forwardChange()`:

- Return of true: The request is marked as FORWARDED and the forward request is removed from the queue.
- Return of false: The request will be put back in the queue. Retries are done once a minute until the request succeeds or the request is in the queue longer than the Event Forwarding Expiration Time (default is 12 hours and may be changed in the Infrastructure Settings). New events and updates are forwarded in the order the forward request was received. Should a request fail due to a Non-RuntimeException error it will block all other requests for this server. All new events in the queue are delivered before delivering any outstanding updates.

For `forwardEvents()`:

- Return of true: All forward event requests are marked as FORWARDED and the forward requests are removed from the queue. If all events have had the forward status set to FORWARDED, the exception is logged.
- Return of false: If there are events that are not marked as FORWARDED, then the first event without a result is handled in the normal exception processing. The remaining events are left queued and retried later.

See also standard error handling for `forwardEvent()` defined in the `com.hp.opr.api.ws.adapter.ExternalProcessAdapter` interface.

For `forwardChanges()`:

- Return of true: All forward change requests are marked as FORWARDED and the forward requests are removed from the queue. If all change requests have had the forward status set to FORWARDED, the exception is logged.
- Return of false: If there are changes that are not marked as FORWARDED, then the first change without a result is handled in the normal exception processing. The remaining changes are left queued and retried later.

See also standard error handling for `forwardEvent()` defined in the `com.hp.opr.api.ws.adapter.ExternalProcessAdapter` interface.

For `receiveChanges()`:

- Error handling is the same as for `receiveChange()` defined in the `com.hp.opr.api.ws.adapter.ExternalProcessAdapter` interface.

If the Groovy script throws an exception, it is handled as follows:

- `org.apache.wink.client.ClientWebException`: The exception is queried for the HTTP status code returned by the server. If an HTTP status code is found the status code is handled just as in the Web Service Integration case described in ["Error Handling" on page 249](#). If the status code does not exist, the exception is handled just as any other exception, see below for more information.
- Any exception other than `ClientWebException`: The exception is recursively searched for the root cause exception and then interpreted as follows:
 - `RuntimeException`: Error is logged to the `opr-event-sync-adapter.log` log file and the request is marked as FAILED. There is no retry for this request.
 - `Non-RuntimeException`: Examples would be `IOException`, `SocketException`, and so on. Error is logged to the `opr-event-sync-adapter.log` log file. In this case it is expected that the connection to the server will recover at some future time and be able to send the request. The request will be put back in the queue. Retries are done once a minute until the request succeeds or the request is in the queue longer than the Event Forwarding Expiration Time (default is 12 hours and may be changed in the Infrastructure Settings). New events and updates are forwarded in the order the forward request was received. Should a request fail due to a `Non-RuntimeException` error it will block all other requests for this server. All new events in the queue are delivered before delivering any outstanding updates.

Web Service Integration

For details, see ["Error Handling" on page 249](#).

Chapter 29: Service Manager Integration

For information on how to connect to HPE Service Manager and to configure event forwarding and synchronization to and from HPE Service Manager, see the *OMi Integrations Guide*.

Part VIII: Web Service Interfaces

A number of web services allow integrators to access OMi functionality from external applications.

The section details several general aspects that need to be observed when using any of the web services. The remainder of the chapter comprises one section for each available web service with reference information and examples.

This part is structured as follows:

- ["Reference Information For All Web Services" on page 248](#)
- ["Action Web Service Interface" on page 258](#)
- ["Downtime Web Service Interface" on page 268](#)
- ["Event Synchronization Web Service Interface" on page 275](#)
- ["Event Web Service Interface" on page 295](#)
- ["Monitoring Automation Web Service Interface" on page 296](#)
- [Status Web Service Interface](#)
- ["Tool Execution Web Service Interface" on page 311](#)
- ["User Management Web Services Interface" on page 322](#)

Chapter 30: Reference Information For All Web Services

This section describes what to consider when using any of the available web service interfaces:

- [Authentication](#) 248
- [Error Handling](#) 249
- [Error Verbosity](#) 249
- [Securing Modify Operations](#) 250
- [Environments with CA SiteMinder](#) 256
- [Session Handling](#) 256
- [Settings](#) 256

Authentication

Consider the following with regard to authentication and security when using web services:

- Web services can only be executed by regular OMi users.
- User rights assumed when executing a web service request are identical to the user rights you would have when performing the same operation using the OMi user interface, being logged in as the user whose credentials are used for web service authentication.
- Network encryption and data integrity is supported only when using the HTTPS protocol.

The following code sample shows a JAVA implementation of basic authentication using a username/password combination:

Example:

```
import org.apache.commons.codec.binary.Base64;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.MediaType;
/** Create the request URL (assuming the capitalized variables are set to
appropriate values). */
String url = "http://" + WS_SERVER_HOSTNAME + ROOT_PATH + REQUEST;

byte[] encodedUserPassword =
    Base64.encodeBase64((username + ":" + password).getBytes());
Response response =
    client.target(url).request(MediaType.APPLICATION_XML_TYPE).
    header("Authorization", "Basic " +
    new String(encodedUserPassword)).get();
```


The following types of authentication are also supported:

- Manual entry of user name and password combination valid for logging on to OMi. This method is useful for ad-hoc access to the web service, for example when using a plug-in such as [Poster](#).
- Light-Weight Single Sign-on (LWSSO)
- Windows Authentication (WinAuth)
- Common Access Card (CAC)

Error Handling

If you implement a web service endpoint that can be directly called by OMi, error handling is as follows:

- **HTTP status of 2xx returned.** The following HTTP status codes indicate a request was processed successfully:
 - 200 (OK)
 - 201 (Created)
 - 202 (Accepted)
- **HTTP status of 3xx returned.** These are redirection status. Redirection is not supported by the service. They are treated the same as 4xx, therefore the request will be marked as FAILED and no further retries are done.
- **HTTP status of 4xx returned.** Any status of 4xx is considered an error in the client request, therefore the request will be marked as FAILED and no further retries are done.
- **HTTP status of 5xx returned.** Any status of 5xx is considered an error on the server. In this case it is expected that the server will recover at some future time and be able to accept the request. The request will be put back in the queue. Retries are done once a minute until the request succeeds or the request times out (default is 12 hours and may be changed in the *Infrastructure Settings*). New events and updates are forwarded in the order the forward request was received. Should a request fail due to a 5xx error it will block all other requests for this server. All new events in the queue are delivered before delivering any outstanding updates.
- Any type of `IOException` encountered while trying to communicate with the server will result in the request being re-queued and then retried as described for HTTP status of 5xx.

Error Verbosity

Web Services return error codes that help administrators identify the cause of possible problems. Administrators can control the types of error codes returned from the Web Service by changing the level of verbosity in the Infrastructure Settings Manager. The following options are available:

Option	Description
Standard	Default. Returns to the caller of the web service the appropriate HTTP error code, as listed in the HTTP 1.1 standard, along with a standard error text message describing the error.

Option	Description
Verbose	Recommended for development environments. Returns a detailed message describing the reason for the error.
Brief	Returns only error codes 400 (Bad Request) or 503 (Service Unavailable), depending on the type of the error, and an error identifier. A detailed error message may be obtained by locating the identifier and detailed error message in the error logs.

1. Navigate to the in the Infrastructure Settings Manager:
Administration > Setup and Maintenance > Infrastructure Settings
2. Select application **Operations Management** and find the section **Operations Management - Web Service Settings > Error Response Verbosity**.
3. *Optional.* Change the default value **Standard** to another value.
4. Review the error log file on the OMi Gateway Server:
`<OMi_HOME>/log/opr-ws-response.log`

Securing Modify Operations

Web Service modify operations must be secured by setting the X-Secure-Modify-Token HTTP header on modify requests (PUT, POST, and DELETE). This header provides enhanced security protection against malicious exploits of web applications.

Protection for modify operations is by default enabled in the Web Service Settings in the Infrastructure Settings Manager. You can disable the setting for backwards compatibility (see "[Disable Enhanced Security Protection](#)" on page 255).

Setting the X-Secure-Modify-Token HTTP Header

Web Service clients must first obtain the `secureModifyToken` cookie, and then set the value of the cookie in the X-Secure-Modify-Token HTTP header:

1. Obtain the `secureModifyToken` cookie before executing any modify requests (PUT, POST, or DELETE).

The recommended approach to obtain the `secureModifyToken` cookie at client startup is to execute an HTTP GET request for the Web Service Service Document at `/opr-web/rest`.

After the HTTP GET operation has completed the cookie is set. During the life of the client and the single sign-on session, the value of the cookie may change. Before each modify operation the HTTP client should get the current value of the cookie from the client. It is not recommended to save this value in a local variable for later use, as it may change during the life of the HTTP client.

2. Set the X-Secure-Modify-Token HTTP header.

The X-Secure-Modify-Token HTTP header must be set by all Web Service clients when executing modify operations (PUT, POST, or DELETE). The value to set this HTTP header to is specified in the `secureModifyToken` cookie.

Note: The client must set *all* cookies returned by the server in subsequent requests. For example, `LWSSO_COOKIE_KEY` and `JSESSIONID` are additional cookies returned by the server and must be set

in subsequent requests to the server. If a new session is established, the `secureModifyToken` previously obtained through a GET request becomes invalid. Therefore the other cookies are also required.

Sample Code Using a Standard Java HTTP Client

The following sample code first gets the value of the `secureModifyToken` cookie and then sets the `X-Secure-Modify-Token` HTTP header.

Get the `secureModifyToken` cookie

The following method gets all the cookies from an initial GET request. The standard Java HTTP client does not automatically manage cookies for the user like the Apache `HttpClient` does. The cookies need to be obtained and managed separately.

```
private static List<HttpCookie> getCookies(final String path)
{
    final URL url = new URL(path);
    final HttpURLConnection connection = url.openConnection();
    final List<HttpCookie> result = new ArrayList<HttpCookie>();

    connection.setRequestMethod("GET");

    // Set the username and password for the request
    byte[] encodedUserPassword = Base64.encodeBase64((username + ":" +
password).getBytes());
    connection.setRequestProperty("Authorization", "Basic " + new String
(encodedUserPassword));

    connection.connect();
    int response = connection.getResponseCode();
    if (response == 200)
    {
        for (int i=1; (final String headerName = connection.getHeaderFieldKey(i)) !=
null; i++)
        {
            if (headerName.equals("Set-Cookie"))
            {
                final String cookieString = connection.getHeaderField(i);
                final List<HttpCookie> cookies = HttpCookie.parse(cookieString);
                if (cookies != null && !cookies.isEmpty())
                    result.addAll(cookies);
            }
        }
    }
    return result;
}
```

Set the X-Secure-Modify-Token HTTP header

The following code adds the cookies to the POST request and the HTTP header X-Secure-Modify-Token if the cookie secureModifyToken exists.

```
final URL url
final List<HttpCookie> cookies = getCookies("http://" + localHostName + ":" + port
+ "/opr-web/rest");

final URL url = new URL("https://" + localHostName + ":" + port + "/opr-
web/rest/9.10/event_list");
final HttpURLConnection connection = url.openConnection();

// Set the cookies and HTTP header for the request
for (HttpCookie cookie : cookies)
{
    // add the cookies to the request
    connection.addRequestProperty("Cookie", cookie.getName() + "=" + cookie.getValue
());
    if (cookie.getName().equalsIgnoreCase("secureModifyToken"))
    {
        // add the HTTP header
        connection.setRequestProperty("X-Secure-Modify-Token", cookie.getValue());
    }
}
...
```

Sample Code Using an Apache HttpClient

The following sample code first gets the value of the secureModifyToken cookie and then sets the X-Secure-Modify-Token HTTP header.

Get the secureModifyToken cookie

The following method may return null, in which case it should be assumed the target web service does not require the X-Secure-Modify-Token HTTP header. (For example, OMi versions lower than 9.10 do not require this HTTP header.)

```
private static String getSecureModifyToken(final HttpClient client, final String
url)

{
    int rc = -1;

    String secureModifyToken = null;

    // get the service document from the base path
    final HttpMethod getMethod = new GetMethod(url);
    getMethod.setFollowRedirects(true);
    getMethod.setDoAuthentication(true);
    getMethod.setRequestHeader("Accept", "text/plain, text/xml, application/xml,
```

```
application/atomsvc+xml");
    getMethod.setRequestHeader("Accept-Language", System.getProperty
("user.language", "en") + "-"
    + System.getProperty("user.country", "US"));
    try
    {
        client.executeMethod(getMethod);
        rc = getMethod.getStatusCode();
    }
    catch (IOException ioe)
    {
        // ignore any errors for backwards compatibility
    }
    if (rc == HttpStatus.SC_OK)
    {
        // look for the secureModifyToken
        Cookie[] cookies = client.getState().getCookies();
        if (cookies != null && cookies.length > 0)
        {
            for (Cookie cookie : cookies)
            {
                if (SECURE_MODIFY_TOKEN.equalsIgnoreCase(cookie.getName()))
                    secureModifyToken = cookie.getValue();
            }
        }
    }
    return secureModifyToken;
}
```

Set the X-Secure-Modify-Token HTTP header

```
HttpClient client = new HttpClient();
client.getState().setCredentials(new AuthScope(hostname, port),
    new UsernamePasswordCredentials(username, password));
final String secureModifyToken = getSecureModifyToken(client,
    "https://" + localHostName + ":" + port + "/opr-web/rest");
String url = "https://" + localHostName + ":" + port + "/opr-web/rest/9.10/event_
list";
PostMethod method = new PostMethod(url);
if (secureModifyToken != null)
    method.setRequestHeader("X-Secure-Modify-Token", secureModifyToken);
...
```

Sample Code Using an Apache Wink RestClient

The following sample code first gets the value of the secureModifyToken cookie and then sets the X-Secure-Modify-Token HTTP header.

Get initial cookies

The following method gets all the cookies from an initial GET request. The Apache Wink RestClient does not automatically manage cookies for the user like the Apache HttpClient does. The cookies need to be obtained and managed separately.

```
private static Set<Cookie> getCookies(final String url, final RestClient client)
{
    final Set<Cookie> cookies = new HashSet <Cookie>();
    final Resource resource = client.resource(url);

    // Set the username and password for the request
    byte[] encodedUserPassword = Base64.encodeBase64((username + ":" +
password).getBytes());
    resource.header("Authorization", "Basic " + new String(encodedUserPassword));
    final ClientResponse response = resource.get();

    final MultivaluedMap<String, String> headers = response.getHeaders();
    if (headers != null)
    {
        for (final Map.Entry<String, List<String>> header : headers.entrySet())
        {
            if ("Set-Cookie".equalsIgnoreCase(header.getKey()))
            {
                for (final String value : header.getValue())
                {
                    if (value != null && value.length() > 0)
                    try
                    {
                        cookies.add(Cookie.valueOf(value));
                    }
                    catch (IllegalArgumentException e)
                    {
                        // ignore this entry
                    }
                }
            }
        }
    }
    return cookies;
}
```

Set the X-Secure-Modify-Token HTTP header

The following code adds the cookies to the REST resource and the HTTP header X-Secure-Modify-Token if the cookie secureModifyToken exists.

```
final RestClient client = new RestClient();
final Set<Cookie> cookies = getCookies("https://" + localhostName + ":" + port +
```

```
"/opr-web/rest", client);

String url = "https://" + localHostName + ":" + port + "/opr-web/rest/9.10/event_
list";
final Resource resource = client.resource(url);

// Set the username and password for the request
byte[] encodedUserPassword = Base64.encodeBase64((username + ":" +
password).getBytes());
resource.header("Authorization", "Basic " + new String(encodedUserPassword));

// Set the cookies and HTTP header for the request
for (Cookie cookie : cookies)
{
    // add the cookies to the resource
    resource.cookie(cookie);
    if (cookie.getName().equalsIgnoreCase("secureModifyToken"))
    {

        // add the HTTP header
        resource.header("X-Secure-Modify-Token", cookie.getValue());
    }
}
...

```

Disable Enhanced Security Protection

Web service clients that set the X-Secure-Modify-Token HTTP header may fail when communicating with web services installed with the OMi version 9.0x and lower. You may therefore disable enhanced security protection in the Web Service Settings in the Infrastructure Settings Manager.

1. Navigate to the Infrastructure Settings Manager:
Administration > Setup and Maintenance > Infrastructure Settings
Edit **Operations Management - Web Service Settings > Secure Modify**.
2. Change the default value true to **false**.
3. *Optional*. Implement the following additional measures for end users to reduce their exposure to malicious attacks when using OMi:
 - Do not allow your web browser to save user names and passwords.
 - Do not use the same web browser to access OMi and the Internet at the same time (tabbed browsing). While logged in to OMi, the web browser should not be used to browse other web sites.
 - HTML-enabled applications that integrate web browsers (for example email or newsreader applications) pose additional risks because simply viewing an email message or a news message may lead to the execution of an attack. Caution should be taken when using client workstations connected to OMi and to such applications.

Environments with CA SiteMinder

If the CA SiteMinder Web Agent is configured for `CssChecking=YES`, characters configured in the CA SiteMinder Web Agent `BadUr1Chars` parameter are rejected by the CA SiteMinder Web Agent. By default the following characters are included in this list:

- Single quotation mark (')
- Greater than sign (>)
- Less than sign (<)

Event Web Service clients must not use these characters in the query parameter section of the URL:

- **String literals:** As the single quotation mark (') may be rejected in environments with CA SiteMinder, surround string literals with double quotation marks ("). See ["Value Types" on page 183](#) for more information.

As CA SiteMinder blocks the % encoding, the string literal itself must escape the offending characters using the dollar sign (\$) instead of percent (%), for example:

- Replace single quotation marks (') with \$60.
- Replace greater than signs (>) with \$3E.
- Replace less than signs (<) with \$3C

See ["URL Escape Codes" on page 184](#) for more information.

- **Operators:** As the greater than sign (>) and less than sign (<) may be rejected by the CA SiteMinder agent, use the GT and LT aliases instead. See ["Operator Aliases" on page 182](#) for more information.

Session Handling

By default, the timeout period of an OMi REST web service session is 30 seconds. You can change the default in the Infrastructure Settings Manager:

Administration > Setup and Maintenance > Infrastructure Settings

Select **Operations Management** from the **Applications** drop-down list. In the **Web Service Settings** table, edit the **REST web service Session timeout** setting.

Alternatively, add the `sessionTimeOut` property to the HTTP request header, or add the query parameter `sessionTimeOut` to the HTTP URL. The `sessionTimeOut` parameter in the HTTP URL overrides the `sessionTimeOut` property in the HTTP request header, and both override the default timeout set in the infrastructure setting.

The timeout period is set when the web service client logs onto OMi for the first time.

Settings

You can define a number of settings for web services in the Infrastructure Settings:

Administration > Setup and Maintenance > Infrastructure Settings

Select **Operations Management - Web Service Settings**.

Details about the effect of these settings are provided with each setting in the Infrastructure Settings Manager.

Chapter 31: Action Web Service Interface

The REST-based Action Web Service enables integrators to access the following functionality from an external application:

- Execute an arbitrary command string on one or more monitored nodes. Requests can be run in parallel.

A command type must be specified. A command can be executable (default), Perl script, VBScript, JavaScript, or Windows Scripting Host script. The script code must be passed in the web service call. For Unix agents, commands are executed in a shell. For Windows agents, commands are not executed in a shell. For example, an "echo hello world" command would read `echo hello world` on Unix and `cmd /c echo hello world` on Windows.

Target hosts can be specified by hostname (from the RTSM) and by IP address.

Requests are dropped when the OMi backend is disabled or restarted. In some cases, an Action Web Service request will not finish. For example, if you use the `ovc-stop` command to stop the action agent process, which sends back a call response, the call will continue to run until the OMi backend is disabled or restarted.

On the agents, environment variables like `OvDataDir`, `OvInstallDir`, and so on are predefined. For a complete list you can call the OMi server using the following command:

```
$ <OvBinDir>/ovdeploy -cmd set [ -host <remoteHost> ] | grep "Ov"
```

Linux output: `OvBinDir = /opt/OV/bin`

Windows output: `Program Files\HP\HP BTO Softeare\bin\win64`

- Cancel the command on the managed nodes.
- Retrieve the command status for the managed nodes.
 - The OMi backend keeps the execution results for 10 minutes after the last managed node has finished execution. Web service clients must retrieve the results in that time period.
- Retrieve the detailed command status for a specific managed node.

The Action Web Service only triggers command strings entered by the user. In this case, an "action" is an arbitrary string entered by the web service user, as opposed to a "tool", which is a predefined string stored in the database. The Action Web Service does not trigger automatic actions or user actions.

For further information about actions, see the *OMi User Guide*.

For information about tools, see the *OMi Administration Guide*.

When using a JAVA client, the classes provided in the `opr-external-api.jar` can be used to marshal or unmarshal the XML content directly. See also the example code in `<OMi_HOME>/opr/examples/action-ws-client`

How to Access the Action Web Service

Your entry point to the web service interface is the Service Document, using the following base URL and syntax:

<code>https://<serviceURL>/<request></code>	
<code><serviceURL></code>	The URL of the Action web service: <code><server>:<port>/opr-web/rest/<version></code> where <code><server></code> is the name of the OMi server and <code><version></code> is the web service interface version (for example: 10.10).
<code><request></code>	The service request. For a complete list and syntax of possible service requests, see "Action Web Service Request Reference" on the next page.

List of Services in the Service Document

OPR Action Web Services	URL
Action Web Service POST request:	<code>https://<server>:<port>/opr-web/rest/10.10/direct_execution</code>
Action Web Service GET DELETE request:	<code>https://<server>:<port>/opr-web/rest/10.10/direct_execution/{contextId}</code>
Action Web Service GET details DELETE request on single node:	<code>https://<server>:<port>/opr-web/rest/10.10/direct_execution/{contextId}/{resultId}</code>

Security and Authorization

Because the Action Web Service enables the user to trigger mass requests that run on all managed nodes, security and authorization settings are especially important. It is highly recommended that you configure OMi to use HTTPS while working with the Action Web Service. Make sure that the following security requirements have been met:

1. Check if the Action Web Service is enabled. The Web Service is enabled by default. To edit the Tool Execution infrastructure setting, go to:
Administration > Setup and Maintenance > Infrastructure Settings
 Select **Applications** and use the list to set the context to **Operations Management**. Scroll down to **Operations Management - Action Web Service Settings**. Set **Allow command execution through Action Web Service** to true.
2. To be able to access the Action Web Service, a user needs to be assigned a role with the **Action Web Service** permission. This permission can be accessed in the **Operations Console** category of the permission section of a role. For more information on permissions, see ["Users, Groups, and Roles" on page 1.](#)
3. The Action Web Service user must specify a valid username and password in the web service.

Error Handling

Error handling for the Action web service is as follows:

HTTP error codes:

- HTTP error code 403: authorization problems
- HTTP error code 409: requested hosts not in the RTSM or no CIs match requirements
- HTTP error code 4xx (other than 403, 409): any form of communication problems with the OMi backend
- HTTP code 200: at least one requested host is in the RTSM and meets the requirements

Individual return codes per requested host (HTTP code always 200):

- Return code -404: requested host is not in the RTSM or the CI does not meet the requirements
- Return code -500: communication problem with the requested host
- Return code 0: execution succeeded
- Other return codes (for example, 1): error code from the command to be executed

Action Web Service Request Reference

The Action Web Service interface supports the following requests:

Action Service

Request	Request Type	Web Service Action
/direct_execution[/ {executionContextID}[/ {nodeSpecificResultID}]]	GET	Retrieve the command status. /<executionContextID>/<nodeSpecificResultID> is specified: Get the detailed command status for one specific managed node. /<executionContextID> is specified, but /<nodeSpecificResultID> is omitted: Get the overview status for all concerned managed nodes. Note: /<executionContextID> is required.
	POST	Execute a command as specified in the posted XML data. Note: • /<executionContextID> is required.
	DELETE	Cancel the execution. /<executionContextID>/<nodeSpecificResultID> is specified: Cancel the execution on a specific node. /<executionContextID> is specified, but /<nodeSpecificResultID> is omitted: Cancel the execution on all concerned nodes with ID <executionContextID>.

XML Structures

For more information on XML structures, see also the OMi public web service schema definitions in <OMi_HOME>/opr/api/schema/OprDataModel.xsd.

The following XML structure examples include comments to help you to tailor them to your specific uses.

This section includes:

- "Execute a command string:" below
- "Retrieve status overview:" on page 263
- "Retrieve detailed status for one node:" on page 263

Execute a command string:

POST - https://<server>:<port>/opr-web/rest/10.10/direct_execution

Input:

```
<direct_execution xmlns="http://www.hp.com/2009/software/opr/data_model" xmlns:\
xs="http://www.w3.org/2001/XMLSchema">
<!--wrapper element for command execution-->
  <command>
    <!-- mandatory; the command string to execute-->
    <tool_type>
      <!-- mandatory; can be om_agent_command, perl, vb_script, java_script or wsh
(Windows scripting host)-->
      <!-- If set to om_agent_command, it's executed by the OM action agent
(opcacta), otherwise it's executed by the built-in module (e.g. for perl) from
the OM monitor agent (opcmona)-->.
      <run_as_username>
        <!-- optional; execute the command as this user; by default the command is
executed as the OM agent user.-->
        <!-- on Unix setting this XML tag has no impact if the OM agent doesn't run
with superuser right;-->
        <!-- for Windows, the user password must be specified to switch to that user,
but as opposed to in Unix, you can switch to any user.-->
        <run_as_user_password>
          <!-- optional; Windows only, password for the user to run the command as; see
above-->
          <execution_hosts>
            <!-- mandatory; list of hosts to execute the command on-->
            <execution_host>
              <!-- mandatory; hostname or IP address of a managed node-->
```

Output:

```
<tool_execution_context xmlns="http://www.hp.com/2009/software/opr/data_model" x
mlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- wrapper element for the execution context-->
  <id>
    <!-- executionContextID; needed for all later calls-->
  <tool_ref>
    <!-- only relevant for tool web service, not for action web service-->
  <user_parameters/>
    <!-- only relevant for tool web service, not for action web service.-->
  <tool_execution_result_ref>
    <!-- status for a specific managed node-->
    <id>
      <!-- the <nodeSpecificResultId>; needed when later fetching result details.-
->
    <execution_host>
      <!-- the host the command is running on-->
    <tool_command>
      <!-- only available for tools of type om_agent_command. command string
executed on the host-->
    <tool_execution_state>
```

```
<!-- can be: pending (not yet started), running, canceling, canceled, failed
and finished-->
<!-- finished = successful execution-->
<started>
<!-- start time in long format, like 2015-09-30T13:03:56.980+02:00-->
<started_text>
<!-- start time in shortened human readable format, like 01:03:56 PM-->
<finished>
<!-- end time (if already terminated) in long format, like 2015-09-
30T13:03:56.980+02:00-->
<finished_text>
  <!-- end time in shortened human readable format, like 01:03:56 PM-->
```

Retrieve status overview:

```
GET - https://<server>:<port>/opr-web/rest/10.10/direct_
execution/<executionContextID>
```

Same as output for the above POST call (<tool_execution_context>).

Retrieve detailed status for one node:

```
GET - https://<server>:<port>/opr-web/rest/10.10/direct_
execution/<executionContextID>/<nodeSpecificResultId>

<tool_execution_result xmlns="http://www.hp.com/2009/software/opr/data_model"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- wrapper element for the detailed status-->
  <id>
  <!-- the nodeSpecificResultId-->
  <tool_ref>
  <!-- only relevant for tool web service, not for action web service.-->
  <execution_host>
  <!-- the host the command is running on-->
  <user_parameters/>
  <!-- only relevant for tool web service, not for action web service-->
  <tool_command>
  <!-- only available for tools of type om_agent_command. command string
executed on the host-->
  <tool_execution_state>
  <!-- can be: pending (not yet started), running, canceling, canceled, failed
and finished-->
  <!-- finished = successful execution-->
  <started>
  <!-- start time in long format, like 2015-09-30T13:03:56.980+02:00-->
  <started_text>
  <!-- start time in shortened human readable format, like 01:03:56 PM-->
  <finished>
  <!-- end time (if already terminated) in long format, like 2015-09-
30T13:03:56.980+02:00-->
```

```
<finished_text>  
<!-- end time in shortened human readable format, like 01:03:56 PM-->  
<output>  
<!-- command output-->  
<result_code>  
<!-- command return code-->
```

Examples

The following XML examples show you how to perform a sample workflow using the Action Web Service.

Submit a request (command "ovc") on three nodes, poll for the overall status, retrieve the detailed result as soon as the command has finished for a host.

1. Start the command:

```
POST - https://<server>:<port>/opr-web/rest/10.10/direct_execution  
<direct_execution xmlns="http://www.hp.com/2009/software/opr/data_model"  
xmlns:\br/>xs="http://www.w3.org/2001/XMLSchema">  
  <command>ovc</command>  
  <tool_type>om_agent_command</tool_type>  
  <execution_hosts>  
    <execution_host>mambo8.mambo.net</execution_host>  
    <execution_host>ia2.mambo.net</execution_host>  
    <execution_host>asdf</execution_host>  
  </execution_hosts>  
</direct_execution>
```

Output:

```
<tool_execution_context xmlns="http://www.hp.com/2009/software/opr/data_  
model" x  
mlns:xs="http://www.w3.org/2001/XMLSchema">  
  <id>ec172b83-277e-4853-90fa-ad7f49ad7262</id>  
  <tool_ref>  
    <id>79aa0a94-a717-4301-b22d-6c9728d52f93</id>  
    <target_id>79aa0a94-a717-4301-b22d-6c9728d52f93</target_id>  
    <label>DirectExecution</label>  
    <tool_type>om_agent_command</tool_type>  
    <has_user_parameter>>false</has_user_parameter>  
    <requires_run_as_user>>true</requires_run_as_user>  
  </tool_ref>  
  <user_parameters/>  
  <tool_execution_result_ref>
```



```
<id>d1714284-5d42-4b45-a95a-ad110555630d</id>
<execution_host>ia2.mambo.net</execution_host>
<tool_command>ovc</tool_command>
<tool_execution_state>running</tool_execution_state>
<started>2015-09-30T13:03:56.980+02:00</started>
<started_text>01:03:56 PM</started_text>
</tool_execution_result_ref>
<tool_execution_result_ref>
  <id>1c4b0db4-2e82-4657-a1c6-ecfa1d2a2e29</id>
  <execution_host>asdf</execution_host>
  <tool_command>ovc</tool_command>
  <tool_execution_state>failed</tool_execution_state>
  <started>2015-09-30T13:03:56.378+02:00</started>
  <started_text>01:03:56 PM</started_text>
  <finished>2015-09-30T13:03:56.397+02:00</finished>
  <finished_text>01:03:56 PM</finished_text>
</tool_execution_result_ref>
<tool_execution_result_ref>
  <id>3a39dd97-bee1-4acb-a595-18c8b4daeaad</id>
  <execution_host>mambo8.mambo.net</execution_host>
  <tool_command>ovc</tool_command>
  <tool_execution_state>running</tool_execution_state>
  <started>2015-09-30T13:03:56.984+02:00</started>
  <started_text>01:03:56 PM</started_text>
</tool_execution_result_ref>
</tool_execution_context>
```

2. Get a status overview for all nodes; use the execution context ID from the previous step.

```
GET - https://<server>:<port>/opr-web/rest/10.10/direct_execution/ec172b83-277e-4853-90fa-ad7f49ad7262
```

Check the output (also of type `<tool_execution_context>`) for terminated commands. Repeat the step until the XML tag `<tool_execution_state>` of one or more nodes changes to "finished" or "failed". To get the details, proceed to step 3. Continue polling the status until the last host is finished.

3. Get the detailed status of one node. Use the execution context ID and appropriate node-specific result ID.

```
GET - https://<server>:<port>/opr-web/rest/10.10/direct_execution/ec172b83-277e-4853-90fa-ad7f49ad7262/d1714284-5d42-4b45-a95a-ad110555630d
```

Output:

```
<tool_execution_result xmlns="http://www.hp.com/2009/software/opr/data_model" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <id>d1714284-5d42-4b45-a95a-ad110555630d</id>
  <tool_ref>
    <id>3269833c-1818-41dc-bd06-aafb6334088f</id>
    <target_id>3269833c-1818-41dc-bd06-aafb6334088f</target_id>
```

```

    <label>DirectExecution</label>
    <tool_type>om_agent_command</tool_type>
    <has_user_parameter>false</has_user_parameter>
    <requires_run_as_user>true</requires_run_as_user>
  </tool_ref>
  <execution_host>ia2.mambo.net</execution_host>
  <tool_command>ovc</tool_command>
  <user_parameters/>
  <tool_execution_state>finished</tool_execution_state>
  <started>2015-09-30T13:03:56.980+02:00</started>
  <started_text>01:03:56 AM</started_text>
  <finished>2015-09-30T13:03:58.292+02:00</finished>
  <finished_text>01:03:58 AM</finished_text>
  <output>coda          OV Performance Core          COREXT
(6983)
Running
ompolparm  OM Parameter Handler          AGENT,EA    (8308)
Running
opcacta    OVO Action Agent                    AGENT,EA    (6948)
Running
opcmona    OVO Monitor Agent                   AGENT,EA    (8275)
Running
opcmsga    OVO Message Agent                   AGENT,EA    (8333)
Running
ovbbccb    OV Communication Broker              CORE        (6897)
Running
ovcd       OV Control                           CORE        (6889)
Running
ovconfd    OV Config and Deploy                 COREXT      (6917)
Running
</output>
  <result_code>0</result_code>
</tool_execution_result>

```


Chapter 32: Downtime Web Service Interface

You can use a RESTful web service running on the Gateway Server to retrieve, update, create, and delete downtimes. HTTP requests can be entered in your browser, and combinations of HTTP requests and XML commands in a REST client. Service authentication is based on basic authentication.

For information about downtime and `opr-downtime` command-line interface, see the *OMi Administration Guide*.

Note: You require administration or superuser privileges to use downtime REST API.

`https://<serviceURL>/<request>`

<code><serviceURL></code>	The URL of the Action web service: <code><server>:<port>/topaz/bsmservices/customers/[customerID]/downtimes</code> where <code><server></code> is the name of the OMi server.
<code><request></code>	The service request.

Aces

The downtime REST service supports the following HTTP requests:

Note: CustomerID is always 1.

Request	Request Type	Action Description
<code>/downtimes/{[downtimeID]}</code>	GET	<code>/<downtimeID></code> is omitted: Lists all configured downtimes. <code>/<downtimeID></code> is specified: Get the downtime with ID <code><downtimeID></code> .
<code>/downtimes + XML of the downtime</code>	POST	Create an event category as specified in the posted XML data. Note: <code>/<categoryID></code> must be omitted.
<code>downtimes/[downtimeID] + XML of the downtime</code>	PUT	Update an event category as specified in the posted XML data. Note: <code>/<categoryID></code> must be specified.
<code>downtimes/[downtimeID]</code>	DELETE	Delete the event category with ID <code><categoryID></code> . Note: <code>/<categoryID></code> must be specified.

Request Reference

The downtime REST service supports the following HTTP requests:

Note: CustomerID is always 1.

Request	Request Type	Action Description
/downtimes/{[downtimeID]}	GET	<p><downtimeID> is omitted: Lists all configured downtimes.</p> <p><downtimeID> is specified: Get the downtime with ID <downtimeID>.</p>
/downtimes + XML of the downtime	POST	<p>Create an event category as specified in the posted XML data.</p> <p>Note: /<categoryID> must be omitted.</p>
downtimes/[downtimeID] + XML of the downtime	PUT	<p>Update an event category as specified in the posted XML data.</p> <p>Note: /<categoryID> must be specified.</p>
downtimes/[downtimeID]	DELETE	<p>Delete the event category with ID <categoryID>.</p> <p>Note: /<categoryID> must be specified.</p>

Allowed Downtime Actions

Use the XML commands listed for the following downtime actions:

Action Description	XML Command
Take no actions	<action name="REMINDER"/>
Suppress alerts and close events	<action name="SUPPRESS_NOTIFICATIONS"/>
Enforce downtime on KPI calculations; suppress alerts and close events (continue monitoring)	<action name="ENFORCE_ON_KPI_CALCULATION"/>
Enforce downtime on Reports and KPI calculations; suppress alerts and close events (continue monitoring)	<action name="ENFORCE_ON_REPORTS"/>

Action Description	XML Command
Enforce downtime on Reports and KPI calculations; suppress alerts and close events (continue monitoring), including all SLAs	<pre><action name="ENFORCE_ON_REPORTS"> <propGroup name="SLA" value="ALL"/> </action></pre>
Enforce downtime on Reports and KPI calculations; suppress alerts and close events (continue monitoring), including specific SLA	<pre><action name="ENFORCE_ON_REPORTS"> <propGroup name="SLA" value="SELECTED"> <prop>dda3fb0b20c0d83e078035ee1c005201</prop> </propGroup> </action></pre>
Stop active monitoring (BPM and SiteScope); enforce downtime on Reports & KPI calculations; suppress alerts and close events	<pre><action name="STOP_MONITORING"/></pre>

Downtime XML Example

The following fields may not exceed the maximum lengths specified:

- Name: 200 characters
- Description: 2000 characters
- Approved by: 50 characters

Note: In Oracle, if you are using East Asian Languages (Chinese, Japanese, or Korean), the maximum number of characters may be less than specified above.

```
<downtime userId="1" planned="true" id="8898e5a5dbc953e04037104bf5737c">
  <name>The name of the downtime</name>
  <action name="ENFORCE_ON_REPORTS">
  </action>
  <approver>The approver name</approver>
  <category>1</category>
  <notification>
    <recipients>
      <recipient id="24"/>
      <recipient id="22"/>
      <recipient id="21"/>
    </recipients>
  </notification>
  <selectedCIs>
    <ci>
      <id>ac700345b47064ed4fbb476f21f95a76</id>
      <viewName>End User Monitors</viewName>
    </ci>
  </selectedCIs>
</schedule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
xsi:type="WeeklyScheduleType">
  <type>WEEKLY</type>
  <startDate>2010-06-10T15:40:00</startDate>
  <timeZone>Europe/Zurich</timeZone>
  <days>
    <selectedDays>WEDNESDAY</selectedDays>
    <selectedDays>THURSDAY</selectedDays>
    <selectedDays>FRIDAY</selectedDays>
    <selectedDays>SATURDAY</selectedDays>
  </days>
  <startTimeInSecs>52800</startTimeInSecs>
  <durationInSecs>300</durationInSecs>
</schedule>
</downtime>
```

Downtime Schedule Examples

Keep the following in mind when setting the downtime schedule:

- Retroactive downtime is not supported. You cannot:
 - Create a downtime that is scheduled in the past.
 - Delete a downtime that has started or that occurred in the past.
 - Modify a downtime that has started or that occurred in the past.
- The date format of startDate/endDate is: **yyyy-MM-dd'T'HH:mm:ssZ**
- For weekly and monthly downtimes, the startDate and endDate should be defined at midnight. For example:
 - <startDate>2010-07-24T00:00:00</startDate>
 - <endDate>2010-09-04T00:00:00</endDate>

Example of a Downtime Schedule with One Occurrence

```
<schedule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="OnceScheduleType">
  <type>ONCE</type>
  <startDate>2010-06-08T14:40:00</startDate>
  <endDate>2010-06-08T14:45:00</endDate>
  <timeZone>Asia/Tokyo</timeZone>
</schedule>
```

Example of a Weekly Downtime Schedule

```
<schedule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="WeeklyScheduleType">
  <type>WEEKLY</type>
  <startDate>2010-06-10T15:40:00</startDate>
  <timeZone>Europe/Zurich</timeZone>
  <days>
```

```
<selectedDays>WEDNESDAY</selectedDays>  
<selectedDays>THURSDAY</selectedDays>  
<selectedDays>FRIDAY</selectedDays>  
<selectedDays>SATURDAY</selectedDays>  
</days>  
<startTimeInSecs>52800</startTimeInSecs>  
<durationInSecs>300</durationInSecs>  
</schedule>
```

Example of a Monthly Downtime Schedule

```
<schedule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="MonthlyScheduleType">  
  <type>MONTHLY</type>  
  <startDate>2010-06-10T14:40:00</startDate>  
  <timeZone>America/Montevideo</timeZone>  
  <days>  
    <selectedDays>WEDNESDAY</selectedDays>  
    <selectedDays>THURSDAY</selectedDays>  
    <selectedDays>FRIDAY</selectedDays>  
    <selectedDays>SATURDAY</selectedDays>  
  </days>  
  <startTimeInSecs>52800</startTimeInSecs>  
  <durationInSecs>300</durationInSecs>  
</schedule>
```

Downtime REST Examples

The Java code examples below were designed to help use the Downtime REST service API. These examples use only standard Java components. For each invoked operation, the sever returns HTTP code which can be used for operation verification on the client side.

The Java Code Downtime REST Examples listed below are available in .txt format in the following directory:

`<OMi_HOME_GW>/AppServer/webapps/site.war/amdocs/eng/API_docs/DowntimeREST_JavaAPI/`

- **CreateDowntime.java**

This is an example of Java code to create a new Downtime, which uses the HTTP **POST** request, (in REST services, POST request is used to create an entity). If the operation runs successfully, the system returns a newly created Downtime in XML format, including the downtime ID.

- **DeleteDowntime.java**

This is an example of Java code to delete a specific Downtime, which uses the HTTP **DELETE** request (in REST services, DELETE request is used to delete an entity). If the operation runs successfully, nothing will be returned.

- **GetAllDowntimes.java**

This is an example of Java code to get all Downtimes, which uses the HTTP **GET** request, (in REST services, GET request is used to get an entity). If the operation runs successfully, the system returns all Downtimes in XML format.

- **GetSpecificDowntime.java**

This is an example of a Java file to get all of a specific Downtime, which uses the HTTP **GET** request, (in REST services, GET request is used to get an entity). If the operation runs successfully, the system will return a specific Downtime in XML format.

- **UpdateDowntime.java**

This is an example of Java code to update a Downtime, which uses the HTTP **PUT** request, (in REST services, PUT request is used to update an entity). If the operation runs successfully, the system will not return anything.

Importing Downtime Data From an External Source

If business management solutions (such as Service Manager, OM, or third party software) create downtime events when integrating with OMi, you may need to import downtime information from an external system. To import this downtime information, create a middle utility using the REST API to pull the events from the external source and post them to OMi.

When importing definitions from an external source, take into account both the import scope and mechanism.

Import Scope

Downtime properties may be different in different systems and software platforms. The common set of downtime properties includes scheduling information and configuration items. In OMi downtime, the mandatory fields are:

- Downtime Name
- CI ID
- Schedule
- Action

Imported events must be translated to match OMi downtime properties.

Import Mechanism

Importing downtimes into OMi is performed by an external utility with access to the formats and properties of the external source. This utility translates the external properties to correlate to the required and optional OMi downtime properties in XML format.

Import Example

An example utility created using Java and Groovy in conjunction with the REST API can be downloaded from:

```
<OMi_HOME_GW>/AppServer/webapps/site.war/amdocs/eng/API_docs/DowntimeREST_
JavaAPI/DTImport.zip
```

The DTImport.zip file includes:

- **DTImport.jar** – Contains Java classes that run the Groovy script and provide script dependencies (mainly the HTTP client that is used to access the REST services).
- **DTImport.bat** – Runs the Java application and sends the OMi URL and user name and password.

- **downtimeFiles folder** – Contains XML files with the downtime definitions.
- **DTImport.groovy** – Groovy script that reads and posts the XML files to the OMi REST service.

The DTImport.bat file invokes the DTImport Java application with a defined integration folder (system property integ.home). The application reads all the Groovy scripts in the scripts folder and invokes them. The example DTImport.groovy script reads all the dt[n].xml files and uses the OMi REST service to create downtime in OMi. You can alter the contents of this file to create your own custom integrations.

Keep the following in mind when editing the file:

- To alter the logic of the utility, edit the Groovy script and run the batch file again. You do not need to build and compile the script.
- To change the integration directory, edit **integ.home** in the batch file.
- You can replace the **DTImport.groovy** script with any other script in the Scripts folder.
- Only add Groovy scripts with a main method to the Scripts folder.
- Only add well formatted XML files to the downtimeFiles folder.
- Use any scheduler (such as Windows Task Scheduler) to run the script.

To run the example:

1. Verify that the Java Runtime environment is installed on the client machine.
2. Extract the DTImport.zip file to the server on which you want to run the integration. This can be the OMi server, the external source, or any other server.
3. Edit the DTImport.bat file to update the OMi URL and credentials.
4. Edit the XML files in the downtimeFiles folder to update the downtime parameters you want to translate to match your system definitions.
5. To enter an existing recipient ID:
 - a. Enter the URL of the JMX console (<https://localhost:29000>) in a web browser.
 - b. Enter your user name and password.
 - c. Retrieve the recipient ID by executing **Test Notification Service > showRecipients()**.
6. To enter existing CI IDs:
 - a. Enter the URL of the DPS JMX console (<http://<Data Processing Server name>:21212/jmx-console>) in a web browser.]
 - b. Enter your user name and password.
 - c. Retrieve the CIIDs by executing **Model Services > retrieveObjectsOfType**. For example, if the CI is a Business application, enter `business_application`.
7. Run the batch file.

Chapter 33: Event Synchronization Web Service Interface

The Event Synchronization Web Service interface can be used to integrate with external (third-party) event processes, such as an incident manager like HPE Service Manager. The interface is used to forward events and event changes from the OMi OPR client to the third-party application, and to synchronize back from the third-party client events and event changes that the external application makes to the events.

An external event process may integrate directly with OMi event processing, through the implementation of an OPR-compliant Event Synchronization Web Service, as opposed to implementing a Groovy script. The external application would need to implement a web service endpoint to receive OPR events and event changes, and, if synchronization is supported by the external application, a REST web service client to submit events and synchronize back event changes to the Event Synchronization Web Service.

(prerequisite) Configure a Connected Server

To use this web service, the following steps must first be performed:

1. Configure a Connected Server of type `External Event Processing` in the Connected Servers manager. The name of the server is the name that must be used when authenticating at the Event Synchronization Web Service. You also specify a password for this user during configuration of the connected server.
2. Forward an event to the target connected server. This can be done by using the Event Browser context menu to manually forward an event, or by configuring a forwarding rule and then triggering that rule.
3. Until an event is forwarded to the target connected server, the server cannot synchronize back any changes. In contrast to the Event Web Service (see "[Event Web Service Interface Reference](#)" on page 158), which allows the authenticated OMi user to view and update any event that the OMi user has access to, the connected server is only allowed to synchronize back changes for events that have been first forwarded to it.
4. The connected server synchronizes back changes to the event.

For details of how to configure connected servers, see OMi Help.

Forward Events and Event Changes from OPR Client

For the OPR client to be able to forward events and event changes to a third-party event processing application, that third-party application must implement an OPR-compliant Event Synchronization Web Service.



Note: You must configure a connected server before using this web service. For details, see "[prerequisite\) Configure a Connected Server](#)" on the previous page. The <base configured URL of the connected server> is the path to the server you specified during the initial connected server setup.

Forward Event

When forwarding events from the OPR client to a third-party application, the following data is relevant:

- HTTP Method: POST
- Base URL: `http://<base configured URL of the connected server>/event`
- Expected Payload: The expected response payload is an `OprEvent` object.

Forward Event Change

When forwarding event changes from the OPR client to a third-party application, the following data is relevant:

- HTTP Method: POST
- Base URL: `http://<base configured URL of the connected server>/event_change/<external_event_ID>`
- Expected Payload: The expected response payload is an `OprEventChange` object.

Forward Events

When forwarding events in bulk from the OPR client to a third-party application, the following data is relevant:

- HTTP Method: POST
- Base URL: `http://<base configured URL of the connected server>/event`
- Expected Payload: The payload is an `OprEventList` object.
- Payload returned: Upon success the target service must return a list of the events sent to it with the ID set to the external Event ID. The `sequence_number` must match the `sequence_number` of the event sent in the POST. For those events which do not have a matching event returned, they are retried in a subsequent request. If an event is to be rejected, an HTTP error must be returned for the entire payload. If an HTTP error 500-599 is returned, each event is individually retried as a non-bulk request. For an HTTP error 400-499, all events in the request are marked as failed and no retry is done.

Forward Event Changes

When forwarding event changes in bulk from the OPR client to a third-party application, the

following data is relevant:

- HTTP Method: POST
- Base URL: `http://<base configured URL of the connected server>/event_change/`
- Expected Payload: The expected response payload is an `OprEventChangeList` object. Each `OprEventChange` item in the list must have the `event_ref` property set. The `global_id` is expected to be set to the ID on the target system, for example, when OMi calls this web service on the target system, the `target_id` is set to the OMi event ID and the `target_global_id` is set to the target object ID.
- Payload returned: Upon success the target service must return a list of the event changes sent to it with the same `sequence_number`. The `sequence_number` must match the `sequence_number` of the event change sent in the POST. For those changes which do not have a matching change returned, they are retried in a subsequent request. If an event change is to be rejected, an HTTP error must be returned for the entire payload. If an HTTP error 500-599 is returned, each event change is individually retried as a non-bulk request. For an HTTP error 400-499, all event changes in the request are marked as failed and no retry is done.

Web Service GET Request

When there is a need to get the external event from the third-party application (for example, when the External Info tab is selected from the Event Browser), the following data is relevant:

- HTTP Method: GET
- Base URL: `http://<base configured URL of the connected server>/event/<external_event_ID>`
- Expected Payload: The payload is the external event in the form of an `OprEvent`.

Web Service Ping Request

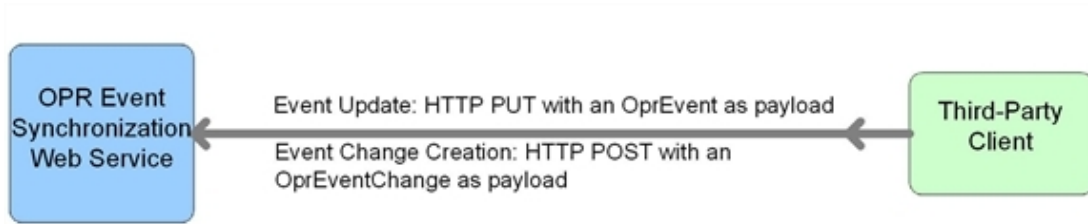
You can send a ping request to a third-party application. When sending a ping request to a third-party application, the following data is relevant:

- HTTP Method: HEAD to base URL
- Base URL:
 - Base URL for forwarding events: `http://<base configured URL of the connected server>`
- Expected Payload: None

Synchronize Event Changes Back from External Client

If synchronization is supported by the external application, a REST web service client is needed to synchronize back event changes to the Event Synchronization Web Service. The payload in this back synchronization is expected to comply with native OPR objects. With a Groovy script configured for connected server, the Groovy script interprets the payload that is defined by the client.

If a client wants to submit a new event or a change to an event, but not invoke the Groovy script, the sub-paths `event_list` or `event_change_list` must be used. These paths will require native OPR objects (for example, `OprEvent` or `OprEventChangeList`). The sub-paths `event_list` and `event_change_list` also enable clients to submit one or more events or event changes.



Event Updates

Event Get

In the case of event get, the following data is relevant:

- HTTP Method: GET
- Base URL: `https://<server.example.com>/opr-gateway/rest/9.10/synchronization/event/<event_id>`
- Payload returned:
 - With a Groovy script configured for the connected server, the Groovy script `toExternalEvent()` method defines the payload.
 - Without a Groovy script configured for the connected server, the payload is an `OprEvent` object.

Event Update

In the case of event updates, the following data is relevant:

- HTTP Method: PUT
- Base URL: `https://<server.example.com>/opr-gateway/rest/9.10/synchronization/event/<event_id>`
- Ping request (from third-party client): HTTP HEAD to base URL
- Payload received:
 - With a Groovy script configured for the connected server, the Groovy script defines the payload.
 - Without a Groovy script configured for the connected server, the payload must be an `OprEvent` object.

See also "[Event Update: Logfile Adapter Examples](#)" on page 281.

Event Change Creation

In the case of event changes, the following data is relevant:

- HTTP Method: POST
- Base URL: `https://<server.example.com>/rest/9.10/synchronization/event_change/<event_id>`
- Ping request (from third-party client): HTTP HEAD to base URL

- Payload received:
 - With a Groovy script configured for the connected server, the Groovy script defines the payload.
 - Without a Groovy script configured for the connected server, the payload must be an `OprEventChange` object.

See also ["Event Change Creation: Logfile Adapter Examples" on page 283](#).

Web Service Ping Request

You can also send a ping request *from* a third-party application. When sending a ping request from a third-party application, the following data is relevant:

- HTTP Method: HEAD
- Base URL: `https://<server.example.com>/opr-gateway/rest/9.10/synchronization`
- Expected Payload: None

Event List Updates

The following URLs point to the sub-path `event_list`. Requests to this sub-path do not call the Groovy script and always have input or output of OPR data structures, for example, `OprEvent` and `OprEventList`.

Event Get

In the case of event get, the following data is relevant:

- HTTP Method: GET
- Base URL: `https://<server.example.com>/opr-gateway/rest/9.10/synchronization/event_list/<event_id>`
- Payload returned:
 - The payload is always an `OprEvent` object.
 - The Groovy script is not called for the sub-path `event_list`.

Event Update

In the case of event updates, the following data is relevant:

- HTTP Method: PUT
- Base URL: `https://<server.example.com>/opr-gateway/rest/9.10/synchronization/event_list/<event_id>`
- Payload received and returned:
 - The payload received must be an `OprEvent` object.
 - The payload returned is always an `OprEvent` object.
 - The Groovy script is not called for the sub-path `event_list`.

See also ["Event Update to event_list Example" on page 291](#).

Event Submit

You can also submit an event *from* a third-party application. When submitting an event from a third-party application, the following data is relevant:

- HTTP Method: POST
- Base URL: `https://<server.example.com>/opr-gateway/rest/9.10/synchronization/event_list`
- Payload received:
 - The payload must be an `OprEvent` or `OprEventList` object.
 - For an `OprEventList` object, the media type must be `text/xml;type=collection` or `application/xml;type=collection`.
 - The Groovy script is not called for the sub-path `event_list`.

See also ["Event Submit Examples" on page 292](#).

Event List Changes

The following URLs point to the sub-path `event_change_list`. Requests to this sub-path do not call the Groovy script and always have input or output of OPR data structures, for example, `OprEvent` and `OprEventList`.

Event Change Creation

In the case of event changes to the sub-path `event_change_list`, the following data is relevant:

- HTTP Method: POST
- Base URL: `https://<server.example.com>/opr-gateway/rest/9.10/synchronization/event_change_list/`
- Payload received:
 - The payload received must be an `OprEventChange` or `OprEventChangeList` object..
 - For an `OprEventChangeList` object, the media type must be `text/xml;type=collection` or `application/xml;type=collection`.
 - The Groovy script is not called for the sub-path `event_change_list`.

See also ["Event Change Creation for event_change_list Examples" on page 293](#).

Event Attributes that Support Back Synchronization

Supported Attribute Updates

Attribute Name	Supported Operations		
	Update	Create	Delete
annotation	Yes	Yes	
assigned_user	Yes		set to null
assigned_group	Yes		set to null
cause	Yes		set to null
custom attribute	Yes	Yes	
description	Yes	Yes	set to null
priority	Yes		
severity	Yes		
solution	Yes		
state	Yes		
title	Yes		set to null
control_transferred_to	Yes		set to null

Event Update: Logfile Adapter Examples

Note: The examples showing event updates in this section are specific to the Logfile Adapter (the sample Groovy script adapter provided with the product), and would work only for the Logfile Adapter, or for accessing the Event Synchronization Web Service directly without having a Groovy script configured for the target connected server.

You can submit event changes with an event update. The HTTP method in this case is PUT, and the expected payload for the Logfile Adapter is an `OprEvent`. Each Groovy script adapter defines its own expected payload, so the expected payload would be different for other Groovy script adapters.

The Event Synchronization Web Service supports sending multiple properties in a single payload. The following examples each show just one changed property, with the exception of the last sample, which shows two changed properties (see ["Change Multiple Properties in One Call" on page 283](#)).

You can send test payloads to the Event Synchronization Web Service using the REST web service command-line utility `RestWsUtil`, supplied with the product. When using the utility, make sure that the event you are trying to update was first forwarded to the target connected server you are using to authenticate with the Event Synchronization Web Service.

For details about the RestWsUtil command-line utility, see ["RestWsUtil Command-Line Interface" on page 190](#).

An example call for an event update is shown below. The following sample call for an event update is for a connected server named `logger` with password set to `Password1`, and attempts to update an event with the ID `0695624b-93fa-40b1-8b0fc9b4ea07a4ec`. The sample call looks like this:

```
RestWsUtil -update update.xml-url "https://<server.example.com>/opr-gateway/rest/9.10/synchronization/event/0695624b-93fa-40b1-8b0fc9b4ea07a4ec" -username logger -password Password1 -verbose
```

The samples that follow show possible XML payloads for the call. The payloads are contained in the `update.xml` file referenced in the call to the `RestWsUtil` command-line utility.

Note: An Event REST WS sample is located at:

```
<OMi_HOME>/opr/examples/event-ws
```

This can be compiled and packaged into a war file for deployment into a J2EE container. See the `README.txt` in this directory for instructions on how to build this webapp and deploy it for testing.

A Event REST WS template is located at:

```
<OMi_HOME>/opr/examples/synchronization-ws
```

This can be used as a starting point for developers who want to develop their own Event Synchronization web service. See the `README.txt` in this directory for instructions on how to build this webapp. The "TODO" sections must be completed by the developer.

Set and Unset the Event Description

The Event Synchronization Web Service supports only the set or unset operations for the description of an event. Insert, update, or delete operations are not supported.

Example: Set the Description

Here is a sample payload to set the description:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">
  <description>This is a new description</description>
</event>
```

Example: Unset the Description

Here is a sample payload to unset the description:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">
  <description/>
</event>
```

Set and Unset the Event Title

The Event Synchronization Web Service supports only the set or unset operations for the title of an event. Insert, update, or delete operations are not supported.

Example: Set the Title

Here is a sample payload to set the title:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <title>This is a new title</title>  
</event>
```

Example: Unset the Title

Here is a sample payload to unset the title:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <title/>  
</event>
```

Change Multiple Properties in One Call

The Event Synchronization Web Service supports sending multiple properties in a single payload. Here we show just two changed properties.

Example: Set Title and Description

The title and description can be set in a single call. The order of the attributes is unimportant and attributes not specified do not affect the current values.

Here is a sample payload to set the event title and description in a single update call:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <title>This is a new title</title>  
  <description>This is a new description</description>  
</event>
```

Event Change Creation: Logfile Adapter Examples

You can submit event changes with an event change creation. The HTTP method in this case is POST, and the expected payload for the LogfileAdapter is an `OprEventChange`. The examples in this section show sample XML payloads for each supported update. If a Groovy script is not configured for the connected server, the Event Synchronization Web Service expects an `OprEventChange` object. Each of the samples that follow represent an `OprEventChange` object.

The Event Synchronization Web Service supports sending multiple properties in a single payload. The following examples each show just one changed property, with the exception of the last sample, which shows two changed properties (see ["Change Multiple Properties in One Call" on page 291](#)).

The examples use an event ID of 531d2673-683f-429f-a742-b8680ee01a76. Change this event ID to the ID of the specific event for which you wish to create a change object. The correct HTTP method for the creation of an event change is the POST method.

You can also use the event change list web service to get examples of event change objects. You can access the event change list web service using this URL:

```
https://<server.example.com>/opr-web/rest/9.10/event_change_list
```

An example call for an event change creation is shown below.

The following sample call for an event change creation is for a connected server named `logger` with password set to `Password1`, and attempts to create an event change for an event with ID `531d2673-683f-429f-a742-b8680ee01a76`. For an event change, the event ID must be passed in the payload, and is not part of the URL. The sample call looks like this:

```
RestWsUtil -create create.xml-url "https://<server.example.com>/opr-gateway/rest/9.10/synchronization/event_change/<event id>" -username logger -password Password1 -verbose
```

The samples that follow show possible XML payloads for the call. The payloads are contained in the `create.xml` file referenced in the call to the `RestWsUtil` command-line utility.

Annotations

The Event Synchronization Web Service supports only insert and update operations for annotations. The delete operation is not supported.

Example: Insert an Annotation

Here is a sample payload showing an annotation insert:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
  </event_ref>
  <changed_properties>
    <annotation_property_change>
      <property_name>annotation</property_name>
      <current_value xsi:type="xs:string">This is a new Annotation.</current_value>
      <change_operation>insert</change_operation>
    </annotation_property_change>
  </changed_properties>
</event_change>
```

Example: Update an Annotation

Here is a sample payload showing an update to an annotation with the ID of `1c108839-9584-45f4-93ab-798bf49797c7`:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
  </event_ref>
  <changed_properties>
    <annotation_property_change>
```

```
<property_name>annotation</property_name>
<annotation_id>1c108839-9584-45f4-93ab-798bf49797c7</annotation_id>
<current_value xsi:type="xs:string">This is an updated
Annotation.</current_value>
<change_operation>update</change_operation>
</annotation_property_change>
</changed_properties>
</event_change>
```

Cause

The Event Synchronization Web Service supports only the set or unset operations for the cause of an event. Insert, update, or delete operations are not supported. You set or unset the cause on the symptom event.

Example: Set the Cause of an Event

Here is a sample payload showing how to set the cause of an event to be the event with the ID of 9915e504-f5a2-471a-ab98-6184a7382d32:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
  </event_ref>
  <changed_properties>
    <string_property_change>
      <property_name>cause</property_name>
      <current_value xsi:type="xs:string">9915e504-f5a2-471a-ab98-
6184a7382d32</current_value>
    </string_property_change>
  </changed_properties>
</event_change>
```

Example: Unset the Cause of an Event

Here is a sample payload showing how to unset the cause of an event:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
  </event_ref>
  <changed_properties>
    <string_property_change>
      <property_name>cause</property_name>
      <current_value xsi:type="xs:string"/>
    </string_property_change>
  </changed_properties>
</event_change>
```

```
</string_property_change>  
</changed_properties>  
</event_change>
```

Custom Attributes

The Event Synchronization Web Service supports only insert and update operations for custom attributes. The delete operation is not supported.

Example: Insert a Custom Attribute

Here is a sample payload showing how to insert a custom attribute, `custom_attribute`, with the value `Some CA value`:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <event_ref>  
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>  
  </event_ref>  
  <changed_properties>  
    <custom_attribute_property_change>  
      <property_name>custom_attribute</property_name>  
      <key>TestCA</key>  
      <current_value xsi:type="xs:string">Some CA value</current_value>  
      <change_operation>insert</change_operation>  
    </custom_attribute_property_change>  
  </changed_properties>  
</event_change>
```

Example: Update a Custom Attribute

Here is a sample payload showing how to update a custom attribute, `custom_attribute`, with the value `Some updated CA value`:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <event_ref>  
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>  
  </event_ref>  
  <changed_properties>  
    <custom_attribute_property_change>  
      <property_name>custom_attribute</property_name>  
      <key>TestCA</key>  
      <current_value xsi:type="xs:string">Some updated CA value</current_value>  
      <change_operation>update</change_operation>  
    </custom_attribute_property_change>  
</event_change>
```

```
</changed_properties>  
</event_change>
```

Description

The Event Synchronization Web Service supports only the set or unset operations for the description of an event. Insert, update, or delete operations are not supported.

Example: Set the Description

Here is a sample payload showing how to set the description in an event change creation to Some description of the event:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <event_ref>  
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>  
  </event_ref>  
  <changed_properties>  
    <string_property_change>  
      <property_name>description</property_name>  
      <current_value xsi:type="xs:string">Some description of the event.  
    </current_value>  
    </string_property_change>  
  </changed_properties>  
</event_change>
```

Example: Unset the Description

Here is a sample payload showing how the description in an event change creation is unset:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <event_ref>  
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>  
  </event_ref>  
  <changed_properties>  
    <string_property_change>  
      <property_name>description</property_name>  
      <current_value xsi:type="xs:string"/>  
    </string_property_change>  
  </changed_properties>  
</event_change>
```

State

The only operation that is possible on the state of an event is to give the state a new value. Insert, update, or delete operations are not applicable for event state changes.

The possible values for the state of an event are: open, in_progress, resolved, closed.

Example: Set the State

Here is a sample payload showing how the state in an event change creation is set to the value in_progress:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
  </event_ref>
  <changed_properties>
    <state_change>
      <property_name>state</property_name>
      <current_value xsi:type="xs:string">in_progress</current_value>
    </state_change>
  </changed_properties>
</event_change>
```

Priority

The only operation that is possible on the priority of an event is to give the priority a new value. Insert, update, or delete operations are not applicable for priority changes.

You can set the priority to one of the following values: none, lowest, low, medium, high, highest.

Example: Set the Priority

Here is a sample payload showing how the priority in an event change creation is set to low:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
  </event_ref>
  <changed_properties>
    <priority_property_change>
      <property_name>priority</property_name>
      <current_value xsi:type="xs:string">low</current_value>
    </priority_property_change>
  </changed_properties>
</event_change>
```

Severity

The only operation that is possible on the severity of an event is to give the severity a new value. Insert, update, or delete operations are not applicable for severity changes.

You can set the severity to one of the following values: critical, major, minor, warning, normal.

Example: Set the Severity

Here is a sample payload showing how the severity in an event change creation is set to normal:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
  </event_ref>
  <changed_properties>
    <severity_property_change>
      <property_name>severity</property_name>
      <current_value xsi:type="xs:string">normal</current_value>
    </severity_property_change>
  </changed_properties>
</event_change>
```

Solution

The Event Synchronization Web Service supports only the set or unset operations for the solution of an event. Insert, update, or delete operations are not applicable for solution changes.

Example: Set a Solution

Here is a sample payload showing how the solution in an event change creation is set to Some solution to the event:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
  </event_ref>
  <changed_properties>
    <string_property_change>
      <property_name>solution</property_name>
      <current_value xsi:type="xs:string">Some solution to the event.</current_
value>
    </string_property_change>
  </changed_properties>
</event_change>
```

Example: Unset a Solution

Here is a sample payload showing how the solution in an event change creation is unset:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
```

```
<target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
</event_ref>
<changed_properties>
  <string_property_change>
    <property_name>solution</property_name>
    <current_value xsi:type="xs:string"/>
  </string_property_change>
</changed_properties>
</event_change>
```

Title

The Event Synchronization Web Service supports only the set or unset operations for the title of an event. Insert, update, or delete operations are not applicable for title changes.

Example: Set the Title

Here is a sample payload showing how the title in an event change creation is set to Some title for the event:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
  </event_ref>
  <changed_properties>
    <string_property_change>
      <property_name>title</property_name>
      <current_value xsi:type="xs:string">Some title for the event.</current_
value>
    </string_property_change>
  </changed_properties>
</event_change>
```

Example: Unset the Title

Here is a sample payload showing how the title in an event change creation is unset:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
  </event_ref>
  <changed_properties>
    <string_property_change>
      <property_name>title</property_name>
      <current_value xsi:type="xs:string"/>
    </string_property_change>
  </changed_properties>
</event_change>
```

```
</string_property_change>  
</changed_properties>  
</event_change>
```

Change Multiple Properties in One Call

The Event Synchronization Web Service supports sending multiple properties in a single payload. Here, we show two changed properties.

Example: Set the Title and Description

Here is a sample payload to set the event title and description in a single update call:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <event_ref>  
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>  
  </event_ref>  
  <changed_properties>  
    <string_property_change>  
      <property_name>title</property_name>  
      <current_value xsi:type="xs:string">Some title for the event.</current_  
value>  
    </string_property_change>  
    <string_property_change>  
      <property_name>description</property_name>  
      <current_value xsi:type="xs:string">Some description for the  
event.</current_value>  
    </string_property_change>  
  </changed_properties>  
</event_change>
```

Event Update to event_list Example

You can update events using OPR data structure only (and not call the Groovy script). To update events using OPR data structures, call update using the event_list sub-path. The payload is the same as, for example, the standard put.

The title and description can be set in a single call as follows. The order of the attributes is unimportant and attributes not specified do not affect the current values.

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">  
  <title>This is a new title</title>  
  <description>This is a new description</description>  
</event>
```

Event Submit Examples

You can submit new events using the URL `https://<server.example.com>/opr-gateway/rest/9.10/synchronization/event_list`. The HTTP method in this case is the POST method and the expected payload is an `OprEvent` or `OprEventList`. For an `OprEventList` object, the media type must be `text/xml;type=collection` or `application/xml;type=collection`.

If the submitter does not add an `OprForwardingInfo` entry for itself in the event, one will be automatically added with `ForwardingType` set to `notify`. This allows the caller to subsequently read the event from the Event Synchronization Web Service.

The following samples show the XML payload. The correct HTTP method to set is the POST method.

Submit a New Event

Here is a sample payload showing submittal of a single event:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">
  <title>This is a new event</title>
  <description>This is a description</description>
</event>
```

Submit a New Event with Synchronization Requested

Here is a sample payload showing an event submittal with synchronization requested:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">
  <title>This is a new event</title>
  <description>This is a description</description>
  <forwarding_info_list>
    <forwarding_info>
      <dns_name>extclt.example.com</dns_name>
      <external_id>IM10219</external_id>
      <external_url>http://extclt.example.com:8081/webtier-
9.20/index.do?ctx=docEngine&file=probsummary&query=number%3D%22IM10219%22</externa
l_url>
      <forwarding_type>synchronize</forwarding_type>
      <state>originated</state>
    </forwarding_info>
  </forwarding_info_list>
</event>
```

Submit a New Event with Synchronization and Transfer Control Requested

Here is a sample payload showing an event submittal with synchronization and transfer control requested:

```
<event xmlns="http://www.hp.com/2009/software/opr/data_model">
  <title>This is a new event</title>
  <description>This is a description</description>
  <forwarding_info_list>
    <forwarding_info>
      <dns_name>extclt.example.com</dns_name>
      <external_id>IM10219</external_id>
      <external_url>http://extclt.example.com:8081/webtier-
9.20/index.do?ctx=docEngine&file=probsummary&query=number%3D%22IM10219%22</externa
l_url>
      <forwarding_type>synchronize_and_transfer_control</forwarding_type>
      <state>originated</state>
    </forwarding_info>
  </forwarding_info_list>
</event>
```

Submit a List of New Events

The media type for `OprEventList` objects must be `text/xml;type=collection` or `application/xml;type=collection`.

Here is a sample payload showing submittal of a list of events:

```
<event_list xmlns="http://www.hp.com/2009/software/opr/data_model">
  <event><title>e0</title><severity>critical</severity></event>
  <event><title>e1</title><severity>normal</severity></event>
  <event><title>e2</title><severity>major</severity></event>
  <event><title>e3</title><severity>minor</severity></event>
  <event><title>e4</title><severity>warning</severity></event>
</event_list>
```

Event Change Creation for `event_change_list` Examples

You can submit new events changes using the URL `https://<server.example.com>/opr-gateway/rest/9.10/synchronization/event_change_list`. The HTTP method in this case is the POST method and the expected payload is an `OprEventChange` or `OprEventChangeList`. For an `OprEventChangeList` object, the media type must be `text/xml;type=collection` or `application/xml;type=collection`.

The following samples show the XML payload. The correct HTTP method to set is the POST method.

Submit a New Event Change

Here is a sample payload showing submittal of a title change:

```
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
```

```
</event_ref>
<changed_properties>
  <string_property_change>
    <property_name>title</property_name>
    <current_value xsi:type="xs:string">Some title for the event.</current_
value>
  </string_property_change>
</changed_properties>
</event_change>
```

Submit a List of New Event Changes

The media type must be `text/xml;type=collection` or `application/xml;type=collection`.

Here is a sample payload showing submittal of a title change for one event, and a state change for another event:

```
<event_change_list xmlns="http://www.hp.com/2009/software/opr/data_model">
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>531d2673-683f-429f-a742-b8680ee01a76</target_id>
  </event_ref>
  <changed_properties>
    <string_property_change>
      <property_name>title</property_name>
      <current_value xsi:type="xs:string">Some title for the event.</current_
value>
    </string_property_change>
  </changed_properties>
</event_change>
<event_change xmlns="http://www.hp.com/2009/software/opr/data_model"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <event_ref>
    <target_id>ad726f4f-ba51-409f-b429-b445b791ac9d</target_id>
  </event_ref>
  <changed_properties>
    <state_change>
      <property_name>state</property_name>
      <current_value xsi:type="xs:string">in_progress</current_value>
    </state_change>
  </changed_properties>
</event_change>
</event_change_list>
```

Chapter 34: Event Web Service Interface

The REST-based Event Web Service enables integrators to programmatically automate operator functions and detect event changes. Most operations that an operator can do in the console while working on events can be done programmatically, to improve efficiency, and enable integration with external applications.

For detailed information about the Event Web Service, see ["Automating Operator Functions and Event Change Detection" on page 157](#).

For general information about using web service interfaces such as the Event Web Service Interface with OMi, see ["Reference Information For All Web Services" on page 248](#).

Chapter 35: Monitoring Automation Web Service Interface

The REST- based Monitoring Automation Web Service enables integrators to access the following Monitoring Automation functionality from an external application:

- List all management template, aspect, or policy template assignments.
- List all assignments of a certain management template, aspect, or policy template.
- List all management template, aspect, or policy template assignments to a certain CI.
- List all management templates and aspects assignable to a certain CI.
- List all deployment jobs created as a result of an assignment.
- Find status and parameter information for a certain assignment of a management template, aspect, or policy template.
- Create, update, and delete assignments of management templates, aspects, or policy templates.
- List available management templates, allowing filtering by CI type or management template ID.
- List available aspects, allowing filtering by CI type or aspect ID.
- List available policy templates, allowing filtering by template ID.

Note: Only direct assignments, not indirect assignments, are listed by the Monitoring Automation web service.

The following sections provide details on how to use the Monitoring Automation Web Service:

- ["Using the Monitoring Automation Web Service Interface" below](#)
- ["Monitoring Automation Web Service Interface Reference" on page 302](#)
- ["Examples" on page 307](#)

For general information about using web service interfaces such as the Monitoring Automation Web Service Interface with OMi, see ["Reference Information For All Web Services" on page 248](#).

Using the Monitoring Automation Web Service Interface

This section describes several aspects to be considered when using the Monitoring Automation Web Service Interface.

Logging

All actions taken by the web service are logged in the following log files:

<OMi_HOME>/log/jboss/opr-webapp.log

<OMi_HOME>/log/jboss/opr-configserver.log

Object References Embedded in the Response

Objects returned in the response include several references that you can use in subsequent requests involving the referenced object.

id and `<target_id>`

The XML tags `id` and `<target_id>` represent the object ID and the ID of a referenced object, respectively.

self and `<target_href>`

The XML tags `self` and `<target_href>` represent the object URL and the URL of a referenced object, respectively.

type and `<target_type>`

The XML tags `type` and `<target_type>` represent a reference for the object and a reference for a referenced object, respectively.

`<link>`

The XML tag `<link>` specifies additional navigation possibilities such as requesting verbose output, requesting a list of deployment jobs for the assignment being created, and so on.

Example: If the response to a GET request is a list of management template versions, each management template version in the list corresponds to a `<management_template_version_ref version="9.20" />` node. The management template node looks similar to the following output:

```
<management_template_version_ref version="9.20"
  type="urn:x-hp:2009:software:data_model:opr:type:reference:
    management_template_version">

  <target_id>d6f9cf24-244a-bce7-1e37-c5446c81e773</target_id>
  <target_type>urn:x-hp:2009:software:data_model:opr:type:
    management_template_version</target_type>
  <target_href>
    https://server.example.com/opr-config-server/rest/ws/9.20/
    management_template_version_list/d6f9cf24-244a-bce7-1e37-c5446c81e773
  </target_href>

  <display_label>MTCIAAttResolution</display_label>
  <major_version>1</major_version>
  <minor_version>0</minor_version>

</management_template_version_ref>
```

The `type`, `<target_id>`, `<target_type>`, `<target_href>` tags are highlighted in bold type.

Looping through the objects in the response, you can use the references to create new requests for the individual objects, for example to retrieve all assignments of a certain management template version.

Parameters

This section explains what parameters are, and how to set them when assigning management templates, aspects, or policy templates to CIs.

Considerations

When executing web service requests, the following should be taken into consideration with regard to Monitoring Automation (MA) parameters:

- Parameter values can be of the type `string`, `numeric`, `password`, or `enum`.
- Certain XML tags are only visible in verbose mode. Examples of these are the following tags:
 - Possible enumeration values of `enum` parameters.
 - Minimum and maximum values of `numeric` parameters.
 - Expert flag.
- The web service returns parameters only in response to web service requests dealing with assignments. To get a list of parameters contained in a management template, aspect, or policy template, request a draft assignment for the artifact.
- The response to a `GET /assignment_list/draft/ci/<CIID>/management_template_version/<MTVersionID>` request contains only parameters that are relevant to the CI with ID `<CIID>`; parameters that are contained in the management template with version ID `<MTVersionID>` that do not apply to this CI are omitted.
- When creating an assignment, parameter specifications require the parameter's ID and its context. HPE recommends copying the context from the response to `GET assignment_list/<assignmentID>` or `GET assignment_list/draft/...` request, rather than trying to construct the context programmatically.
- The context of a parameter in a management template specifies the configuration object the parameter is defined in (the context of parameters in aspects and policy templates has no special meaning). Typically the context consists of the topology path, which describes how the aspects contained in a management template link sub-CIs to CIs, and the version ID of the aspect containing the parameter, as defined in the management template being assigned.
- In verbose mode, the context provided in a draft response includes the label and description of the aspect containing the parameter to assist users in resolving the parameter reference.
- If you omit a parameter from an assignment creation request, its default value is used.
- HPE recommends setting as few parameters as possible when creating assignments. The number of parameters to be set during assignment can be minimized by creating good default values for the parameters. For more information about management template, aspect, or policy template parametrization, see the *OMi Administration Guide*.
- When creating an assignment using the web service, the CI to which the assignment is to be made must be known. Therefore, any conditions affecting parameter values are resolved and no longer conditional in the responses provided by the web service. As an example, consider a parameter value with a conditional value of `10` for Windows and `20` for Unix. When retrieving assignment information from the web service, the value is resolved and specified as `10` in a response for a CI on Windows, and as `20` for a CI on Linux.
- In contrast to parameter values changed by using the tuning functionality in the MA user interface,

parameter values set using the web service can be overwritten by assignments created from automatic assignment rules. For this reason, HPE recommends not using the assignment web service in conjunction with active auto-assignment rules.

- When upgrading an assignment to a different management template, aspect, or policy template version, all parameter values set by the web service assignments are overwritten by the values as specified in the new management template, aspect, or policy template version, unless you specify the `useExisting=true` URL parameter when calling one of the draft-assignment generation calls. Specifying the `useExisting=true` URL parameter causes the parameter values from an existing assignment to be reused for a new assignment, when neither the parameter type nor the value range changed between versions.
- In OMi versions 10.11 and higher, the parameter XML is also provided with the `base_parameter_id` XML tag. If the `base_parameter_id` is identical for a parameter in two management template, aspect, or policy template versions, then the existing parameter values are used. Existing parameter values cannot be reused, for example, when the parameter type changed from string to password, or when an enum type parameter with the choices `yes` and `no` is expanded to include the `maybe` option in a new version. Additionally, when a parameter is removed in a new policy version, the existing parameter values cannot be reused, even if a parameter with the same name was added in the new version.
- There are two types of parameters, simple and multi-instance parameters, where the latter have an XML tag `<is_instance_parameter>` set to `true`. Simple parameters have a value, whereas instance parameters have an instance list. Each instance in the list consists of an instance value and a list of dependent parameters.

Example: Consider a multi-instance parameter `fileSystem` with the dependent parameters `usageThreshold` (in %) and `messageSeverity` (a number of severity indicators such as `warning` or `critical`). `fileSystem` is used to monitor two instances: `C:` and `D:`. For `C:` the threshold for creating a message with severity `critical` is 90%, while for `D:` the threshold for severity `warning` is 95%.

Note the following when using multi-instance parameters in web service requests:

- Unless multiple instance definitions are specified, a multi-instance parameter is created for only one instance.
- When specifying multiple instances in the definition of a multi-instance parameter, you can give each instance a sequence number to ensure policy template conditions are processed in the correct order.
- Dependent parameters can be overwritten by another instance associated with the multi-instance parameter they depend on if that instance has a higher priority, as defined by the value of XML tag `<ui_order>`. Typically, the instance with the highest priority has a `ui_order` value of 0.
- A response to a web service request for information related to an existing assignment can contain the following flags:
 - `is_default`: Set to the value `false` if the parameter was overwritten.
 - `is_tuned`: Specified if the parameter value was changed using the Monitoring Automation tuning user interface.

- **mandatory**: Specified if the management template, aspect, or policy template defines the parameter as mandatory.
- **readOnly**: Specified if the management template, aspect, or policy template defines the parameter as read-only.
- **expert**: Specified if the management template, aspect, or policy template defines the parameter as an expert parameter.
- Parameter defaults can be specified as a literal using the attribute `value`, or symbolically using the attribute `model_property` with the value set to the name of the CI attribute to provide the value.
 - When symbolic notation is used the parameter default value is calculated from the CI or sub-CI the assignment refers to.
 - Literals take precedence over calculated values.
 - HPE recommends never changing `model_property`, in order to avoid ambiguities with regard to the property's CI type and resolvability.
- The following parameter attributes can be modified:
 - `value`
 - `model_property`
 - Creation of additional instances for a multi-instance parameter.

All other parameter attributes are for information only.

- The values of mandatory parameters must be provided when requesting assignment creation.
- Input validation is done at assignment creation time. If the validation fails (for example, if a mandatory parameter is not specified) the web service returns a descriptive error.

Security of Passwords Passed as Parameter Values

The following security considerations apply to passwords passed in management template parameter values:

- Passwords are passed to the web service as plain text. For security, it is recommended to use a TLS connection for the communication between web service client and web service server.
- Though passed as plain text, passwords are automatically encrypted when stored in the database.
- When retrieving password parameters by executing a GET request the client does not receive the actual value, but the placeholder string `*****`.

This also applies to default passwords returned in a response representing a draft for a POST request. Password parameters posted with the value `*****`, however, instruct Monitoring Automation to use the value stored in the database, removing the need to replace the placeholders with the actual password before posting the response.

Recommended Procedure for Creating POST Requests

The encoding used for XML requests and responses does not depend on the type of request. This

enables a response to be reused for a POST request creating an assignment.

The XML encoding used for assignment-related HTTP requests and responses uses the same format. This enables a GET response to be reused for a POST request when creating an assignment, as follows:

1. Execute a GET request for the assignment to be created:
 - To create an assignment differing from an existing assignment only in the value of a few parameters, it is useful to start with an `/assignment_list/<assignmentID>` request, resulting in a response containing all parameter values for the assignment with ID `<assignmentID>`.
 - To start an assignment from scratch, use the GET `/assignment_list/draft/ci/<CIID>/management_template_version/<MTVersionID>` request, which results in a response representing a draft of the input needed for the assignment creation request (the same procedure can be applied to aspects and policy templates). The draft contains all parameters contained in management template `<CIID>` to be assigned, initialized to their default values. Alternatively, you can use the URL parameter `useExisting=true` in the draft-call to use parameters from an existing assignment to the same CI and configuration artifact.
2. Modify the response by overwriting the parameter values to be changed in the assignment to be created.
3. Post the modified response as a POST `/assignment_list` request to create the new assignment.

Using this method makes your automation algorithm independent of those XML elements not involved in the modification, which greatly increases the robustness of your code compared to manually creating the input.

To verify that the assignment was created, execute a GET request for the new assignment or the deployment jobs it should have created, or inspect the assignment using the OMi MA user interface.

Schema and Class Information

XML Schema

The following file contains the XML schema used for encoding object information:

```
<OMi_HOME>/opr/api/schema/OprDataModel.xsd
```

JAXB Annotated Classes

The following file contains the JAXB annotated classes:

`<OMi_HOME>/lib/opr-external-api.jar`

If you are programming in Java, you can use these classes directly instead of generating classes from the schema. See the *Javadoc API Documentation* for details about the classes provided.

See also the example source code for a java web service client in the following files:

- `<OMI_Home>/opr/examples/assignment-ws-client`
- `<OMi_Home>/opr/examples/mgmt-templates-ws-client`

Verbose and Minimal Mode

The level of detail in the input and output data for a web service request can be determined by specifying verbose or minimal mode:

- In `minimal` mode, input and output data contain a minimal set of XML tags needed to define an object. The minimal set is a true subset of the set of XML tags used in `verbose` mode.
- Default is `minimal` mode.
- To use `verbose` mode, add the suffix `?view=verbose` to the URL when calling the web service.
- When using `minimal` mode, the XML response contains the XML tag `<link rel="verbose">`, the value of which is a URL that provides the verbose version of the response file when called.

Monitoring Automation Web Service Interface Reference

This section provides reference information for the Monitoring Automation (MA) Web Service Interface.

Request Syntax

The Monitoring Automation Web Service interface is accessed by using a URL with the following syntax:

<code>https://<serviceURL>/<request></code>	
<code><serviceURL></code>	The service document where all Monitoring Automation public web services is located here: <code><server>:<port>/opr-config-server/rest/</code> The individual URL of the Monitoring Automation web service: <code><Server>:<port>/opr-config-server/rest/ws/<version></code> where <code><Server></code> is the name of the OMi server and <code><version></code> is the web service interface version (for example: 9.20 or 10.11).

<request>	The service request. For a complete list and syntax of possible service requests, see "Header Syntax" below and "Request Reference" on the next page .
-----------	--

Note: All requests require authentication, and POST and DELETE requests require a header specifying a secure modify token. For more information, see ["Header Syntax" below](#).

Header Syntax

The Monitoring Automation Web Service interface uses the following headers:

Header Name	Header Value
Authorization	<p>The string <code>Basic</code>, followed by a space and the encoded user name and password to be used for authentication.</p> <p>JAVA code example using the <code>org.apache.commons.codec.binary.Base64</code> codec with a RESTEasy client:</p> <pre>import org.apache.commons.codec.binary.Base64; byte[] encodedUserPassword = Base64.encodeBase64 ("myUserName" + ":" + "myPW").getBytes(); response = client.target(URL). header("Authorization", "Basic " + encodedUserPassword). get();</pre>
X-Secure-Modify-Token	<p>The value of the secure modify token as set in the set-cookie.</p> <p>JAVA code example extracting and passing the token:</p> <pre>Object header = response.getHeaders().get("Set-Cookie"); for(String cookie : (List<String>)header) { cookie = cookie.substring(0, cookie.indexOf(";")); String cookieName = cookie.substring(0, cookie.indexOf("=")); String cookieValue = cookie.substring(cookie.indexOf("=") + 1, cookie.length()); if(cookieName.equalsIgnoreCase("secureModifyToken")) break; } response = client.target(URL).header("Authorization","Basic " +encodedUserPassword).header(X-Secure-Modify- Token",cookieValue).post(xml);</pre>

Request Reference

The Monitoring Automation Web Service interface supports the following requests:

Request	Request Type	MA Web Service Action
/assignment_list [/<assignmentID>]	GET	<p>/<assignmentID> is omitted: List all current assignments in the database.</p> <p>/<assignmentID> is specified: Get the assignment with ID <assignmentID>.</p> <p>Note: The 9.20 web service available at <Server>:<port>/opr-config-server/rest/ws/9.20/assignment_lists only retrieves a list of management template assignments, while the 10.11 web service, available at <Server>:<port>/opr-config-server/rest/ws/10.11/assignment_lists, retrieves a list of all types of assignments.</p>
	POST	<p>Create a management template, aspect, or policy template assignment as specified in the posted XML data.</p> <p>Note: /<assignmentID> must be omitted.</p>
	DELETE	<p>Delete the management template, aspect, or policy template assignment with ID <assignmentID>.</p> <p>Note: /<assignmentID> must be specified.</p>
/assignment_list/ci/<CIID>	GET	<p>List all management templates, aspects, or policy templates assigned to the CI with ID <CIID>.</p> <p>Note: The 9.20 web service available at <Server>:<port>/opr-config-server/rest/ws/9.20/assignment_lists only retrieves a list of management template assignments, while the 10.11 web service, available at <Server>:<port>/opr-config-server/rest/ws/10.11/assignment_lists, retrieves a list of all types of assignments.</p>

Request	Request Type	MA Web Service Action
/assignment_list/draft/ci/<CIID>/management_template/{<MTID> management_template_version/<aspectVersionID>}	GET	Get a draft for creating an assignment: <ul style="list-style-type: none"> • If management_template/<MTID> is used, the latest version of the specified management template is assigned. • If management_template_version/<MTVersionID> is used, the specified management template version is assigned. You can make any required modifications to the response, and then POST it using an /assignment_list request to create the assignment.
/assignment_list/management_template/<MTID>	GET	List all assignments of the management template with ID <MTVersionID>.
/assignment_list/management_template_version/<MTVersionID>	GET	List all assignments of the management template with version ID <MTVersionID>.
/assignment_list/draft/ci/<CIID>/aspect/{<aspectID> aspect_version/<aspectVersionID>}	GET	Get a draft for creating an assignment: <ul style="list-style-type: none"> • If aspect/<aspectID> is used, the latest version of the specified aspect is assigned. • If aspect_version/<aspectVersionID> is used, the specified aspect version is assigned. You can make any required modifications to the response, and then POST it using an /assignment_list request to create the assignment.
/assignment_list/aspect/<aspectID>	GET	List all assignments of the aspect with ID <aspectID>.
/assignment_list/aspect_version/<aspectVersionID>	GET	List all assignments of the aspect with version ID <aspectVersionID>.

Request	Request Type	MA Web Service Action
/assignment_list/draft/ci/<CIID>/template/{<templateID> tempalate_version/<templateversionID>	GET	Get a draft for creating an assignment: <ul style="list-style-type: none"> • If template/<templateID> is used, the latest version of the specified template is assigned. • If template_version/<templateVersionID> is used, the specified template version is assigned. You can make any required modifications to the response, and then POST it using an /assignment_list request to create the assignment.
/assignment_list/template/<templateID>	GET	List all assignments of the template with ID <templateID>.
/assignment_list/template_version/<templateVersionID>	GET	List all assignments of the template with version ID <templateVersionID>.
/deployment_job_list [/{<DeploymentJobID> assignment /<AssignmentID>}]	GET	<p>Neither <DeploymentJobID> nor assignment/<AssignmentID> is specified: List all deployment jobs in the database.</p> <p><DeploymentJobID> is specified: Retrieve the deployment job with ID <DeploymentJobID>.</p> <p><assignment/AssignmentID> is specified: List all deployment jobs created as a result of the assignment with ID <AssignmentID>.</p>
/management_template_list[/{<MTID> /ci_type/<CIType>}]	GET	<p><MTID> is omitted: List all management templates in the database.</p> <p><MTID> is specified: List all versions of the management template with ID <MTID>.</p> <p>/ci_type/<CIType> is specified: List all management templates assignable to CIs which have the CI type <CIType>.</p>
/management_template_list/ci/<CIID>]	GET	List all management templates assignable to the CI with ID <CIID>.
/management_template_version_list[/{<MTVersionID>]	GET	<p><MTVersionID> is omitted: List all versions of all management templates in the database.</p> <p><MTVersionID> is specified: Retrieve the management template with the version ID <MTVersionID></p>

Request	Request Type	MA Web Service Action
<code>/aspect_list[/ {<aspectID> /ci_ type/<CIType>}]</code>	GET	<p><aspectID> is omitted: List all aspects in the database.</p> <p><aspectID> is specified: List all versions of the aspect with ID <i><aspectID></i>.</p> <p>/ci_type/<CIType> is specified: List all aspects assignable to CIs of the CI type <i><CIType></i>.</p>
<code>/aspect_list/ci/<CIID></code>	GET	List all aspects assignable to the CI with ID <i><CIID></i> .
<code>/aspect_version_list [/<aspectVersionID>]</code>	GET	<p><aspectVersionID> is omitted: List all versions of all aspects in the database.</p> <p><MTVersionID> is specified: Retrieve the aspect with the version ID <i><aspectVersionID></i></p>
<code>/template_list[/ {<templateID>}]</code>	GET	<p><templateID> is omitted: List all policy templates in the database.</p> <p><templateID> is specified: List all versions of the policy template with ID <i><templateID></i>.</p>
<code>/template_version_list [/ <templateVersionID>]</code>	GET	<p><templateVersionID> is omitted: List all versions of all policy templates in the database.</p> <p><templateVersionID> is specified: Retrieve the policy template with the version ID <i><templateVersionID></i>.</p>

Examples

This section describes a number of use cases for the Monitoring Automation Web Service Interface.

Each use case describes the workflow to be followed. Many of these use cases are covered by the code samples located in the following directories:

```
<OMi_HOME>/opr/examples/assignment-ws-client
<OMi_HOME>/opr/examples/mgmt-template-ws-client
```

Assumptions

The examples are based on the following assumptions:

- The CI is part of the topology monitored by OMi and can be reached over the network.
- Authentication and the secure modify token are properly specified if needed. For details, see ["Reference Information For All Web Services" on page 248](#).
- The OMi account used for authentication has sufficient permissions to carry out the requested operations.

Scenario 1: Monitor a New CI

Recommended workflow (same can be applied for aspects and policy templates):

1. Issue the following web service request to list all management templates assignable to CIs of the CI type *myType*:

```
GET /management_template_list/ci_type/myType
```

Alternatively, list the management templates assignable to the CI:

```
GET /management_template_list/ci/<CIID>
```

2. The response contains a list of management template versions. To get the latest management template version, denoted as *latestMTV*:

- a. Issue the following web service request to simulate the assignment of the management template version:

```
GET /assignment_list/draft/ci/myCI/management_template_version/latestMTV
```

- b. In the response, update any parameter values to be changed.
- c. Post the response using the following request to create the assignment:

```
POST assignment_list
```

Scenario 2: Delete All Assignments to a CI

Recommended workflow:

1. Issue the following web service request to list all assignments to the CI with the ID *myCI*:

```
GET /assignment_list/ci/myCI
```

2. The response contains a list of assignments. For each returned assignment, denoted as *myAssg*, issue the following request to delete it:

```
DELETE /assignment_list/myAssg
```

Note: When a CI is removed, Monitoring Automation automatically deletes any assignments to the CI, and these steps do not need to be performed.

Scenario 3: Temporarily Disable Monitoring of a CI

Recommended workflow:

1. Issue the following web service request to list all assignments to the CI with the ID *myCI*:

```
GET /assignment_list/ci/myCI
```

2. The response contains a list of assignments. Modify the response by setting the *is_enabled* flag to *false* for each returned assignment.
3. Post the response using the following request to update the assignments:

```
POST assignment_list
```

To enable the assignments, repeat the workflow but set the *is_enabled* flag to *true*.

Scenario 4: Change a Parameter Value in an Existing Assignment

Recommended workflow:

1. Issue the following web service request to retrieve the assignment with the ID `myAssg`:
`GET /assignment_list/myAssg`
2. The response contains a parameter value block. Search for the desired parameter, denoted as `myParm`, in the parameter value block by searching for the node `parameter_value > parameter > display_label` with value `myParm`. This node defines the parameter called `myParm` in the user interface.
3. Modify the response by modifying the content of node `value` as desired.
4. Post the response using the following request to update the assignment:
`POST assignment_list`

Scenario 5: Update Assignments to a New Management Template Version

The following scenario is divided into two options, one reusing existing parameter values, and the other without reusing existing parameter values.

Scenario 5a: Update Assignments to a New Management Template Version without reusing existing parameter values

Recommended workflow:

1. Issue the following web service request to list all assignments of the management template version with the ID `myMTV`:
`GET /assignment_list/management_template_version/myMTV`
2. The response contains a list of assignments. For each returned assignment, denoted as `myAssg`:
 - a. Based on the response, determine the ID of the CI to which the management template is assigned.
 - b. Determine the value of each parameter contained in the assignment. If the new management template version uses the same default values as `myMTV`, it is sufficient to determine the values of only those parameters that were changed, as indicated by their flag `is_default` being set to `false`.
 - c. Issue the following web service request to assign the new management template version to the CI (the version ID of the new management template is `newMTV`):
`GET /assignment_list/draft/ci/myCI/management_template_version/newMTV`
 - d. Plug the parameter values retrieved from the assignment of `myMTV` into the draft response for `newMTV`.
 - e. Post the modified draft response using the following request to create the assignment:
`POST assignment_list`

Note: MA automatically ensures that only one version of a management template is assigned to a particular CI at any time by deleting any existing assignments of older versions of a management template before assigning it.

Scenario 5b: Update assignments to a aspect version, reusing existing parameter values

Recommended workflow (the same procedure also applies to management templates and policy templates):

1. Issue the following web service request to list all assignments of the aspect version with the ID `myAspectV`:
`GET /assignment_list/aspect_version/myAspectV`
2. The response contains a list of assignments. For each returned assignment, denoted as `myAssg`:
 - a. Based on the response, determine the ID of the CI to which the aspect is assigned.
 - b. Issue the following web service request to assign the new aspect version to the CI. Specify the `useExisting` flag to take over parameter values:
`GET /assignment_list/draft/ci/<CIID>/aspect_version/newAspectV?useExisting=true`
 - c. Look over the draft response. You can now fill out parameters for which no value was set, or change the values that were set by default, if desired.
 - d. Post the modified draft response by using the following request to create the assignment:
`POST assignment_list`

Scenario 6: Determine Which Assignment Parameters Are Mandatory

Recommended workflow (the same procedure can be used for aspect or policy template assignments):

1. Issue the following web service request to simulate the assignment of the management template version:
`GET /assignment_list/draft/ci/myCI/management_template_version/latestMTV`
2. The response contains a parameter value block. For each parameter, determine the value of the flag required.
3. If the `required` flag is specified and has the value `true`, the parameter is mandatory and must be specified when creating an assignment based on the draft.

Scenario 7: Verify Whether Deployment Was Successful

Recommended workflow:

1. Issue the following request to check for any deployment jobs started as a result of creating the assignment with the ID `myAssg`:
`/deployment_job_list/assignment/myAssg`
2. Examine the response:
 - If the response does not contain any deployment jobs, the assignment was successfully deployed.
 - If the response contains deployment jobs having the state `failed`, deployment failed. Correct any problems indicated in the job description, and create the assignment again.

- If the response contains deployment jobs having the state `running`, deployment is not finished. Wait for a reasonable period to allow the deployment to finish, and repeat the workflow until all jobs having the state `running` have completed.

Tool Execution Web Service Interface

The REST-based Tool Execution web service enables integrators to access the following functionality from an external application:

- Request a list of preconfigured tools that can be applied to a specific CI or in the context of an event.
- Execute a tool on a set of nodes.
- Cancel a tool execution.
- Retrieve the results of a tool execution.

For more information on tools, see the *OMi Administration Guide*. For information on the `opr-tool` command-line interface, see the *OMi Administration Guide*.

How to Access the Tool Execution Web Service

Your entry point to the web service interface is the service document, using the following base URL and syntax:

<code>https://<serviceURL>/<request></code>	
<code><serviceURL></code>	The URL of the Tool Execution web service: <code><server>:<port>/opr-web/rest/<version></code> where <code><server></code> is the name of the OMi server and <code><version></code> is the web service interface version (for example: <code>10.11</code>).
<code><request></code>	The service request. For a complete list and syntax of possible service requests, see "Tool Execution Web Service Request Reference" on the next page .

List of Services in the Service Document

OPR Tool Web Services	URL
Tool Web Service preview POST	<code>https://<server>:<port>/opr-web/rest/10.11/tool_execution/execution_preview/{contextId}</code>
Tool Web Service POST GET DELETE	<code>https://<server>:<port>/opr-web/rest/10.11/tool_execution/execution/{contextId}</code>
Tool Web Service query tools POST	<code>https://<server>:<port>/opr-web/rest/10.11/tool_execution</code>
Tool Web Service preparation POST	<code>https://<server>:<port>/opr-web/rest/10.11/tool_execution/preparation</code>

OPR Tool Web Services	URL
Tool Web Service details GET DELETE	https://<server>:<port>/opr-web/rest/10.11/tool_execution/execution/{contextId}/{resultId}

Security and Authorization

To be able to access the Tool Execution web service, a user needs to be assigned a role with the correct level of **Tools (Execution)** permission. This permission can be accessed in the **Operations Console** category of the permission section of a role. For more information on permissions, see ["Users, Groups, and Roles" on page 1](#).

Error Handling

Error handling for the Tool Execution web service is as follows:

HTTP error codes:

- HTTP error code 403: authorization problems
- HTTP error code 409: requested hosts are not in the RTSM or no CIs match the requirements
- HTTP error code 4xx (other than 403, 409): any form of communication problems with the OMi backend
- HTTP code 200: at least one requested host is in the RTSM and meets the requirements

Individual return codes per requested host (HTTP code always 200):

- Return code -404: requested host is not in the RTSM or the CI does not meet the requirements
- Return code -500: communication problem with the requested host
- Return code 0: execution succeeded
- Other return codes (for example, 1): error code from the command to be executed

Tool Execution Web Service Request Reference

Request Reference

The Tool Execution web service interface supports the following requests:

Request	Request Type	MA Web Service Action
/tool_execution	POST	Retrieve the list of tools applicable for the specified CI(s) or event(s).
/tool_execution/preparation	POST	For a specified tool and CI(s) or event(s), retrieve the parameters to be filled out by the user, and create an execution context in the backend (valid for 10 minutes).

Request	Request Type	MA Web Service Action
/tool_execution/execution_preview/<contextId>	POST	Preview what would be executed, based on the parameters that the user has filled in.
/tool_execution/execution/<contextId>	POST	Execute the tool.
	DELETE	Cancel the tool execution for all affected nodes.
	GET	Retrieve the tool execution status for all affected CIs.
/tool_execution/execution/<contextId>/<resultID>	DELETE	Cancel the tool execution for a single CI.
	GET	Retrieve the tool execution status of a single CI.

XML Structures

For more information on XML structures, see also the OMi public web service schema definitions in <OMi_HOME>/opr/api/schema/OprDataModel.xsd.

The following XML structure examples include comments to help you to tailor them to your specific uses.

Get a contextID:

POST - https://<server>:<port>/opr-web/rest/10.11/tool_execution/preparation

Input:

```
<tool_preparation xmlns="http://www.hp.com/2009/software/opr/data_model"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <tool_id>
  <ci_ids>
    <ci_id>
      <!-- mandatory; multiple ci_id-tags can be used.-->
    <ci_origin>
```

Output:

```
<tool_execution_context xmlns="http://www.hp.com/2009/software/opr/data_model"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <id>
  <!-- the newly created context ID -->
  <tool_ref>
    <id>
    <target_id>
    <label>
    <description>
    <tool_type>
```

```
<has_user_parameter>  
<requires_run_as_user>  
<name>  
<ci_type>  
<tool_execution_result_ref>  
<id>  
<!-- the newly created reference ID for a single target CI -->  
<execution_host>  
<tool_command>  
<tool_execution_state>
```

Execute a tool:

POST - `https://<server>:<port>/opr-web/rest/10.11/tool_execution/execution/<contextId>`

Input:

```
<tool_execution xmlns="http://www.hp.com/2009/software/opr/data_model"  
xmlns:xs="http://www.w3.org/2001/XMLSchema">  
</tool_execution>
```

Output:

```
<tool_execution_context xmlns="http://www.hp.com/2009/software/opr/data_model"  
xmlns:xs="http://www.w3.org/2001/XMLSchema">  
<id>  
<tool_ref>  
<id>  
<target_id>  
<label>  
<description>  
<tool_type>  
<has_user_parameter>  
<requires_run_as_user>  
<name>  
<ci_type>  
</tool_ref>  
<user_parameters/>  
<run_as_username>  
<tool_execution_result_ref>  
<id>  
<execution_host>  
<tool_command>  
<tool_execution_state>
```

Retrieve execution status for all concerned CIs:

GET - `https://<server>:<port>/opr-web/rest/10.11/tool_execution/execution/<contextId>`

```
<tool_execution_context xmlns="http://www.hp.com/2009/software/opr/data_model"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <id>
  <tool_ref>
    <id>
    <target_id>
    <label>
    <description>
    <tool_type>
    <has_user_parameter>
    <requires_run_as_user>
    <name>
    <ci_type>
  <run_as_username>
  <tool_execution_result_ref>
    <id>
    <execution_host>
    <tool_command>
    <tool_execution_state>
    <started>
    <started_text>
    <finished>
    <finished_text>
```

Retrieve execution status for a single CI:

```
GET - https://<server>:<port>/opr-web/rest/10.11/tool_
execution/execution/<contextId>/<resultID>

<tool_execution_result xmlns="http://www.hp.com/2009/software/opr/data_model"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <id>
  <tool_ref>
    <id>
    <target_id>
    <label>
    <description>
    <tool_type>
    <has_user_parameter>
    <requires_run_as_user>
    <name>
    <ci_type>
  <execution_host>
  <tool_command>
  <user_parameters/>
  <run_as_username>
  <tool_execution_state>
  <started>
  <started_text>
```

```
<finished>
<finished_text>
<executed_in_context_of>
  <entry>
    <key>
    <value>
<output>
<result_code>
```

For a given tool and CI(s) or event(s), retrieve the parameters to be filled out by the user, and create an execution context in the backend (valid for 10 minutes):

GET - `https://<server>:<port>/opr-web/rest/10.11/tool_execution/preparation`

Input (when specifying CI IDs):

```
<tool_preparation xmlns="http://www.hp.com/2009/software/opr/data_model"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <tool_id>
  <ci_ids>
    <ci_id>
    <!-- mandatory; multiple ci_id-tags can be used.-->
  <ci_origin>
```

Input (when specifying event IDs):

```
<tool_preparation xmlns="http://www.hp.com/2009/software/opr/data_model"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <tool_id>
  <event_ids>
    <event_id>
    <!-- mandatory; multiple event_id-tags can be used.-->
  <ci_origin>
  <!-- mandatory; has one of the following values: event_related_ci, event_node,
or event_source_ci -->
```

Output:

```
<tool_execution_context xmlns="http://www.hp.com/2009/software/opr/data_model"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <id>
  <tool_ref>
    <id>
    <target_id>
    <label>
    <description>
    <tool_type>
    <has_user_parameter>
    <requires_run_as_user>
    <name>
    <ci_type>
```

```
<user_parameters>
  <entry>
    <key>
    <value>
  <entry>
    <key>
    <value>
  <entry>
    <key>
    <value>
```

Retrieve a list of tools applicable for a given CI:

POST - https://<server>:<port>/opr-web/rest/10.11/tool_execution

Input:

```
<tool_query xmlns="http://www.hp.com/2009/software/opr/data_model"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <ci_ids>
    <ci_id>
    <!-- mandatory; multiple ci_id-tags can be used.-->
```

Output:

```
<tool_list xmlns="http://www.hp.com/2009/software/opr/data_model"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <id>
  <tools_for_ci>
    <tool_ref>
      <id>
      <target_id>
      <label>
      <description>
      <tool_type>
      <has_user_parameter>
      <requires_run_as_user>
      <name>
      <ci_type>
    <tool_ref>
      <id>
      <target_id>
      <label>
      <description>
      <tool_type>
      <has_user_parameter>
      <requires_run_as_user>
      <name>
      <ci_type>
    <tool_ref>
```

```
<id>  
<target_id>  
<label>  
<description>  
<tool_type>  
<has_user_parameter>  
<requires_run_as_user>  
<name>  
<ci_type>
```

Retrieve a list of tools applicable for a given event (specified by ID):

POST - `https://<server>:<port>/opr-web/rest/10.11/tool_execution`

Input:

```
<tool_query xmlns="http://www.hp.com/2009/software/opr/data_model"  
xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <event_ids>  
    <event_id>  
    <!-- mandatory; multiple ci_id-tags can be used.-->
```

Output:

```
<tool_list xmlns="http://www.hp.com/2009/software/opr/data_model"  
xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <id>  
  <tools_for_event_related_ci>  
    <!--Only appears if there is at least one tool associated with related CI  
of the specified event -->  
    <tool_ref>  
      <!-- Information for one or multiple tools - like above -->  
    <tools_for_event_node>  
      <!--Only appears if there is at least one tool associated with the node  
on which the related CI of the event is hosted-->  
      <tool_ref>  
        <!-- Information for one or multiple tools - like above -->  
    <tools_for_event_source_ci>  
      <!--Only appears if there is at least one tool associated with the source  
CI of the event -->  
      <tool_ref>  
        <!-- Information for one or multiple tools - like above -->
```

Preview what would be executed based on the parameters the user has filled in:

POST - `https://<server>:<port>/opr-web/rest/10.11/tool_execution/execution_preview/<contextID>`

```
<tool_execution xmlns="http://www.hp.com/2009/software/opr/data_model"  
xmlns:xs="http://www.w3.org/2001/XMLSchema">  
<user_parameters>
```

```
<entry>  
<key>  
<value>
```

Output:

```
<tool_execution_context xmlns="http://www.hp.com/2009/software/opr/data_model"  
xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <id>  
  <tool_ref>  
    <id>  
    <target_id>  
    <label>  
    <description>  
    <tool_type>  
    <has_user_parameter>  
    <requires_run_as_user>  
    <name>  
    <ci_type>  
  <user_parameters>  
    <entry>  
      <key>  
      <value>  
  <run_as_username>  
  <tool_execution_result_ref>  
    <id>  
    <execution_host>  
    <tool_command>  
    <tool_execution_state>
```

Cancel tool execution for all affected nodes:

```
POST - https://<server>:<port>/opr-web/rest/10.11/tool_  
execution/execution/<contextID>
```

Output:

```
<tool_execution_context xmlns="http://www.hp.com/2009/software/opr/data_model"  
xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <id>  
  <tool_ref>  
    <id>  
    <target_id>  
    <label>  
    <description>  
    <tool_type>  
    <has_user_parameter>  
    <requires_run_as_user>  
    <name>  
    <ci_type>
```

```
<user_parameters>
  <entry>
    <key>
    <value>
  <entry>
    <key>
    <value>
<tool_execution_result_ref>
  <id>
  <execution_host>
  <tool_command>
  <tool_execution_state>
  <started>
  <started_text>
```

Cancel execution for a single CI:

POST - https://<server>:<port>/opr-web/rest/10.11/tool_execution/execution/<contextID>/<resultID>

Output:

```
<tool_execution_context xmlns="http://www.hp.com/2009/software/opr/data_model"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <id>
  <tool_ref>
    <id>
    <target_id>
    <label>
    <description>
    <tool_type>
    <has_user_parameter>
    <requires_run_as_user>
    <name>
    <ci_type>
  <user_parameters>
    <entry>
      <key>
      <value>
    <entry>
      <key>
      <value>
  <tool_execution_result_ref>
    <id>
    <execution_host>
    <tool_command>
    <tool_execution_state>
    <started>
    <started_text>
```


Chapter 36: User Management Web Services Interface

The User Management Web Services enable integrators to access the following user management functionality from an external application:

- Create, update, or delete a user.
- List all current users in the database, allowing filtering by user ID.
- List all time zones that can be used when setting up new users.
- Create, update, or delete a user group.
- List all available user groups in the database.
- List all groups available for LDAP mapping, as specified during LDAP setup.
- Create, update, or delete a role in the database.
- List all current roles in the database.
- List all roles which are assigned to the user with the specified ID.
- List all roles which are assigned to the user group with the specified ID.
- List all available authorizable resources that can be set in a role.
- Create or delete an event category.
- List available event categories, allowing filtering by category ID.

The easiest way to work with services that allow addition or modification of objects is to retrieve an XML example with the **GET** request, then modify it to your needs and **POST** to create or **PUT** to update the object. When using a JAVA client, the classes provided in the `opr-external-api.jar` can be used to marshal or unmarshal the XML content directly.

For general information about using the User Management web service interface, see ["Reference Information For All Web Services" on page 248](#).

For more information on users, groups, and roles, see the *OMi Administration Guide*.

How to Access the User Management Web Services

Your entry point to the web service interface is the Service Document, using the following base URL and syntax:

<code>https://<serviceURL>/<request></code>	
<code><serviceURL></code>	The URL of the User Management web services: <code><server>:<port>/opr-web/admin/rest/<version></code> where <code><server></code> is the name of the OMi server and <code><version></code> is the web service interface version (for example: 10.01).

<request>	The service request. For a complete list and syntax of possible service requests, see "User Management Web Services Request Reference" on the next page.
-----------	--

List of OPR Administration Services in the Service Document

OPR Administration Service	URL
User Service:	https://<server>:<port>/opr-web/admin/rest/10.01/user_object_list
User Service:	https://<server>:<port>/opr-web/admin/rest/10.01/user_object_list/{id}
Time Zone Service:	https://<server>:<port>/opr-web/admin/rest/10.01/time_zone_list
User Group Service:	https://<server>:<port>/opr-web/admin/rest/10.01/user_group_list/{id}
User Group Service:	https://<server>:<port>/opr-web/admin/rest/10.01/user_group_list
LDAP User Group Service:	https://<server>:<port>/opr-web/admin/rest/10.01/ldap_group_list
User Role Service:	https://<server>:<port>/opr-web/admin/rest/10.01/role_list
User Role Service:	https://<server>:<port>/opr-web/admin/rest/10.01/role_list/{id}
User Role Service:	https://<server>:<port>/opr-web/admin/rest/10.01/role_list/user/{id}
User Role Service:	https://<server>:<port>/opr-web/admin/rest/10.01/role_list/user_group/{id}
Authorizable Resource Service:	https://<server>:<port>/opr-web/admin/rest/10.01/auth_resource_list
Event Category Service:	https://<server>:<port>/opr-web/admin/rest/10.01/event_category_list
Event Category Service:	https://<server>:<port>/opr-web/admin/rest/10.01/event_category_list/{id}

User Management Web Services Request Reference

The User Management Web Services interface supports the following requests:

User Service

Request	Request Type	Web Service Action
/user_object_list[/ {userID}]	GET	<p>/<userID> is omitted: List all current users in the database.</p> <p>/<userID> is specified: Get the user with ID<userID>.</p>
	POST	<p>Create a user as specified in the posted XML data.</p> <p>Note:</p> <ul style="list-style-type: none"> • /<userID> must be omitted.
	DELETE	<p>Delete the user with ID <userID>.</p> <p>Note:</p> <ul style="list-style-type: none"> • /<userID> must be specified.
	PUT	<p>Edit the user with ID <userID>.</p> <p>Note:</p> <ul style="list-style-type: none"> • /<userID> must be specified.

Time Zone Service

Request	Request Type	Web Service Action
/time_zone_list	GET	Lists all time zones that can be used when setting up new users.

User Group Service

Request	Request Type	Web Service Action
/user_group_list[/ {userGroupID}]	GET	<p>/<userGroupID> is omitted: List all current user groups in the database.</p> <p>/<userGroupID> is specified: Get the user group with ID<userGroupID>.</p>
	POST	<p>Create a user group as specified in the posted XML data.</p> <p>Note:</p> <ul style="list-style-type: none"> • /<userGroupID> must be omitted.
	DELETE	<p>Delete the user group with ID <userGroupID>.</p> <p>Note:</p> <ul style="list-style-type: none"> • /<userGroupID> must be specified.
	PUT	<p>Edit the user group with ID <userGroupID>.</p> <p>Note:</p> <ul style="list-style-type: none"> • /<userGroupID> must be specified.

LDAP User Group Service

Request	Request Type	Web Service Action
/ldap_group_list	GET	Lists all groups available for the LDAP mapping as specified during the LDAP setup.

User Role Service

Request	Request Type	Web Service Action
/role_list [/<roleID>]	GET	<p>/<roleID> is omitted: List all current roles in the database.</p> <p>/<roleID> is specified: Get the role with ID <roleID>.</p>
	POST	<p>Create a role as specified in the posted XML data.</p> <p>Note:</p> <ul style="list-style-type: none"> /<roleID> must be omitted. <div style="border: 1px solid gray; padding: 5px; margin: 10px 0;"> <p>Note: When adding or modifying a role via the Web Service, permissions can be set on a view by adding/removing the following permission tags to the role:</p> <pre><permission> <resource_key>rtsm-view.{VIEW_NAME}</resource_key> <operation_key>{KEY}</operation_key> </permission></pre> <p>In the resource key, the VIEW_NAME must be specified to select which view receives the permission. The operation_key specifies the permitted operation. Possible values are <i>change</i>, <i>view</i> or <i>delete</i>.</p> </div>
	DELETE	<p>Delete the role with ID <roleID>.</p> <p>Note:</p> <ul style="list-style-type: none"> /<roleID> must be specified.
	PUT	<p>Edit the role with ID <roleID>.</p> <p>Note:</p> <ul style="list-style-type: none"> /<roleID> must be specified.
/role_list/user_object/<userID>	GET	List all roles which are assigned to the user with the specified ID.
/role_list/user_group/{userGroupID}	GET	List all roles which are assigned to the user group with the specified ID.

Authorizable Resource Service

Request	Request Type	Web Service Action
/auth_resource_list	GET	Lists all available authorizable resources that can be set in a role.

Event Category Service

Request	Request Type	Web Service Action
/event_category_list[/categoryID]	GET	<p>/categoryID is omitted: List all current event categories in the database.</p> <p>/categoryID is specified: Get the event category with ID <categoryID>.</p>
	POST	<p>Create a event category as specified in the posted XML data.</p> <p>Note:</p> <ul style="list-style-type: none"> • /categoryID must be omitted.
	DELETE	<p>Delete the event category with ID <categoryID>.</p> <p>Note:</p> <ul style="list-style-type: none"> • /categoryID must be specified.
	PUT	<p>Edit the event category with ID <categoryID>.</p> <p>Note:</p> <ul style="list-style-type: none"> • /categoryID must be specified.

Examples

This section describes the recommended workflow for creating a role and assigning permissions to the role. In addition, the example creates user groups based on LDAP groups:

1. *Optional.* Set up event categories to fine tune permissions on events not assigned to users:

Example:

```
POST - https://<server>:<port>/opr-web/admin/rest/10.01/event_category_list
<event_category xmlns="http://www.hp.com/2009/software/opr/data_model">
  <category>MyEvtentCategory</category>
</event_category>
```

2. Get all available resources and operations:

Example:

```
GET - https://<server>:<port>/opr-web/admin/rest/10.01/auth_resource_list
```

3. For each permission you want to assign to a role, select the resource key and the operation key from the output. For example, to assign permissions to the User Group Assignments manager, the Content Pack manager, and the RTSM view All My Windows Servers, identify the following keys:

Example:

```
<auth_resource_list ...>
<auth_resource>
  <key>omi-event-to-group</key>
  ...
  <auth_operation_list>
    <auth_operation>
      <key>full-control</key>
</auth_resource>
<auth_resource>
  <key>omi-content-mgr</key>
  ...
  <auth_operation_list>
    <auth_operation>
      <key>full-control</key>
</auth_resource>
<auth_resource>
  <key>rtsm-view</key>
  ...
  <key>view</key>
  ...
<auth_resource>
  <key>rtsm-view.All My Windows Servers</key>
  ...
</auth_resource_list>
```

Note: If the <auth_resource> is part of <child_auth_resource_list>, it might not directly

specify all operations. Instead, search at the parent <auth_resource> item.

4. Create a role and configure the permissions identified in the previous step:

Example:

```
POST - https://<server>:<port>/opr-web/admin/rest/10.01/role_list

<role xmlns="http://www.hp.com/2009/software/opr/data_model">
  <name>Role with RTSM views REST API</name>
  <permission>
    <resource_key>rtsm-view.All My Windows Servers</resource_key>
    <operation_key>view</operation_key>
  </permission>
  <permission>
    <resource_key>omi-event-to-group</resource_key>
    <operation_key>full-control</operation_key>
  </permission>
  <permission>
    <resource_key>omi-content-mgr</resource_key>
    <operation_key>full-control</operation_key>
  </permission>
  <permission>
    <resource_key>omi-event.unassigned-event.MyEventCategory</resource_
key>
    <operation_key>assign-to</operation_key>
  </permission>
  <permission>
    <resource_key>omi-event.assigned-event</resource_key>
    <operation_key>launch-automatic-action</operation_key>
  </permission>
  <permission>
    <resource_key>omi-tool.Database_Operational_Tools</resource_key>
    <operation_key>execute</operation_key>
  </permission>
  <permission>
    <resource_key>omi-tool.MyToolCategory</resource_key>
    <operation_key>execute</operation_key>
  </permission>
</role>
```

Tip: Get the role ID from the output for later use, for example:

```
<id>61a2d926-b9ab-42dd-99da-8ebe2a09c82c</id>
```

5. Create a user group and assign the previously created role to the group:

Example: Grant permissions to all LDAP users

- a. *Prerequisite.* LDAP mapping settings must be configured with “Automatically create LDAP users” and “Add new users to groups”.
- b. Submit the following POST request.

Example:

```
POST - https://<server>:<port>/opr-web/admin/rest/10.01/user_group_list

<user_group xmlns="http://www.hp.com/2009/software/opr/data_model">
  <name>Group Name</name>
  <description>Group with LDAP mapping and Role for ALL LDAP
users</description>
  <event_assignment>true</event_assignment>
  <ldap_auto_assignment>true</ldap_auto_assignment>
  <user_group_to_role_list>
    <user_group_to_role>
      <role>
        <id>61a2d926-b9ab-42dd-99da-8ebe2a09c82c</id>
      </role>
    </user_group_to_role>
  </user_group_to_role_list>
</user_group>
```

Example: Grant permissions to users that are members of specific LDAP groups

- a. *Prerequisite.* LDAP mapping settings must be configured with “Synchronize LDAP groups with OMi groups”.
- b. List all available LDAP groups.

Example:

```
GET - https://<server>:<port>/opr-web/admin/rest/10.01/ldap_group_list
```

Identify the LDAP groups in the output to which you want to assign to the role.

- c. Submit the following POST request:

Example:

```
POST - https://<server>:<port>/opr-web/admin/rest/10.01/user_group_list

<user_group xmlns="http://www.hp.com/2009/software/opr/data_model">
  <name>Group Name</name>
  <description>Group with LDAP mapping and Role for LDAP
users</description>
  <event_assignment>true</event_assignment>
  <ldap_auto_assignment>false</ldap_auto_assignment>
  <user_group_to_role_list>
    <user_group_to_role>
      <role>
        <id>61a2d926-b9ab-42dd-99da-8ebe2a09c82c</id>
      </role>
    </user_group_to_role>
  </user_group_to_role_list>
  <ldap_group_list>
    <ldap_group>
      <name>EMEA Operators</name>
      <domain>emea</domain>
      <ldap_server>YourLDAPServer</ldap_server>
    </ldap_group>
    <ldap_group>
      <name>EMEA Administors</name>
      <domain>emea</domain>
      <ldap_server>YourLDAPServer</ldap_server>
    </ldap_group>
  </ldap_group_list>
</user_group>
```

6. Query to which groups a user belongs:

Retrieve the user's information and search for the related groups in the user_group_list:

```
GET- https://<server>:<port>/opr-web/admin/rest/10.01/user_object_list/
{userID}
```

```
<user_object xmlns="http://www.hp.com/2009/software/opr/data_model"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <id>c58e8046-825e-4023-8091-c05d76ef7138</id>
  <version>0</version>
  <login>ingroup</login>
  <name>user in group</name>
  <email_address>example@example.com</email_address>
  <time_zone>America/La_Paz</time_zone>
  <ldap_user>false</ldap_user>
  <super_administrator>false</super_administrator>
  <inactive>false</inactive>
  <user_object_to_role_list>
    <user_object_to_role>
      <role>
        <id>626529a9-6cd2-494d-be64-38833b73ff77</id>
        <version>2</version>
        <name>DB Expert Role</name>
        <description>Expert permission for databases.</description>
      </role>
    </user_object_to_role>
  </user_object_to_role_list>
  <user_group_list>
    <user_group>
      <id>76ac4edb-1d29-4e42-9fee-d14fc732b02f</id>
      <version>1</version>
      <name>MiddleGroup</name>
      <event_assignment>true</event_assignment>
      <ldap_auto_assignment>false</ldap_auto_assignment>
    </user_group>
  </user_group_list>
</user_object>
```

7. Retrieve roles assigned to users or groups:

Retrieve all roles assigned to a specific user

```
GET- https://<server>:<port>/opr-web/admin/rest/10.01/role_list/user_object/
{ID}
```

Retrieve all roles assigned to a specific group

```
GET- https://<server>:<port>/opr-web/admin/rest/10.01/role_list/user_group/
{ID}
```

8. Remove a user from a group:

To remove a user from a group, first get the group, then remove the user from the users list of the output, and then send the request to modify the group without the specified user.

```
1. GET- https://<server>:<port>/opr-web/admin/rest/10.01/user_group_list/
```

```
{groupID}
<user_group>
<id>00000000-0000-0000-8689-17929130e34a</id>
<version>1</version>
<name>Administrators</name>
<description>Omi Administrators</description>
<event_assignment>>false</event_assignment>
<ldap_auto_assignment>>false</ldap_auto_assignment>
<user_object_list>
  <user_object>
    <id>00000000-0000-0000-95ac-ae0c2214712b</id>
  </user_object>
  <user_object>
    <id>00000000-0000-0000-b315-65efb9991798</id>
  </user_object>
  <user_object>
    <id>00000000-0000-0000-8593-821aed2d2d52</id>
  </user_object>
</user_object_list>
<user_group_to_role_list>
  <user_group_to_role>
    <role>
      <id>ca252aa1-11e7-4645-bcee-6757ea6ec2c5</id>
      <version>1</version>
      <name>Administrator Role</name>
      <description>All permissions</description>
    </role>
  </user_group_to_role>
</user_group_to_role_list>
</user_group>
```

2. Remove the user with ID 00000000-0000-0000-95ac-ae0c2214712b

```
<user_object>
  <id>00000000-0000-0000-95ac-ae0c2214712b</id>
</user_object>
```

3. Send the request to modify the group without the specified user(s).

PUT- https://<server>:<port>/opr-web/admin/rest/10.01/user_group_list/{groupID}

```
<user_group>
<id>00000000-0000-0000-8689-17929130e34a</id>
<user_object_list>
  <user_object>
    <id>00000000-0000-0000-b315-65efb9991798</id>
  </user_object>
```

```
<user_object>
  <id>00000000-0000-0000-8593-821aed2d2d52</id>
</user_object>
</user_object_list>
<user_group_to_role_list>
  <user_group_to_role>
    <role>
      <id>ca252aa1-11e7-4645-bcee-6757ea6ec2c5</id>
    </role>
  </user_group_to_role>
</user_group_to_role_list>
</user_group>
```

Note: If the PUT request does not include the full list of roles assigned to the user, then the group loses the role assignment information as well.

9. Delete a group or a set of groups:

Delete a single group

```
DELETE - https://<server>:<port>/opr-web/admin/rest/10.01/user_group_list
{groupID}
```

Delete a set of groups

To delete a set of groups, use the multiMode parameter to specify the group ids in a comma separated list of the id parameter:

```
https://<server>:<port>/opr-web/admin/rest/10.01/user_group_
list/?multiMode=true&id=00000000-0000-0000-83d4-05bbbce8ebf,00000000-0000-
0000-8e69-6e14d4d995a9
```

Part IX: Groovy Scripts

OMi supports script-based customization. Using scripts enables adding functionality to the OMi process flow in situations where the required functionality is not provided in OMi or cannot be added using the standard OMi tools and configuration options.

How this Section is Organized

This section documents where and how Groovy scripts can be added. It includes the following parts:

- There are several different areas where customization is possible. How to develop and deploy scripts for each area is discussed in detail in section "[Development and Deployment of Scripts](#)" on [page 341](#).
- Reference information applicable to any script is provided in section "[Reference Information](#)" on [page 362](#).
- Guidance on best practices when developing scripts is provided in section "[Best Practices](#)" on [page 337](#).

Introduction to Customization Scripts

The following aspects are critical to developing successful customization scripts:

- **Script Execution Trigger**

The trigger causing the script to be executed. For example, scripts can be configured to run in conjunction with the occurrence of an event. Triggers are detailed in "[Development and Deployment of Scripts](#)" on [page 341](#), in the sections relevant to the various types of customization under the *Process Flow* heading.

- **Script Execution Process Location**

The place in the process flow when the script is executed. For example, event-based scripts can be configured to be executed before the event is processed, or after. Available locations are detailed in "[Development and Deployment of Scripts](#)" on [page 341](#), in the sections relevant to the various types of customization under the *Process Flow* heading.

- **Programming Language**

OMi supports Groovy scripts. Groovy is a dynamic object-oriented programming language for the Java platform. When used as a scripting language for the Java Platform, Groovy is dynamically compiled to Java Virtual Machine (JVM) bytecode making it compatible with external Java code and libraries. Groovy is similar to Java, and most Java code is syntactically valid in Groovy. Details about the Groovy syntax are outside the scope of this guide, but there are numerous sources on the Internet you can refer to. A good start is to search for Groovy (programming language) on *Wikipedia*, and use the list of references mentioned in the article. The official web site is at the following URL:

<http://groovy.codehaus.org/>

For more information about obtaining the Groovy console, as well as OMi-specific information on installation, deployment and debugging using the console, see ["The Groovy Console" on page 362](#).

- **Script Format**

Scripts need to follow a specific format so that the OMi procedure calling the script is able to execute it correctly in the various process locations where scripts can be added. Template scripts and examples are provided with the installation. Details about the format to be used for the various types of customization are detailed in ["Development and Deployment of Scripts" on page 341](#), in the sections relevant to the various types of customization under the *Format* heading.

- **Information Functions**

Scripts are often used to add information about the managed configuration items (CIs) to the standard information set. The information to be added must be obtained from various services and processes. Use the functions from the appropriate Application Programming Interface (API). For example, when enriching event information you can use the functions from the *RTSM Java API* to retrieve information from a CI. The available OMi and external APIs are listed in ["Available APIs" on page 365](#). Examples of functions are detailed in ["Development and Deployment of Scripts" on page 341](#), in the sections relevant to the various types of customization under the *Example* heading.

- **Best Programming Practices**

Best practices and things you should avoid are detailed in ["Best Practices" on page 337](#). Any specific best practices that should be followed are detailed in ["Development and Deployment of Scripts" on page 341](#), in the sections relevant to the various types of customization under the *Best Practices Specific to ...* heading.

- **Script Management**

Scripts are placed on the server running the OMi instance and configured for the appropriate process using the OMi console. How to configure a script depends on where it is used. The steps you need to take to include and activate your scripts are detailed in ["Development and Deployment of Scripts" on page 341](#), in the sections relevant to the various types of customization under the *Script Management* heading.

Chapter 37: Best Practices

The tips in this section will help you to develop high-quality scripts.

Improving Performance

Customization scripts are executed as part of the processing flow, causing customization to slow down process execution. As long as the increase in total processing time is small this is not a problem, but if the total processing time increases to a significant portion of the interval between subsequent process runs, all OMi processes, including event and topology synchronization, could be negatively affected or even disrupted. Therefore, it is important to make the absolute time it takes to execute your custom script as small as possible.

Cache Data Wherever it Improves Performance

Caching data that takes time to retrieve is a useful way to speed up performance. Examples are data from resources external to the synchronization process, such as RTSM data, data from user databases, and data from files.

Caching comprises the following operations:

1. Connecting to the external resource.
2. Loading the data from the resource and storing it in a data structure in memory, which typically has a much shorter access time. The data in memory is called cache data.
3. Using the data in the custom process.
4. Disconnecting from the external resource.
5. Removing the cached data from memory.

Typically, connecting to and loading from the external resource are the most time-consuming steps.

When implementing caching, consider the following aspects:

- Avoiding is better than fixing, so try to minimize dependency on external resources:
 - When contemplating using external data, contemplate whether it is really necessary or not.
 - If data is available or derivable from an internal resource, don't use an external resource.
 - If data is available or derivable from a number of external resources, use the minimum number of resources.
 - If data is available or derivable through a number of different APIs, use the API with the fastest performance.
- Cache only data that is likely to be reused. CI data is likely to be reused, and should be cached in most situations. It is not recommended, however, to always cache all available CI data. If your script only uses events associated with a certain type of CI, for example, you could apply a filter to cache relevant events only. Try to find a balance, however, between caching only data that would be reloaded against increasing code complexity, which could decrease process performance due to the

need to evaluate numerous `if` and `case` conditions.

- Connecting to an external resource needs to be done only once. The best place to connect to resources containing data used throughout the script is in the script class's `init()` method. Disconnect in the `destroy()` method. Code inside the `process()` method can then directly access the resource.
- Loading the data will usually be done while using the code, because different data is needed in different places. However, if you reuse the same data over and over again, you could use the `init()` method to connect, load the data to a local array, and disconnect right away.
- Having a resource open and storing data in memory uses system resources. To optimize resource usage you may want to connect the first time data is accessed and disconnect after loading in the `process()` method; this is just as fast, as long as you connect only once. Keep in mind, however, that it is always better to optimize script performance for absolute time than to minimize system resource usage.
- After finishing debugging, remove any logging code which is not necessary in the production environment from the script to avoid unnecessary overhead.

Avoid Using Slow APIs and Functions

- Do not use APIs known to have slow performance characteristics in performance critical code. Most notably, pay attention to the performance of APIs when inserting custom scripts in the event pipeline.
- Access the RTSM using its JAVA API, which has a better performance than other APIs such as SOAP-based web services APIs.
- It is not permitted to use shell functions such as `System.exit()`. Shell function calls cause the JVM, which executes the Groovy script, to be stopped and restarted.
- Perform time-consuming processes in background threads wherever possible.

Improving Code Quality

The quality of your code has a direct impact on the ease with which the resulting script can be integrated and maintained.

Debug Your Scripts to Minimize Potential Problems

Running problematic scripts can lead to performance issues, even if the code executes. In-depth testing and debugging is necessary to produce robust scripts that are safe to use in your production environment.

- Use the Groovy Console during script validation and testing (see ["The Groovy Console" on page 362](#)).
- When debugging the script, include plenty of logging statements allowing you to track the behavior of the code. HPE recommends encapsulating the logging of debug information in a `debug()` method, making it easy to remove or deactivate logging in the production environment (see ["Improving Performance" on the previous page](#)). Exceptions are calls to the `init()` and `destroy()` methods. HPE recommends logging these using an `info()` method so you can track successful loading and unloading of your script, in both debugging and production environments.

- Use comments and verbose variable names to facilitate future script maintenance.
- If an exception gives rise to a run-time error, the process could be disrupted. Catch exceptions by using try/catch blocks, and provide an elegant exit in the exception handler.
- When connecting in the `init()` method, check the validity of the database handle before accessing the resource. If the resource was disconnected for reasons outside the control of the script, handling this exception properly allows you to recover the connection by reconnecting.
- Use `?` when referencing object properties to avoid a `NullPointerException` when the object evaluates to `null`. When referencing a property using `?` the whole expression containing the reference evaluates to `null`, rather than throwing an exception.

Example: The following code snippet will still work if the `assignedUser` object evaluates to `null`:

```
if (event.assignedUser?.userName) {...}
```

- Avoid using `def` statements wherever possible to enhance the transparency of the script and facilitate future script maintenance. HPE recommends declaring the actual object type instead of using a `def` statement, making it easier to find the properties and methods of the instanced object. Another advantage of a transparent script is that it is easier to debug. Note, however, that there are cases in which using the `def` statement is justified. As an example, consider the backward compatibility of a script containing a function having an argument of a class type that was not available in a previous version. If the argument is declared as `type def` the script will still load in the previous version, even though the class is not defined in that version. In this case, adding a comment specifying the actual object type of the argument in question.
- Use the internal script error feature to preempt process disruptions caused by scripts experiencing run-time problems. If a script class has a public method named `getInternalScriptError()` and the method returns a value of 10 or higher, the script is skipped by the platform service executing the script. To use this feature, insert the following code:

```
int internalScriptError = 0;
def int getInternalScriptError() { return internalScriptError; }
def setInternalScriptError() { internalScriptError++; }
```

This code causes the member variable `internalScriptError`, which holds the script error level, to be initialized to 0, and enables the programmer to increase the error level by calling the `setInternalScriptError()` method whenever an error occurs. If, for example, the `setInternalScriptError()` method is called whenever an attempt to connect to the database fails, the script is skipped after 10 tries, and no longer uses process time.

Facilitating writing code

The following tips help you make writing Groovy scripts for OMi as easy as possible.

Use an External Editing Tool

The OMi administration screens that are used to configure scripts on the platform are useful to manage the script, but are not sufficient to edit, maintain, test and debug scripts. You must rely on an external editing tool to create a more versatile editing environment

- Line numbering and a text search function are useful for matching log file references to code lines. Neither of these is available in the script form included in the OMi Console, so an external editor is

useful during script development.

- For most scripts a simple tool such as Notepad++, vi or GNU emacs is sufficient.
 - Notepad++ is a freeware improvement on Notepad featuring line numbering and Java syntax highlighting, as well as an optional auto indent plug-in. You can get Notepad++ from <http://notepad-plus-plus.org/download/>.
 - vi is a visual text editor developed for UNIX, and is installed on UNIX systems. Several free and open source software implementations of vi exist; for more information, see <http://en.wikipedia.org/wiki/Vi>.
 - GNU Emacs is an extensible, customizable text editor featuring syntax highlighting, full Unicode support, and many optional extensions. You can get GNU emacs from <http://www.gnu.org/software/emacs/>.
- For more involved scripts requiring syntax checking use the Groovy console. For details about installing and using the Groovy console, see "The Groovy Console" on page 362.
- For large event forwarding scripts such as the Service Manager adapter script it could be useful to turn to more comprehensive script editors, for example Eclipse or IntelliJ.

Use Script Templates

Script templates and examples are available in the directories where scripts can reside. Starting a new script from a template rather than from scratch makes it easier to follow the script format rules and make it less you forget something.

Be Careful with Copy-and-Paste

Be careful with copy-and-paste from emails, PDF files or Microsoft Word documents.

- Using copy-and-paste could result in invalid code because of extra line breaks or because of line breaks being treated differently by the Groovy engine. The Groovy engine does not require a semicolon to end a statement, and will sometimes treat a line break as the end of a statement. Unintentional line breaks could therefore lead to unintentional code behavior.
- If you do copy and paste code, inspect it line by line and rejoin statements split across two lines, which the Groovy engine may unintentionally interpret as two separate statements.

Chapter 38: Development and Deployment of Scripts

This section documents triggers, process flow, script management, best practices and examples for the areas where scripts can be used. The following types of scripts are covered:

- [Event Processing Interface Scripts](#) 341
- [Custom Action Scripts](#) 347
- [Certificates Scripts](#) 349
- [Service Health Scripts](#) 351
- [Topology Synchronization Scripts](#) 352
- [Event Forwarding Scripts](#) 354
- [External Instruction Retrieval Scripts](#) 357

Event Processing Interface Scripts

The Event Processing Interface (EPI) enables you to run user-defined Groovy scripts during event processing. Actions include modification and enhancement of events using data external to the EPI process. You could, for example, add data from an external SQL database.

Note: EPI scripts are not used to forward events; for information about scripts to forward events, see ["Event Forwarding Scripts" on page 354](#).

This section includes:

- ["Process Flow" below](#)
- ["Script Management" on the next page](#)
- ["Format" on page 343](#)
- ["Best Practices Specific to EPI Scripts" on page 344](#)
- ["Example" on page 345](#)

Process Flow

- Scripts are executed in the following working directory on the OMi gateway server:

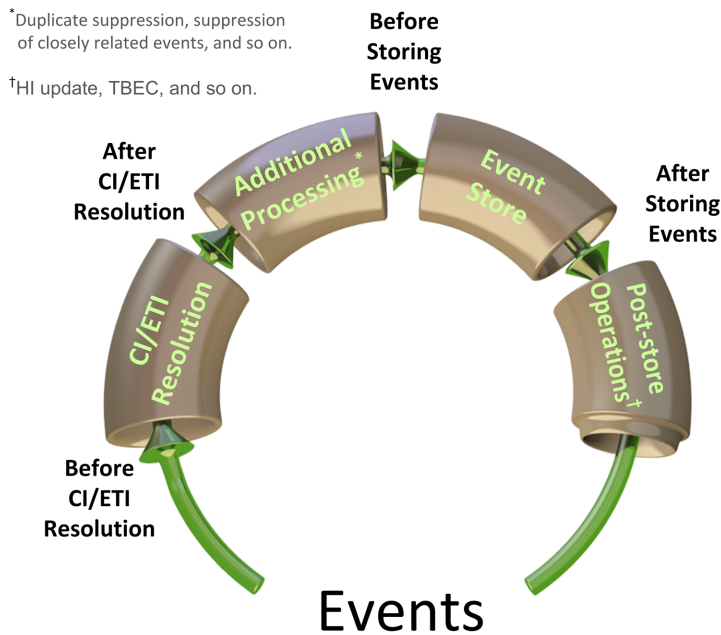
`hpbsm_opr-scripting-host`

- Logging information is written to the following log file:

`<OMi_HOME>/log/opr-scripting-host/opr-scripting-host.log`

EPI scripts are triggered automatically every time the scripting host process is executed. You can configure the location in the event pipeline process flow where the script is executed when you configure the script in the OMi Console. You can choose several locations, as visualized in the figure below:

1. Before CI/ETI Resolution: The script is executed when the event enters the event pipeline, before CI/ETI resolution.
2. After CI/ETI Resolution: The script is executed immediately after CI/ETI resolution
3. Before Storing Events: The script is executed after all event processing has taken place, but before the event is stored in the database.
4. After Storing Events: The script is executed after the event has been stored in the database. All scripts configured in this location are executed in read-only mode.

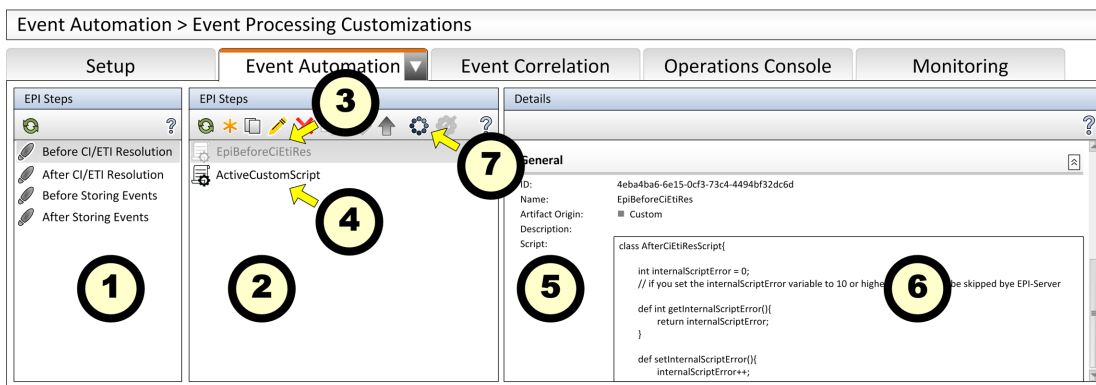


Script Management

Scripts are stored in the database, and can be accessed from the OMi Console.

To configure an event forwarding script and activate it:

1. In OMi, navigate to Event Processing Customizations:
Administration > Event Processing > Automation > Event Processing Customizations
The Event Processing Customizations screen looks similar to the figure below.



The screen contains three panes:

a. *EPI Steps* ①

This pane lists the possible locations in the event pipeline where scripted actions can be added to the event processing. The listed locations correspond to the locations shown in the figure above.


b. *Event Processing Scripts* ②

This pane lists all scripts that have been configured for the selected location. When you select a location in the EPI Steps pane, a list of scripts is shown with gray or black text. Scripts shown in gray ③ are not active, scripts in black ④ are.

c. *Details*

This pane shows the properties of the actual script, including an editor field with the script code ⑥.

By default, some sample scripts and placeholders are present in the database. HPE recommends using the sample classes provided as a starting point when writing a new script to ensure that all required methods are available.

- After your script is finished activate it by either selecting it and clicking the  **Activate Item** button, or by right-clicking it and selecting **Activate Item**. The script is now executed every time the event pipeline is processed.

Format

When writing an EPI script, you must apply the following rules:

- A script *must* be contained in a class.
 - You can choose your own name for the class. The name of the class(es) you configure are added to the *Event Processing Scripts* pane in the OMi administration console.
 - The class does not inherit any properties or methods.

Example of a class declaration:

```
class EpiBeforeStoringEvents{ ... }
```

- If your script class includes public `init()` and `destroy()` methods, these methods are called when the script class is instantiated and destroyed, respectively. You can use these methods to implement customized initialization and clean-up actions that should be called only once (see also

["Best Practices" on page 337](#)).

3. The script class *must* have a public method named `process(eventList)`. This method is called by the platform when the script is executed. The argument `eventList` receives an array of `Event` objects you can step through using a `for` statement (see the example in this section).
4. If a script class has a public method named `getInternalScriptError()` and the method returns a value of 10 or higher, the script is skipped by the platform service executing the script. To use this feature, insert the following code:

```
int internalScriptError = 0;
def int getInternalScriptError() { return internalScriptError; }
def setInternalScriptError() { internalScriptError++; }
```

This code causes the member variable `internalScriptError`, which holds the script error level, to be initialized to 0, and enables the programmer to increase the error level by calling the `setInternalScriptError()` method whenever an error occurs.
5. You may include any other methods, properties, and members as needed.

Best Practices Specific to EPI Scripts

To minimize the impact of EPI scripts on the performance of OMi processing, consider the following aspects:

- Be aware of the number of scripts loaded at any one time. Loading a script introduces overhead until the scripts is unloaded, so loading too many scripts at any one time could lead to performance problems.
- Use background threads and cache information where possible (see also ["Best Practices" on page 337](#)).
- If you need data from the RTSM (or any other external files or databases), open the connection to the RTSM in the `init()` method and close the connect in the `destroy()` method (see also ["Best Practices" on page 337](#)).
- Minimize the processing time required to execute the script's `process()` method. Especially EPI scripts should be written for absolute speed. See also ["Best Practices" on page 337](#).
- EPI scripts will have a negative impact on the overall event processing if they experience problems with external resources such as DNS availability, network availability, and slow file access, especially if the `process()` method uses those resources.
- Access the RTSM using its JAVA API (see also ["Best Practices" on page 337](#)).
- You can configure the order in which scripts are executed when you configure the script in the OMi Console. Scripts can be moved up or down in the sequence using the up and down icons in the *Scripts Panel* toolbar. By default, scripts are marked read/write; use the read-only icon to mark a script read-only. Whether scripts are read-write or read-only has the following impact on when they are executed relative to other scripts:
 - Scripts that need to modify events need read/write access to the database. They are marked read/write and are executed in the order they are configured to avoid potential conflicts.
 - Scripts in read-only mode may be executed in parallel (asynchronously) with read/write scripts.

This means that read-only scripts have less impact on event throughput than read-write scripts. Therefore, mark scripts read-only wherever possible. To mark a script read/write, use the read/write icon.

Note: Scripts located at the After Storing Events process step are *always* executed in read-only mode, because events stored in the database can no longer be modified by an EPI script.

- Scripts using an event filter may have slightly more impact on overall event throughput. Balance the use of filters against the number of events you would have to load without filtering.
- You can configure the timeout value for each script in the OMi Console.
 - Scripts running in read/write mode (synchronously) abort when the timeout time has elapsed, and any changes the script made to events are rolled back.
 - Scripts running in read-only mode (asynchronously) simply abort when the timeout time has elapsed.

HPE recommends configuring a timeout for EPI scripts to minimize potential impact of any single script experiencing problems on the overall OMi pipeline, thereby minimizing the risk of process disruptions.

Configuring a timeout is especially important for scripts using external resources. The best timeout value must be ascertained by testing the script under conditions where access to external resources the script depends on are problematic. When testing scripts, look at the following log file to see how long each script is taking to execute for each event it receives:

- Do not store too much data in memory (see also JVM statistics).
- Release resources when they are no longer needed, and watch for memory leaks during debugging.
- Check the validity of resource handles and reconnect is necessary (see also ["Best Practices" on page 337](#)).

Example

The following sample script applies to the hypothetical situation where critical events are escalated and managed in an OMi-external escalation database. The script retrieves the owner of escalated events from the escalation database and adds it to the OMi event information.

Note: In the example, `external.api.scripting.dataBase` and `Database externalDB` are examples used to demonstrate access to an OMi-external database. The database classes are not provided by OMi; to implement the example, you must first create the classes yourselves.

```
import com.hp.opr.api.scripting.Event;
import external.api.scripting.dataBase;

class actionOnStatus
  /* A script must always be declared as a class.
   * The name of the script class is the name to be used as the name of
   * the Event Processing Script in the OMi Administration console. */
{
  int internalScriptError = 0;
  // Note: The EPI server does not execute scripts
  // if their getInternalScriptError method returns
  // a value of 10 or higher.
  def int getInternalScriptError()
  { return internalScriptError; }

  def setInternalScriptError()
```

```
        // Use this method to increment the internalScriptError.
    { internalScriptError++; }

    Database externalDB;
        // Handle to an external database
        // containing the data used to
        // enrich events, in this case
        // information about an escalation
        // process.
    def connectionString = "MyCloud,MyUsername,MyPwd";

    def init()
    {
        // Actions to be taken when the Groovy script is compiled,
        // which happens once, when the EPI is activated.
        externalDB = connectDB(connectionString);
        if(!externalDB)
        {
            throwDatabaseException(
                "Could not connect to external database"
                + " with connection string "
                + connectionString + "."
            );
            setInternalScriptError();
        }
    }

    def destroy()
    {
        // Actions to be done before destroying the script cache.
        externalDB.disconnect()
    }

    /* The process() method is called every time the EPI server processes
    * the event pipeline.
    *
    * In this example the process() method retrieves the escalation owner
    * of the related CI for all critical events in the pipeline from an
    * external database, and uses the information to set the event's OmUser
    * property. */

    def process(eventList)
        // The eventList argument passes an array containing
        // the events in the event pipeline.
    {
        try
        {
            for( Event in eventList )
            {
                String stringOmUser = "";
                String stringQuery = "";

                int eventSeverity = Event.getSeverity();
                if( eventSeverity == CRITICAL)
                {
                    def RecordSet rsEventInfo;

                    stringQuery = "SELECT escalationOwner AS eo "
                        + "FROM servers WHERE hostname = "
```

```
        + """" + Event.getRelatedCiHint() + """" "
        + "SORT BY contactPriority ASC";

    rsEventInfo = externalDB.Open(StringQuery);
    if( rsEventInfo.RecordCount > 0 )
    {
        stringOmUser =
            "Primary escalation owner: "
            + rsEventInfo.Fields("eo");
    } else {
        stringOmUser =
            "No escalation owner for CI: "
            + Event.getRelatedCiHint();
    }

    setOmUser(stringomUser);
}
// Anything else that needs to be done when this
// event is processed goes here.
}
}

catch( DatabaseException e )
{
    setInternalScriptError();

    // Anything else that needs to be done when this
    // exception is thrown goes here.
}

finally
{
    // Actions to be done after all events are processed.
    // In this example, nothing is needed here.
}
}
```

Custom Action Scripts

Custom actions can be executed for events that are selected in the OMi Event Browser.

This section includes:

- ["Triggers" below](#)
- ["Process Flow" on the next page](#)
- ["Script Management" on the next page](#)
- ["Examples" on page 349](#)

Triggers

Custom actions are launched manually by an operator for a particular event. To launch a custom action:

1. In OMi, select a perspective containing Event Browser and Actions windows, for example the default Event Perspective or Health Perspective.

2. Select the event you want to apply the action to. All actions available for the selected event are listed in the Actions window.
3. Click the action to be executed.

Process Flow

- Scripts are executed in the following working directory on the OMi gateway server:
hpbsm_opr-scripting-host
- Logging information is written to the following log file:
<OMi_HOME>/log/opr-scripting-host/opr-scripting-host.log

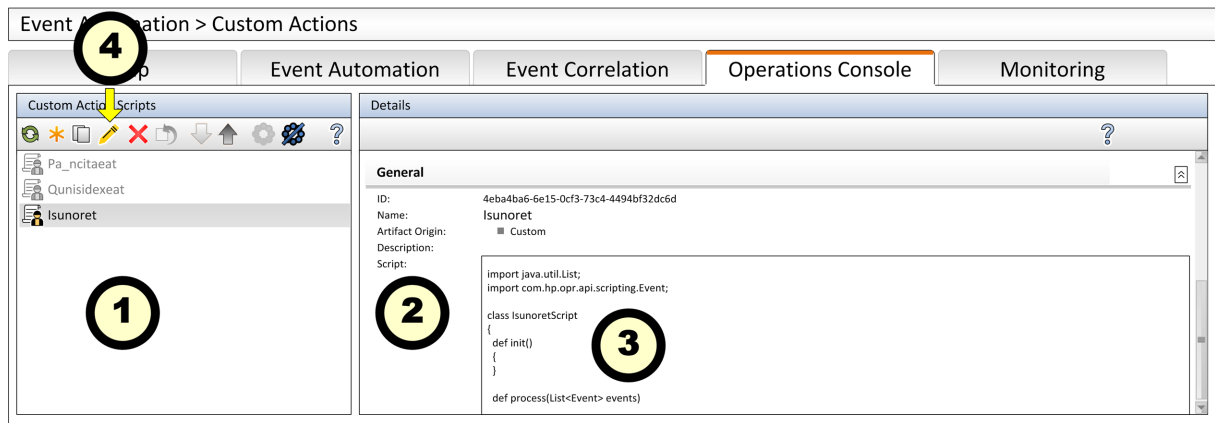
Script Management

To configure a Custom Action script and make it available from the Event Browser:

1. In OMi, navigate to Custom Actions:

Administration > Operations Console > Custom Actions

The Custom Actions screen looks similar to the figure below.



The screen contains two panes:



- a. *Custom Action Scripts* ①

This pane lists the configured scripts. Active scripts are shown in black, non-active scripts in gray.


- b. *Details* ②

This pane shows the properties of the actual script, including the script code ③.

2. You cannot edit the code in the script window directly, but you must use the toolbar ④ in the Custom Action Scripts pane to access the script, as follows:

- To edit an existing script, click the  **Edit Item** button. The script wizard is shown. Click the **Script** tab to access the script for editing.
- To configure a new script, click the  **New Item** button. The script wizard is shown, allowing

you to enter the script and some information about it, such as a name and description. The wizard provides a skeleton for the script class, which you can use as a starting point.

- To activate a script, click the  **Activate Item** button.

Examples

```
import java.util.List;
import com.hp.opr.api.scripting.Event;

class IsunoretAction
{
    def process(event) // Process the event passed
                        // by the event browser.
    {
        // Example: Service ID of the event is
        // manually set tpo a predefined value."

        event.setOmServiceId("Isunoret");
    }
}
```

Certificates Scripts

From version 9.2x, you can automatically process certificate requests using a script. The following functions are available:

- Grant the request.
- Deny the request.
- Ignore the request, leaving it to be granted or denied manually.

A Groovy script dealing with certificate requests is provided with the installation (see ["Example" on the next page](#)), which can be used as the basis for a custom scripts.

This section includes:

- ["Process Flow" below](#)
- ["Script Management" on the next page](#)
- ["Example" on the next page](#)

Process Flow

The certificate script class is instantiated the first time a certificate is requested. The `process()` method is called every time a certificate is requested.

- Scripts are executed in the following working directory on the OMi gateway server:

hpbsm_opr-scripting-host

- Logging information is written to the following log file:

<OMi_HOME>/log/opr-scripting-host/opr-scripting-host.log

Script Management



Scripts are stored in the database, and can be accessed from the OMi console.


To configure a certificate script:

1. In OMi, navigate to the Certificate Requests manager:

Administration > Setup and Maintenance > Certificate Requests

The **Certificate Requests** tab lists all certificate requests that have been received during the time range selected in the drop-down list.

2. Click the **Autogrant Script** tab. The default Autogrant Script is shown, which grants or denies certificates based on the value of the request's `nodeName` attribute. The default script is not activated by default.
 - To customize the script, click the  **Edit Item** button, or double-click the script window, and make the required changes.
 - To activate the script, click the  **Activate Item** button.

Only one script can be configured as certificate script. The script is activated as soon as it is configured. If no certificate scripts is needed, deactivate the Autogrant Script script by clicking the  **Deactivate Item**.

Example

The following example grants or denies certificate request based on the requester's subnet.

```
import java.net.InetAddress;
import java.util.Date;
import java.util.List;
import com.hp.opr.api.scripting.CertificateRequest;

class AutoGrantTest
{
    def init()
    {
    }

    def destroy()
    {
    }

    Boolean boolInSubnet( String IPAddress )
    {
        // Check if the passed IP address
        // is in the allowed subnet.
        String stringAllowedSubnet = "128.0";

        ByteOne = IPAddress.indexOf('.');
        ByteTwo = IPAddress.indexOf('.', ByteOne + 1 );

        String stringSubnet;
        stringSubnet = IPAddress.substring( 0, ByteTwo );
    }
}
```

```

return( stringSubnet.equals( stringAllowedSubnet ) );
}

def process(List<CertificateRequest> requests)
{
    // Retrieve the first request object.
    def request = requests.get(0);

    if ( boolInSubnet( request.getIpAddress() ) )
    {
        request.deny();
    }
    else
    {
        request.grant();
    }
}
}
}

```

Service Health Scripts

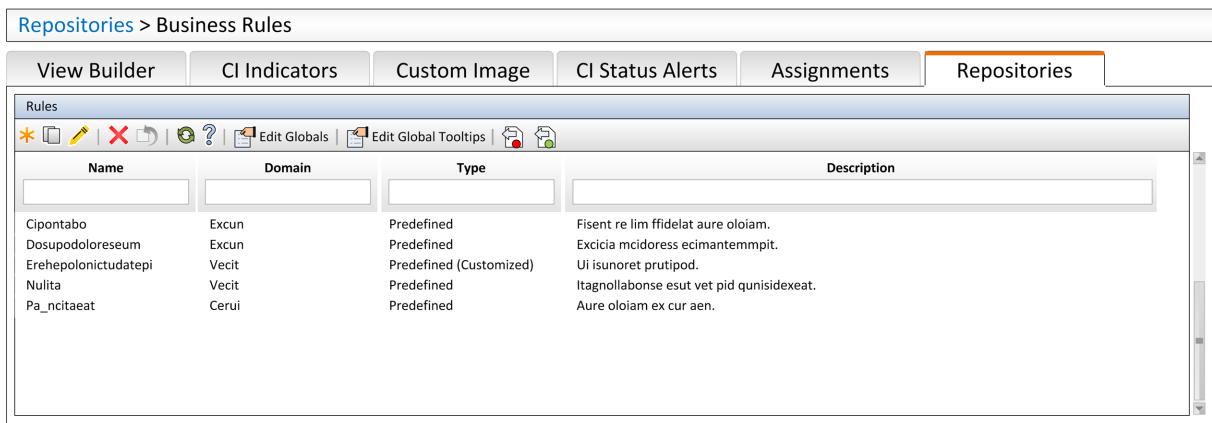
Service health scripts are used to implement business logic rules used to calculate Key Performance Indicators (KPI), which are used by Service Health Analyzer (SHA) to evaluate the health status of managed networks, their components, and the services associated with them.


Service health scripts are executed as part of the Marble process.


Script Management



To configure a business rule and use it for KPI calculation:

- In OMi, navigate to Business Rules:
Administration > Service Health > CI Status Calculation > Business Rules
- The Business Rules screen looks similar to the figure below.



The screen contains a list of business rules, some of which are predefined as indicated by the **Type** column. You can add new rules by clicking the **New** icon  or customize a rule by selecting

it and clicking the **Edit** icon . For detailed information about creating business rules, see the *OMi Administration Guide*.

3. After creating a rule, you can include it in a KPI. To create or customize a KPI, click the **Repositories** tab and select the **KPIs** link to show a list of KPIs. Create a new KPI by clicking the **New** icon  or customize a KPI by selecting it and clicking the **Edit** icon . The new or customized rules are now included in the Unselected Rules or Selected Rules lists in the KPI Editor dialog. For detailed information about creating KPIs, see OMi Help.

Topology Synchronization Scripts

The topology synchronization package enables you to include user-defined Groovy scripts in the topology synchronization process.

Scripts can be used to trigger custom actions as part of the topology synchronization process, such as:

- Manipulate service hierarchies beyond transformation and enrichment limitations posed by model.
- Perform post-synchronization actions.
- Modify topology data without the need to alter the topology model in OM.

This section includes:

- ["Process Flow" below](#)
- ["Script Management" on the next page](#)
- ["Best Practices Specific to Topology Synchronization Scripts" on the next page](#)
- ["Example" on page 354](#)

Process Flow

Scripts are triggered automatically as part of the `hpbsm_wde` process.

Logging information is written to the following log file:

```
<OMi_HOME>/log/wde/opr-svcdiscserver.log
```

Caution: If execution of an automatically triggered script negatively affects the synchronization process, both the topology and event synchronization processes are at risk of being disrupted. Adhere to the best practices described in ["Best Practices" on page 337](#) as closely as possible to minimize this risk.

The name of a script file determines where in the topology synchronization process it is executed. Three locations in the process trigger script files, provided the naming of the script files adheres to the following convention:

1. If present, the script named `preEnrichment.groovy` is executed before the mapping of OM nodes and services.
2. If present, the script named `preUpload.groovy` is executed after OM nodes and services are mapped, but before they are uploaded to the uCMDB.
3. If present, the script named `postUpload.groovy` is executed after the nodes and services are uploaded to the uCMDB.

Script Management

Topology Synchronization scripts are an integral part of the Topology Synchronization package, and are stored in the database when the package is installed. The topology synchronization process executes the scripts residing in the database without referring to the scripts in the file system, so if you make changes or additions to the scripts in the file system you must upload the Topology Synchronization package again.

To add, remove, or change a script:

1. Place the completed or changed script in the following folder:

```
<OMi_HOME>/conf/opr/topology-sync/sync-packages/<Sync Package Name>
```

To remove a script:

- a. Use the `sdtool` command to remove the script from the database.
- b. Delete the script from the file system.

Note: Scripts installed with the standard installation are present in the file system. Do not delete these scripts, because they are used by the system.

2. Use the `sdtool` command to upload the synchronization package to the database.

Best Practices Specific to Topology Synchronization Scripts

When developing topology synchronization scripts, consider the following aspects:

- In a multi-server environment particular care must be taken to keep the scripts in sync. After uploading a synchronization package including new or modified scripts to the database, the synchronization process on all server is changed according to the scripts residing in the file system of the server used to upload the package. The scripts residing in the file system of all other servers, however, no longer match the scripts in the database, so if one of these servers is used to upload scripts, the synchronization may revert to an earlier, undesired state. To keep this from happening, HPE recommends using the same server for managing topology synchronization scripts.
- Pre-upload and post-upload scripts included in the same sync package share their scope with respect to variables. Therefore, values assigned to variables in the `preUpload.groovy` script can be used in the `postUpload.groovy` script in the same package. Scripts across different sync packages do not share the same scope, so the same variable names may be used in scripts as long as they are not part of the same synchronization package.
- Scripts always have access to the `scriptInterface` and `syncData` objects. The API functions provided by these objects encapsulate the internal objects. Always reference these objects rather than accessing the internal objects directly. For example, use `myCI = createCI(caption:"myCI")` to return an `ICi` implementation without exposing the `CI` class, rather than using `new CI(...) myCI`. See also ["Available APIs" on page 365](#).
- Use only Java interfaces, not classes. Additionally, a set of special factory functions can be used. See also ["Available APIs" on page 365](#).

Example

Suppose that the captions of services that contain the string `disk` are to be highlighted by adding the prefix `>>>` and the suffix `<<<`, and the number of changes made is to be reported via email to the Administrator.

Adding the prefixes and suffixes can be done after the OM nodes are mapped. The following `preUpload.groovy` script changes the captions and stores the number of changes in the variable `iChanges`:

```
iChanges = 0;

// Get all the resolved services using the getConfigurations()
// method of the syncData object and store them in the array services.
arrayServices = syncData.getConfigurations();

// Check all elements of the array services for the
// substring "disk", and if found add pre- and suffix and
// increment the counter.
arrayServices.findAll{ it.getCaption().indexOf( "disk" ) >= 0 }.each
{
    stringCaption = svc.getCaption();
    svc.setCaption( ">>> " + stringCaption + " <<<" );
    iChanges++;
}
```

The email is sent after the upload is completed, as part of the `postUpload.groovy` script (Note that the `sendmail` command in the last example is not available on a standard Windows platform). Because this script is included in the same synchronization package as the `preUpload.groovy` script, the variable `iChanges` is still in scope.

```
msg = "Sync Ready! ${iChanges} CIs modified!";
scriptInterface.exec( "sendmail Administrator " + msg );
```

Event Forwarding Scripts

Event forwarding scripts forward events to Northbound applications. The main use of event forwarding scripts is to build adapters to external systems.

Event forwarding scripts tend to be more complicated than other scripts, because they need to be able to handle the following types of operations:

- Forwarding events arriving in the pipeline.
- Forwarding updates to events.
- Receiving and forwarding changes to events.
- Supporting correlation of events to processes running on the external server receiving the forwarded events.
- Supporting the exchange of external information required by the script, such as information gleaned from web service calls, information provided by command line tools, or information written to log files by the script.

Note: Event scripts are not used to enrich events; for information about scripts to enrich events, see ["Event Processing Interface Scripts" on page 341](#).

This section includes:

- ["OprEvent Class" below](#)
- ["Process Flow" on the next page](#)
- ["Script Management" on the next page](#)
- ["Example" on page 357](#)

OprEvent Class

Unlike other scripts, Event Forwarding scripts use the concrete class `OprEvent`.

The `OprEvent` class strictly limits code used to modify event information to customizations of the `receiveChange()` method, which is called when the target server notifies an event change.

The following code is a valid declaration for an implementation of an event forwarding script:

```
package com.hp.opr.api.ws.adapter;
import com.hp.opr.api.ws.model.event.OprEvent;

public abstract interface ExternalProcessAdapter
{
    void init(final InitArgs args);
    void destroy();

    Boolean ping(final PingArgs args);

    /* The following method can be customized */
    Boolean receiveChange(final ReceiveChangeArgs args);

    Boolean getExternalEvent(final GetExternalEventArgs args);
    Boolean forwardEvent(final ForwardEventArgs args);
    Boolean forwardChange(final ForwardChangeArgs args);
    String toExternalEvent(final OprEvent event);
}
}
```

The `OprEvent` class is imported, and the script class (which is called `ExternalProcessAdapter` here, but may have any name) has the standard `init()` and `destroy()` methods. The following methods implement event forwarding:

- The `ping()` method tries to contact the connected server and returns `true` if successful.
- The customizable `receiveChange()` method receives the external changes, and enables OMi to synchronize changes made by the external server. The `receiveChange()` method is called by the Event Synchronization web service to respond to the following HTTP requests:
 - PUT requests to:
`/opr-gateway/rest/synchronization/event/<event ID>`
 - POST requests to:
`/opr-gateway/rest/synchronization/event_change/<event ID>`

The HTTP header `Content-Type` must be set to one of the following values:

- application/xml
- application/json
- application/soap+xml
- text/xml
- text/plain

Note: If the connected server does not support synchronization the `receiveChange()` method should set the HTTP response code in the argument to 403 (Forbidden), and return `false`.

The `receiveChange()` method should return `true` if it executed successfully and the values passed in the argument have been updated.

- The `getExternalEvent()` method retrieves event details in response to user requests made using the Event Browser for the events that have transferred control to a process on the external server. The values corresponding to the event details are received in and passed by the method's argument `GetExternalEventArgs`.
- The `forwardEvent()` method forwards the event passed in the method's `ForwardEventArgs` argument to the external process.
- The `forwardChange()` method forwards the event changes that have occurred since the last successful call to this method for the event passed in the method's `ForwardChangeArgs` argument. The first time the method is called for a particular event all changes since the event was forwarded are included.
- The `toExternalEvent()` method converts the event to an external object, and is called by the Event Synchronization web service to respond to the following HTTP request:

GET request to:

`/opr-gateway/rest/synchronization/event/<event id>`

Process Flow

Event forwarding scripts are triggered when a change arrives from the target server as part of the Event Forwarding process on the gateway server.

- Scripts are executed in the following working directory on the OMi gateway server:

`hpbsm_opr-scripting-host`

- Logging information is written to the following log file:

`<OMi_HOME>/log/opr-scripting-host/opr-scripting-host.log`

Script Management

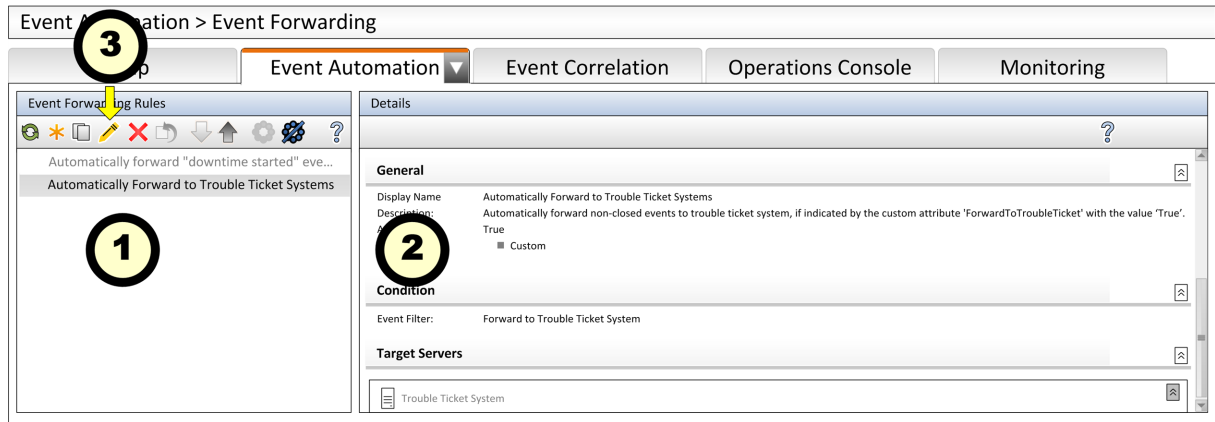
Scripts are stored in the database, and can be accessed from the OMi Console.

To configure an event forwarding script and activate it:

1. In OMi, navigate to Event Forwarding:

Administration > Event Processing > Automation > Event Forwarding

The Event Forwarding screen looks similar to the figure below.



The screen contains two panes:

- a. *Event Forwarding Rules* ①
This pane lists the configured event forwarding rules. Active scripts are shown in black, non-active scripts in gray.
 - b. *Details* ②
This pane shows the properties of the selected rule.
2. You cannot edit the code in the script window directly, but you must use the toolbar ③ in the Event Forwarding Rules pane to access the script, as follows:
- To edit an existing rule, click the **Edit Item** button. The Edit Event Forwarding Rule dialog is shown. Customize the rule and click **OK**.
 - To configure a new rule, click the **New Item** button. The Edit Event Forwarding Rule dialog is shown. Enter the information for the rule and click **OK**.
 - To activate a rule, click the **Activate Item**.

By default, the following rules are installed:

- Automatically forward "downtime started" events to Trouble Ticket System
- Automatically Forward to Trouble Ticket System

Example

A sample LogfileAdapter script is provided with the system.

External Instruction Retrieval Scripts

External instruction retrieval scripts are used to get the event instructions from external interfaces, such as knowledge bases or databases. OMi triggers the execution of an instruction retrieval script

when at least one specified event filter matches the event selected in the Event Browser (the instructions are retrieved when you select the **Instructions** tab in the Event Details pane).

This section includes:

- ["Process Flow" below](#)
- ["Script Management" on the next page](#)
- ["Script Output" on the next page](#)
- ["Examples" on page 360](#)

Process Flow


- Scripts are executed in the following working directory on the OMi gateway server:

hpbsm_opr-scripting-host

- Logging information is written to the following log file:

<OMi_HOME>/log/opr-scripting-host/opr-scripting-host.log

When the script execution is triggered by OMi, the event is passed into the script. The script can access all properties of the event including the specified instruction parameters, such as the instruction interface name (property 'event.instructionInterfaceName') and the parameter string (property 'event.instructionParameterString'). As a parameter, you can enter any option that you want to pass to the external program generating the instruction text, for example a policy variable or a policy parameter configured in the policy template.

The **Instruction Available** flag indicates whether the instructions are available for the event (the  icon in the **I** column for the selected event is displayed). This property is set to true at the time the event is created and stored in the database if the corresponding message from the agent (received on the OMi server and converted to the event) contains one of the following:

- The UUID of the instruction text that was specified in the policy
- The name of an instruction text interface to be used for the instruction text retrieval

Additionally, before the events are loaded into the Event Browser, OMi evaluates all active external instruction text interface definitions and checks whether their event filters match the event. If at least one event filter matches, the **Instruction Available** event flag is also set to true.

Note: In situations where you disable or delete the matching external instruction text interface definition and restart the OMi server, the **Instruction Available** flag of related events is no longer set to true.

The mapping to an instruction text interface can be achieved in one of the following ways:

- By creating one event filter that matches everything and defining one mapping from this event filter to a single Groovy script. Depending on the instruction interface name of the event that is passed to the script, it is possible to retrieve the instruction text from different sources.

Tip: To avoid generating a high number of events having the **Instruction Available** property set to true but containing no actual instructions, define the event filter so that it only matches the events for which the instruction text *is* available.

To facilitate the event filtering, you can use the **Instruction Interface Name** and **Instruction Parameter String** Advanced Filter elements. This way, you limit the set of events displayed in

the Event Browser to those with the specified instruction interface and parameter string. For more information about event filters, see the *OMi User Guide*.

- By creating multiple event filters and mappings to Groovy scripts. The first event filter that matches will call the mapped Groovy script. If you choose this alternative, define event filters so that they do not match the same set of events.

Note: OMi always chooses the first event-matching filter. As it is not possible to define the order in which event filters are evaluated, you must only define disjunct event filters, that is, the event filters that match different sets of events.

Script Management

Scripts are stored in the database and can be accessed from the OMi console.

To configure an instruction retrieval script:

1. Navigate to the External Instructions manager:

Administration > Operations Console > External Instructions





The screen contains two panes:

- a. *Scripts*

This pane lists the configured scripts. Active scripts are shown in black, inactive scripts in gray.

- b. *Details*

This pane shows general and advanced properties of the selected script, including the scripts themselves.

2. Use the toolbar in the Scripts pane if you want to:
 - Edit an existing script. Click the  **Edit Item** button to open the script wizard, and then click the **Script** tab to access the script for editing.
 - Configure a new script. Click the  **New Item** button to open the script wizard. Enter the script and some information about it, such as a name and description. The wizard provides a skeleton for the script class, which you can use as a starting point.
 - Activate/deactivate a script by clicking the  **Activate Item** or  **Deactivate Item** buttons.

Script Output

The event, including the related instructions, can be viewed in the OMi Event Browser. The instructions are displayed on the **Instructions** tab in the Event Details pane.

The Groovy script returns the output in the plain text or the HTML format. If the instruction text contains URLs, they are automatically converted into hyperlinks. The URLs starting with `http://`, `https://`, `ftp://`, `fttps://`, `telnet://`, and `mailto:` URI scheme names are supported. Note that you can add URLs of external Web sites, support sites, documentation repositories, troubleshooting information, and so on.

You can also type the address of any Web site that begins with `www`.

In the Event Browser, when you click the hyperlink, a new window opens with the page that was returned by the script.

Examples

OMi delivers a predefined content pack containing a sample instruction retrieval script and a filter for external instruction interfaces. A sample policy is included as well. You can use this initial script as a basis for developing customized scripts to better suit your business needs. For more information on content packs, see the *OMi Administration Guide*.

This section provides two instruction retrieval Groovy scrip samples.

A sample instruction retrieval Groovy script:

```
import com.hp.opr.api.scripting.Event

class GroovyScriptSkeleton
{
    // called before the method "getInstructions" is called
    def init()
    {}
    // called after the method "getInstructions" is called
    def destroy()
    {}
    // This method gets as parameter the event for which instruction text should be retrieved and
    // returns the instruction text that is displayed in the event browser.
    def getInstruction(Event event)
    {
        if ("iti" == event.instructionInterfaceName)
        {
            // the command to execute; it gets the parameter string (as send by the agent) as argument
            // so, if the agent sends the name of the agent system, this command will ping the agent system
            // and the output of the ping command is returned as instruction text
            def command = "ping -n 1 " + event.instructionParameterString
            def proc = command.execute() // execute the command
            proc.waitForOrKill(3000) // wait for 5sec before time-out
            if (proc.exitValue() == 0)
                return "ping of node '" + event.instructionParameterString + "': " + proc.in.text
            else
            {
                if (event.instructionInterfaceName == null)
                    return "No instructions are available for event with ID '" + event.id + "'."
                else
                    return "Cannot get instructions for interface '" + event.instructionInterfaceName + "'."
            }
        }
    }
    // This method gets as parameter the list of events for which instruction text should be retrieved.
    // The instruction text for all events is returned as map with the ID of the event as key and the instruction
    text for this event as value.
    def getInstructions(List<Event> events)
    {
        events.collectEntries { event ->
            [ event.id, getInstruction(event) ]
        }
    }
}
```


A sample instruction retrieval Groovy script with the HTML output:

```
import com.hp.opr.api.scripting.Event

class GroovyScriptSkeleton
{
    def init()
    {}

    def destroy()
    {}

    def getInstruction(Event event)
    {
        if ("iti" == event.instructionInterfaceName)
        {
            // add your code to retrieve the instruction text from an external source
            return """<html><body><h1>My First Heading</h1><p>My first paragraph.</p><a
href="http://www.google.com">This is a link</a>
</body></html>"""
        }
        else
        {
            return "Cannot get instruction text for interface '" + event.instructionInterfaceName + "'."
        }
    }

    def getInstructions(List<Event> events)
    {
        events.collectEntries { event ->
            [ event.id, getInstruction(event) ]
        }
    }
}
```

Chapter 39: Reference Information

This section contains reference information related to Groovy scripts, or links to reference information in case the information itself is already documented elsewhere.

The following information is included:

- "[The Groovy Console](#)" below describes how to install the Groovy console, and how to prepare it for developing scripts for OMi. It also includes tips on debugging scripts.
- "[Available APIs](#)" on page 365 lists internal and external APIs that may be useful when developing Groovy scripts for OMi.
- "[The opr-script Command-Line Interface](#)" on page 367 describes how to enable or disable a Groovy script.

The Groovy Console

The best way to develop and debug Groovy scripts is by using the Groovy console.

Installation

To prepare your environment for developing Groovy scripts:

1. Install Java.

Groovy requires Java version 1.4 or higher. If not already installed, get the latest Java distribution from the following URL:

<http://java.sun.com>

Run the installer and set the environment variable `JAVA_HOME`. On Windows, follow these steps:

- a. Open the **Control Panel**, and go to section **System and Security > System**.
 - b. Click **Advanced System Settings**. The *System Properties* dialog opens.
 - c. Click the **Environment Variables** button. The *Environment Variables* dialog opens.
 - d. Click **New...** underneath the **User variables** box and add a variable `JAVA_HOME` and assign the path where Java is installed as its value.
 - e. Select the variable `%Path%` in the **System variables** box, click **Edit...** and add the variable `%JAVA_HOME%` to the path.
2. To install the Groovy console, obtain the installer for your operating system from the download page at the official Groovy web site at the following URL:

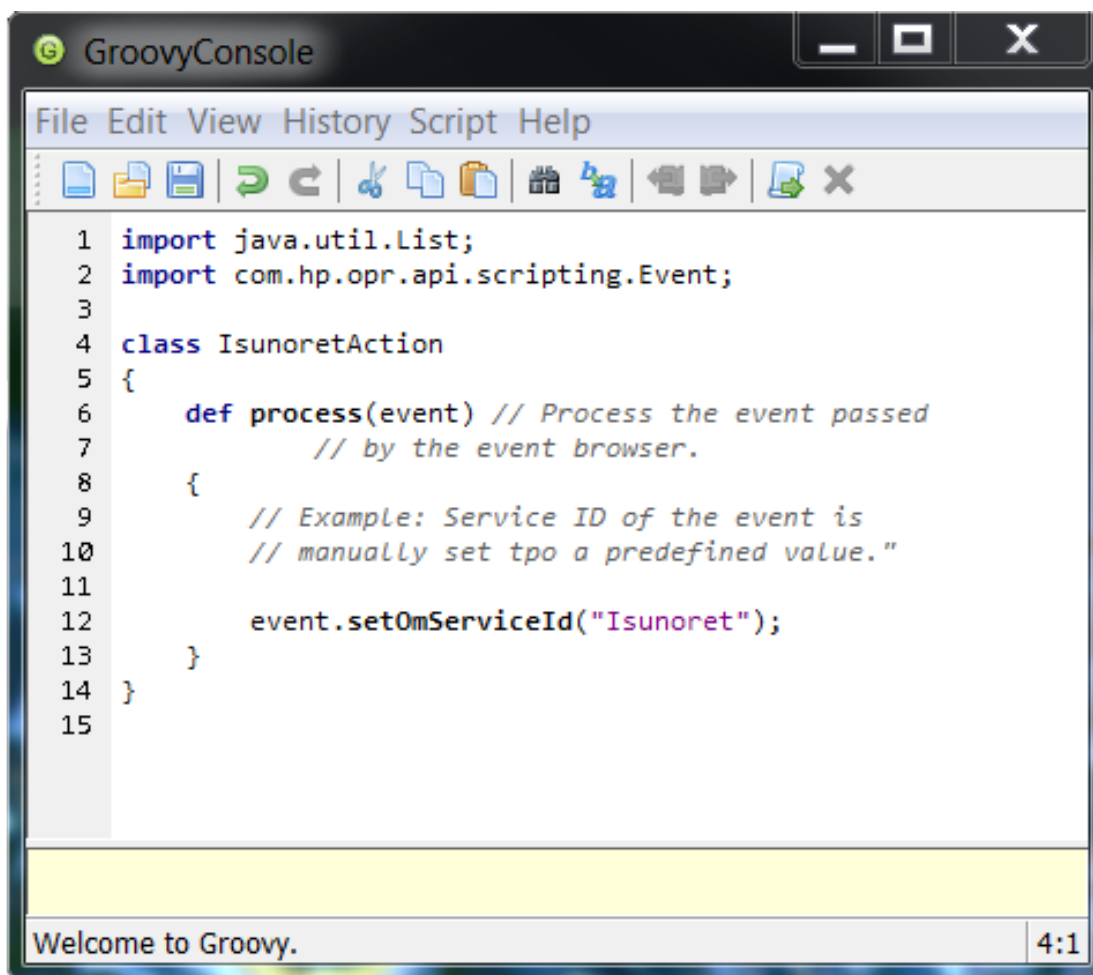
<http://groovy.codehaus.org/>

By default, the installer adds the environment variable `GROOVY_HOME` holding the installation path to your system and adds the path to the binaries to your path. In the rest of this section, `%GROOVY_HOME%` is used to denote the installation path.

Note: The installer has had some issues with spaces in the installation path. HPE

recommends installing the console in a directory without spaces in the path.

The Groovy Console looks like the following figure:



3. When running the Groovy console on a server other than the OMi Data Processing Server or Gateway Server you must copy the platform libraries to a predefined location. To make the libraries available to the Groovy console:
 - a. Open the file %GROOVY_HOME%\conf\groovy-starter.conf in a text editor.
 - b. Make sure a line with the following format is active (if necessary, add or uncomment it):

```
load !{user.home}/.groovy/lib/*.jar
```
 - c. If you made changes, save the file.
 - d. Copy the libraries to the directory corresponding to user.home, where user.home denotes the directory where the library files reside.

If in doubt, copy the libraries to the following directory, which is always loaded:

```
%GROOVY_INSTALL%\lib
```

For OMi customizations, the following libraries are often needed:

- Apache Commons Libraries
<install>/lib/commons-*.jar
This library Includes general functions such as the logging function log4j.
- JSON Library
<install>/lib/json.jar
For parsing and generating JSON code.
- <install>/lib/opr-external-api.jar
- <install>/lib/opr-common.jar
- Apache Wink REST Web Services APIs
<install>/lib/wink-*.jar
- JSR-311 REST APIs
<install>/lib/jsr311-api.jar
- RTSM Topology APIs
<install>/lib/ucmdb-api.jar

Compatibility with OMi

OMi supports Groovy versions listed in the following table. Features contained in later versions of Groovy are not supported.

OMi Version	Latest Compatible Groovy Version
10.10	2.4.5
10.0x	2.2.1
9.13, 9.2x	1.7.3
9.10, 9.11 and 9.12	1.6.0 for EPI and Custom Action scripts. 1.7.3 for other scripts.
9.01	1.6.0 for EPI and Custom Action scripts. 1.5.6 for other scripts.

Debugging Using the Groovy Console

The following tips can help you when testing and debugging Groovy scripts:

- The Groovy console is an interpreter. Certain errors, such as references to classes without `import`, are not reported until the line of code is actually executed.
- Make sure you have an `import` statement for each class used in the script.
- Use extensive logging while debugging. Logging can be a very powerful tool to assist in debugging

but be careful not to log unnecessary information. Too many log entries can make it hard to find entries for a specific issue, as well as cause the logs to roll over too quickly.

Note: Make sure to deactivate the logging code before moving the script to the production environment. Log information that is not to be deactivated in the production environment should be used sparingly. See also ["Best Practices" on page 337](#).

- Your script can be enabled to log to Apache by using the Apache LogFactory class, which is defined in the Apache Common Libraries. To include the log factory in your script class, declare a member variable of type Log and initialize it to the name of the log method for the relevant OMi package, for example:

```
private static Log s_log = LogFactory.getLog("com.hp.opr.epi.MyLogs")
```

Notes for logging to Apache:

- Make sure you declare the member variable as `static` so it is initialized only once.
- The package name `com.hp.opr.epi` should be used if you are using one of the OMi scripts. This will automatically add your log entries to the EPI scripting log file (see ["Event Processing Interface Scripts" on page 341](#)).
- To access a custom package you must supply the properties file for your package. The properties files can be found in a sub-directory in the directory `conf/core/Tools/log4j`. The name of the sub-directory corresponds with the name of the process triggering your script, as detailed under the heading Trigger for each script type described in section ["Development and Deployment of Scripts" on page 341](#).
- To adjust the logging level for a script, edit the logging level entry in the appropriate properties file (see previous bullet point).
- To create an Apache log entry, use the following syntax (provided you used the same name for the Log member):

```
s_log.error("<Error text>")
```

For example:

```
s_Log.error("Attempt to get event failed; HTTP status: ${response.statusCode}")
```

Specify the appropriate error level as the method name (`error()` in the example). To track the behavior of your script, HPE recommends to include a log entry at error level `info()` so you can verify that script loads successfully.

Available APIs

This section details the available platform and external APIs:

- ["External APIs" on the next page](#)
- ["Functions Used in EPI Scripts" on the next page](#)
- ["API for Topology Synchronization Scripts" on page 367](#)
- ["API for Service Health Rules" on page 367](#)

External APIs

API	Applications
Java JDK	<p>Generic JDBC access for functions such as the following:</p> <ul style="list-style-type: none"> • Establishing a connection to the database. • Connecting with data sources. • Handling SQL Exceptions. • Creating and populating tables. • Retrieving and modifying data. • Using database transactions. <p>For details, see http://docs.oracle.com/javase/tutorial/jdbc/basics.</p>
	<p>General Web Service access.</p> <p>Use function <code>URL.openConnection()</code></p>
	<p>Command line tools.</p>
Apache	<p>Providing debugging support by logging at runtime without the need to modify the application binary: Use the Apache Log4j logging API (see also "The Groovy Console" on page 362).</p> <p>For details, see http://logging.apache.org/log4j/1.2/.</p>
	<p>Communication with RESTful Web services: Use the Wink JSR Client API.</p> <p>For details, see http://incubator.apache.org/wink/.</p>
	<p>Leveraging the HTTP protocol to retrieve information in an environment using distributed communication: Use the Apache HTTP Components API.</p> <p>For details, see http://hc.apache.org/.</p>
	<p>Communication with Web services supporting Simple Object Access Protocol (SOAP): Use the Apache Axis2 API.</p> <p>For details, see http://axis.apache.org/axis2/java/core/.</p>

Functions Used in EPI Scripts

The Groovy script API with full documentation of all arguments and types can be found in the Java API Documentation delivered with the product. It can be found in the following directory:

`<OMi_HOME>/opr/api/doc.`

For detailed information about developing EPI scripts, see ["Event Processing Interface" on page 122](#).

API for Topology Synchronization Scripts

For more information about the interfaces and object types required for developing your own topology synchronization scripts, see the following Java API documents:

- `<OMi_HOME>/opr/api/doc/opr-external-api-javadoc.zip`
- `<OMi_HOME>/opr/api/doc/opr-ts-interfaces-javadoc.zip`

The following is a list of externally visible types and functions.

- Exposed Interfaces:
 - `ICi`
 - `ICiRelation`
 - `ICiMessage`
 - `IScriptingInterface`
 - `INavigator`
 - `INode`
 - `ISyncData`
- Exposed Classes:
 - `CiMessageCategory`
 - `CiMessageSeverity`

- The `IScriptingInterface` interface is accessible through the variable `scriptInterface`.
- The `ISyncData` accessible through the variable `syncData`.

For detailed information about developing topology synchronization scripts, see ["Topology" on page 19](#).

API for Service Health Rules

For detailed information about developing service health rules, see ["Event Type and Health Indicators" on page 31](#).

The opr-script Command-Line Interface

You can use the opr-script command-line interface (CLI) to enable and disable Groovy scripts of the following types:

- Custom action scripts
- Event processing interface (EPI) scripts

- External instruction retrieval scripts

Tip: To check whether a script is active, see the Active check box in the details of the corresponding script. When you disable a script using the `opr-script` CLI, the Active check box is cleared.

Location

`<OMi_HOME>/opr/bin/opr-script.[bat|sh]`

Synopsis

`opr-script -help | -version | <CONNECTION_INFO> <operation>`

Options

Syntax for <CONNECTION_INFO>

`-username <login name> [-password <password> | -smartcard | -winCrypto | -jks <keystore path> -jksPassword <keystore password>] [[-port <port>] [-server <gatewayserver>] [-ssl] | [-u <URL>]]`


Note: If <CONNECTION_INFO> is omitted, the command is executed on the server to which you are logged on.

Option	Description
<code>{-username -user}</code> <code><login name></code>	Sets the login name of the user required to execute CLI operations on the target gateway server.
<code>{-password -pw}</code> <code><password></code>	Sets the password for the specified user. If using SSH on Cygwin, either enter the password in free text or use other communication methods, for example Java keystore, Windows keystore, or smart card authentication. Default value: empty string
<code>{-smartcard -sc}</code>	Use certificate stored on smart card or security token for authentication. When OMi is configured to use CAC authentication, the CLI tools under <code><OMi_HOME>/opr/bin/</code> do not directly prompt users to enter the password for the smartcard connected to the system. Instead, users must specify that a smartcard authentication is to be run, using the option <code>-sc</code> or <code>-smartcard</code> . Users attempting to run a tool without the <code>-smartcard</code> option automatically receive an error message. For more information, see Using Command-Line Tools When Client Certificate Enforcement Is Used .

Option	Description
{-winCrypto -wc}	<p>If OMi is configured for TLS mutual authentication, this option specifies to use the Windows certificate store for authentication. The certificate store must hold exactly one client certificate, which OMi will use to authenticate the user. This option is only available on Windows systems.</p> <p>For details, see Configuring Client Certificate or Smart Card Authentication.</p>
{-jks -j} <keystore path>	<p>If OMi is configured for TLS mutual authentication, this option can be used to specify the Java keystore to be used for authentication. The keystore must hold exactly one client certificate, which OMi will use to authenticate the user.</p> <p>Note: It is not necessary that the client certificate contains the flag "Smart Card Logon (1.3.6.1.4.1.311.20.2.2)" in the "Enhanced Key Usage" field.</p> <p>For details, see Configuring Client Certificate or Smart Card Authentication.</p>
{-jksPassword -jp} <keystore password>	<p>Password for accessing the Java keystore.</p>
{-port -p} <port>	<p>Uses port <port> to connect to the target gateway server.</p> <p>Default: 80 (HTTP) 443 (HTTPS)</p>
-server <GatewayServer>	<p>Sets the target gateway server, using <GatewayServer> as the hostname or IP address to locate it.</p> <p>Default: FQDN of the OMi gateway server</p>
-ssl	<p>When this option is specified, the HTTPS protocol is used to connect to the target gateway server. If omitted, the HTTP protocol is used.</p> <p>Cannot be used in conjunction with the -url option.</p>
{-url -u} <url>	<p>Sets the target gateway server, using <url> as the URL to locate it.</p> <p>Default: https://<OMi gateway FQDN>:80/opr-config-server</p>

Syntax for <operation>

-enable|-disable -type <type> -id <id>

Option	Description
<code>{-enable -e}</code>	Enables a previously disabled Groovy script.
<code>{-disable -d}</code>	Disables a Groovy script.
<code>-type <type></code>	Groovy script of the type: <ul style="list-style-type: none"> • CUSTOM_ACTION • EPI • EXTERNAL_INSTRUCTION
<code>-id <id></code>	ID of Groovy script. The ID of the script is displayed in the script's properties in the corresponding UI. For example, to get the ID of a custom action script, navigate to: Administration > Operations Console > Custom Actions Select the script in the Custom Action Scripts pane and click  Edit Item . Copy the ID in the lower right corner of the General page.

Exit Status

Exit Status	Description	Output
0	Successful completion of the requested operation	No output.
1	Failure of the requested operation	An error message stating why the operation failed, followed by the tool's help text.
300-399	HTTP Redirection (300-399)	An error message stating the HTTP error number and description. For more information about HTTP error status values, see publicly available HTTP documentation.
400-499	HTTP Client Error (400-499)	
500-599	HTTP Internal Server Error (500-599)	

Restrictions

The user running the opr-script command-line interface must be an OMi user with permissions to create custom actions, EPI, or external instruction retrieval scripts.

Non-admin users also need the following file permissions to operate this command-line tool:

File	Windows Permissions	Linux Permissions
<OMi_HOME>/conf/TopazInfra.ini	read	r
<OMi_HOME>/log/opr-clis.log	full control	rw
<OMi_HOME>/log/opr-pgctl.log	full control	rw
Note: This file is not available on gateway server systems.		
<OMi_HOME>/conf/encryption.properties	read	r
<OMi_HOME>/conf/seed.properties	read	r

Examples

- **Disable a custom action script:**

```
opr-script -user myU -password myPwd -disable -type CUSTOM_ACTION -id cebc429e-ab51-ae70-b070-454a092cbdc6
```

- **Enable an EPI script:**

```
opr-script -user myU -password myPwd -enable -type EPI -id 603bb351-b346-ac2e-f7e2-454e4fc1d1ec
```

Part X: Service Health

This section describes how to use the Rules API to create new business rules. These define how Key Performance Indicators (KPIs) are calculated.

This section is structured as follows:

- ["Service Health Rules API" on page 373](#)
- ["Service Health External APIs" on page 397](#)

Chapter 40: Service Health Rules API

Note: In OMi versions 9.00 and later, the rules that calculate indicator statuses and values based on samples ("[API Sample Rule](#)" on page 376 and "[API Duration-Based Sample Rule](#)" on page 377) are used to calculate metric-based health indicators (HIs).

Throughout the Rules API documentation, you will see references to various methods used to calculate KPIs. **In OMi versions 9.00 and later, when calculating sample-based values, these methods are used to calculate metric-based HIs.**

This chapter describes how to use the Rules API to create new business rules. Business rules are used to calculate Key Performance Indicators (KPIs). A KPI must have an associated business rule that defines how the KPI is calculated. For details about the default Service Health rules, see the *OMi Administration Guide*.

It is recommended to create rules with the Rules API. The Rules API enables you to create rules using the Groovy scripting language with Groovy runtime environment. Users of the Rules API should be familiar with Groovy and Java, and with OMi administration and applications. For more information about developing and deploying Groovy scripts, see "[Groovy Scripts](#)" on page 335.

The Rules API classes are documented in Javadoc format in the *OMi Rules API Reference*. These files are located in the following folder:

```
<OMi_HOME_GW>/AppServer/webapps/site.war/amdocs/eng/API_docs/Rules_API/index.html
```

Service Health API Rules

These are the types of Service Health API rules:

- **Group and Sibling Rule.** This rule calculates KPIs based on data received from other KPIs, rather than from original sample data. For details, see "[API Group and Sibling Rule](#)" on the next page.
- **Sample Rule.** This rule calculates KPIs based on original data taken from sample fields; the number of samples included in the calculation is limited by a maximum number of samples rule parameter. For details, see "[API Sample Rule](#)" on page 376.
- **Duration-Based Sample Rule.** This rule calculates KPIs based on original data taken from sample fields; a duration parameter defines which samples are included in the calculation. For details, see "[API Duration-Based Sample Rule](#)" on page 377.

Creating API Rules

Rules can be created using the Rules API in these ways:

- Using the KPI and Health Indicator Customizations per CI page to create a rule for a specific KPI.
- Using a text file to create a new rule for multiple KPIs.
- Using a clone of an API rule in the Rule Repository to create a new rule.

These ways are described in "[Creating Rules with the Rules API](#)" on page 378.

Tooltips and Log Files

To display KPI information in tooltips when working with the Rules API, see ["How to Work with Tooltip Entries" on page 384](#).

You can write to log files from the Rules API code, as described in ["How to Write to Log Files From the Rules API Code" on page 385](#).

API Group and Sibling Rule

An API Group and Sibling Rule calculates KPIs based on data received from other indicators, rather than from original sample data. The received data can come from the KPIs of child CIs, or from other KPIs or HIs associated with the same CI.

Note: If you are creating a sibling rule, make sure that the KPI is calculated after its sibling KPIs, as defined by the KPI's Calculation Order field. For details, see the *OMi Administration Guide*.

Group and Sibling Rule Methods and Fields

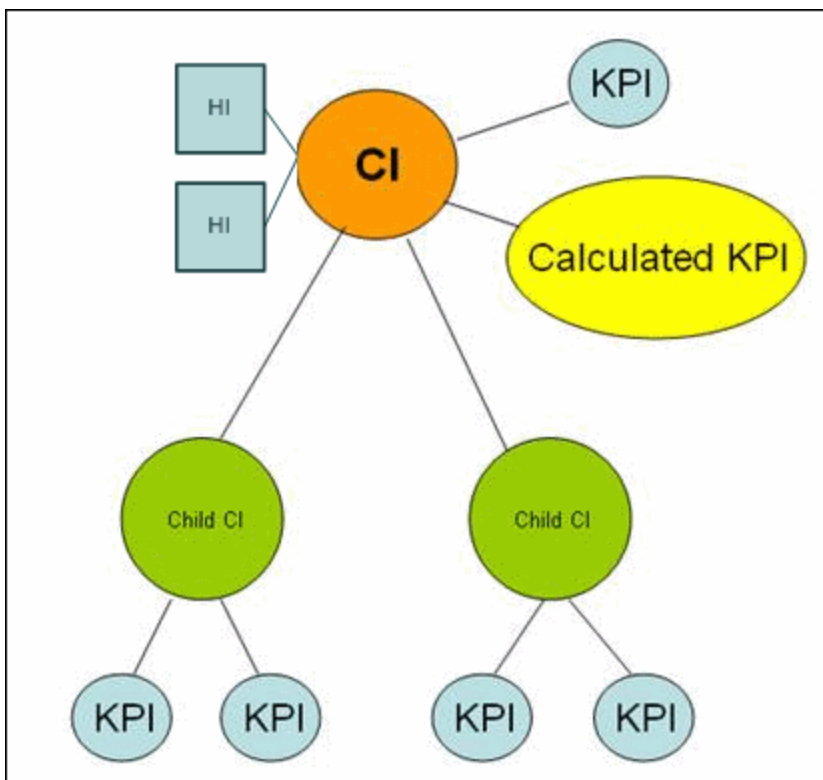
The Group and Sibling rule implements the Rules API Interface **GroupAndSiblingCalculator**, using the following guidelines:

- In this interface, the only method is **calculateKPI**. The method signature is:

```
public void calculateKPI(CI ci, KPI kpi)
```
- The **calculateKPI** method includes the parameters **ci** and **kpi**, which represent the current CI, and the KPI whose value the API rule calculates.
 - The **ci** parameter type is **CI**, and is used as an accessor to KPIs of child CIs or sibling KPIs, or HIs on the CI.
 - The **kpi** parameter type is **KPI**, and is used to set calculation results.

In the following illustration, the Calculated KPI is calculated based on the sibling or child KPIs, and it is represented by the **kpi** parameter.

The CI to which the Calculated KPI is assigned, is represented by the **ci** parameter, and it is an accessor to the other KPIs or HIs.



The Rules API classes are documented in Javadoc format in the *OMi Rules API Reference*. These files are located in the following folder:

`<OMi_HOME_GW>/AppServer/webapps/site.war/amdocs/eng/API_docs/Rules_API/index.html`

For detailed examples of Group and Sibling rules, see ["Examples - API Group and Sibling Rule" on page 389](#).

API rules can be defined within the Service Health KPI and Health Indicator Customizations per CI page or Rule Repository, or using a text file template, as described in ["Creating Rules with the Rules API" on page 378](#).

Defining a Group and Sibling Rule in the KPI and Health Indicator Customizations per CI page or Rule Repository

To define a Group and Sibling rule using the KPI and Health Indicator Customizations per CI page or within the Rule Repository, enter the **calculateKPI** method implementation in the **KPI Calculation Script** area.

The parameters **ci** and **kpi** of the **calculateKPI** method are available for use in this script.

For detailed instructions, see ["How to Define an API Rule in the KPI and Health Indicator Customizations per CI Page" on page 379](#) or ["How to Define an API Rule in the Rule Repository" on page 383](#).

Accessing a Specific Child KPI in the KPI and Health Indicator Customizations per CI Page

When creating a Group rule for a specific KPI in the KPI and Health Indicator Customizations per CI page, to access a specific child KPI, the API includes a mechanism to simplify the code. When defining your KPI Calculation Script, you can enter the format "**<CI name>".<KPI name>**".

For an example of this, see ["Example - Specific Child CI Group Rule" on page 392](#) in ["Examples - API Group and Sibling Rule" on page 389](#).

Defining a Group and Sibling Rule Using a Text File

To define a Group and Sibling rule using a text file, use the **DashboardGroupAndSiblingTemplate.groovy** template as described in ["How to Create a Text File-Based API Rule" on page 380](#).

Within the text file, enter the **calculateKPI** method body.

API Sample Rule

A Sample rule calculates KPIs based on original data taken from sample fields; the number of samples included in the calculation is limited by a maximum number of samples parameter.

Sample Rule Methods and Fields

The Sample rule implements the Rules API Interface **LeafCalculator**, using the following guidelines:

- In this interface, the only method is **calculateKPI**. The method signature is:

```
public void calculateKPI(CI ci, KPI kpi, List<Sample> samples)
```
- The **calculateKPI** method includes the parameters **ci**, **kpi**, and **samples**. These represent the current CI, the KPI whose value the rule calculates, and the samples to be used in the rule calculation based on the **Maximum number of samples** parameter. (If this parameter value is 1, list one sample in this field.)
 - The **kpi** parameter type is **KPI**, and is used to set calculation results.
 - The **samples** parameter is a **List of Sample** objects, which hold sample field values.
- The rule must also set the **sampleFields** field to define which sample fields are held by the **Sample** object. These values are the values used by the rule.

For detailed examples of Sample rules, see ["Examples - API Sample Rule" on page 386](#).

API rules can be defined within the Service Health KPI and Health Indicator Customizations per CI page or the Rule Repository, or using a text file template, as described in ["Creating Rules with the Rules API" on page 378](#).

The Rules API classes are documented in Javadoc format in the *OMi Rules API Reference*. These files are located in the following folder:

```
<OMi_HOME_GW>/AppServer/webapps/site.war/amdocs/eng/API_docs/Rules_API/index.html
```


Defining a Sample Rule in the KPI and Health Indicator Customizations per CI page or Rule Repository

To define a Sample rule using the KPI and Health Indicator Customizations per CI page or within the Rule Repository, fill in the fields as follows:

- **Sample Fields.** List the sample fields which are held by the **Sample** object; separate between the sample names with a comma (for example: "u_iStatus", "dResponseTime").
- **KPI Calculation Script.** Enter the **calculateKPI** method implementation; do not enter the method signature. The parameters **ci**, **kpi**, and **samples** of the **calculateKPI** method are available for use in this script.
- **Maximum number of samples.** By default only the most recent sample is included (default=1). You can use this field to change this setting.

For detailed instructions, see ["How to Define an API Rule in the KPI and Health Indicator Customizations per CI Page" on page 379](#) or ["How to Define an API Rule in the Rule Repository" on page 383](#).

Defining a Sample Rule Using a Text File

To define a Sample rule using a text file template, use the **DashboardSampleRuleTemplate.groovy** template file as described in ["How to Create a Text File-Based API Rule" on page 380](#).

Within the text file, enter the **calculateKPI** method body, and define the **sampleFields** field.

API Duration-Based Sample Rule

A Duration-Based Sample rule calculates KPIs based on original data taken from sample fields; the duration rule parameter defines which samples are included in the calculation. For example, if duration is defined as fifteen minutes, all samples collected during the last fifteen minutes are included in the calculation.

Duration-Based Sample Rule Methods and Fields

The Duration-Based Sample rule implements the Rules API Interface **LeafCalculator**, using the following guidelines:

- In this interface, the only method is **calculateKPI**. The method signature is:

```
public void calculateKPI(CI ci, KPI kpi, List<Sample> samples)
```
- The **calculateKPI** method includes the parameters **ci**, **kpi**, and **samples**. These represent the current CI, the KPI whose value the rule calculates, and the list of samples to be used in the rule calculation.
 - The **kpi** parameter type is **KPI**, and is used to set calculation results.
 - The **samples** parameter is a **List** of **Sample** objects, which hold sample field values.
- The rule must also set the **sampleFields** field to define which sample fields are held by the **Sample** object. These values are the values used by the rule.

For detailed examples of this rule, see ["Examples - API Sample Rule" on page 386](#).

API rules can be defined using the Service Health KPI and Health Indicator Customizations per CI page, using a text file, or within the Rule Repository, as described in ["Creating Rules with the Rules API" below](#).

The Rules API classes are documented in Javadoc format in the *OMi Rules API Reference*. These files are located in the following folder:

```
<OMi_HOME_GW>/AppServer/webapps/site.war/amdocs/eng/API_docs/Rules_API/index.html
```

Defining a Duration-Based Sample Rule in the KPI and Health Indicator Customizations per CI page or Rule Repository

To define a Duration-Based Sample rule using the KPI and Health Indicator Customizations per CI page or within the Rule Repository, fill in the fields as follows:

- **Sample Fields.** List the sample fields which are held by the **Sample** object; separate between the sample names with a comma (for example: "u_iStatus", "dResponseTime").
- **KPI Calculation Script.** Enter the method implementation; do not enter the method signature. The parameters **ci**, **kpi**, and **samples** of the **calculateKPI** method are available for use in this script.
- **No data timeout** and **duration.** (Optional) You can define the timeout period and duration parameters, as described in [List of Rule Parameters](#).

For detailed instructions, see ["How to Define an API Rule in the KPI and Health Indicator Customizations per CI Page" on the next page](#) or ["How to Define an API Rule in the Rule Repository" on page 383](#).

Defining a Duration-Based Sample Rule Using a Text File

To define a Duration-Based Sample rule using a text file template, use the **DashboardDurationBasedSampleRuleTemplate.groovy** template file as described in ["How to Create a Text File-Based API Rule" on page 380](#).

Within the text file, enter the **calculateKPI** method body, and define the **sampleFields** field.

Creating Rules with the Rules API

There are a number of ways to create rules using the Rules API, as described in the following section.

Define a rule for a specific KPI using the KPI and Health Indicator Customizations per CI page

Each Service Health KPI has these applicable API rules: API Group and Sibling Rule, API Sample Rule, or API Duration-Based Sample Rule. From the KPI and Health Indicator Customizations per CI page, you can assign one of the API rules to a KPI, and enter a calculation script (and other rule details) to define rule logic for that KPI.

You can then edit the rule details in the KPI and Health Indicator Customizations per CI page at any time to change the rule logic for the KPI.

For details, see ["How to Define an API Rule in the KPI and Health Indicator Customizations per CI Page" below](#).

Create a rule using a text file

For each of the API rules (Group and Sibling Rule, Sample Rule, or Duration-Based Sample Rule) there is a corresponding template file, located in the **<Data Processing server root directory>\BLE\rules\groovy\templates** directory. You can use one of the template files to create a text file defining a new rule. You then add this rule to the Rule Repository, and it can be applied like any out-of-the-box rule.

The API code cannot be seen or changed within Service Health, but only within the text file. If you make changes to the code within the text file, these changes are applied to all instances where the rule has been assigned, after you reload Service Health rules.

For details, see ["How to Create a Text File-Based API Rule" on the next page](#).

Define a rule within the Rule Repository

The Rule Repository contains three API rules: API Group and Sibling Rule, API Sample Rule, or API Duration-Based Sample Rule. You can use the Rule Repository to clone an API rule and enter a calculation script (and other rule details) to define the rule logic.

After the rule is applied to a KPI, you can edit rule details within the KPI and Health Indicator Customizations per CI page at any time to change the rule logic for a specific KPI.

For details, see ["How to Define an API Rule in the Rule Repository" on page 383](#).

How to Define an API Rule in the KPI and Health Indicator Customizations per CI Page

Each KPI has three applicable API rules. Within the KPI and Health Indicator Customizations per CI page, assign one of the API rules to a KPI, and enter the calculation script (and other rule details) to define the rule logic for that KPI.

1. Assign an API rule to a KPI

To assign an API rule for a specific KPI assigned to a CI, select **Administration > Service Health > CI Status Calculation > KPI and Health Indicator Customizations per CI**. Select **Add KPI** to assign a new KPI to the CI, or **Edit KPI** to modify an existing KPI. For details on this process, see the *OMi Administration Guide*.

From the list of applicable business rules, select one of the API rules: API Group and Sibling Rule, API Sample Rule, or API Duration-Based Sample Rule. For a description of the rule types, see ["Service Health Rules API" on page 373](#).

2. Define the KPI's rule logic

Depending on the type of rule you are creating, define the rule methods and fields as described in:

- ["API Group and Sibling Rule" on page 374](#)
- ["API Sample Rule" on page 376](#)
- ["API Duration-Based Sample Rule" on page 377](#)

How to Create a Text File-Based API Rule

There are three rule template files corresponding to the three API rules; each template implements the rule's interface.

Create a text file defining a new rule using one of the templates, and then add the new rule to the Business Rule Repository. The rule can then be applied like any out-of-the-box rule.

The API code cannot be seen or changed within Service Health, but only within the text file. If you make changes to the code within the text file, these changes are applied to all instances where the rule has been assigned, after you reload Service Health rules.

1. Create a text file for a rule

Based on the type of rule you want to create, copy and rename one of the template files located in the **<Data Processing server root directory>\BLE\rules\groovy\templates** directory.

Within your copy of the template, define the rule methods and fields as described in:

- ["API Group and Sibling Rule" on page 374](#)
- ["API Sample Rule" on page 376](#)
- ["API Duration-Based Sample Rule" on page 377](#)

Save the file to the **<Data Processing server root directory>\BLE\rules\groovy\rules** directory.

You must now add a rule in the Rule Repository that uses the rule logic in the text file.

2. Add a rule in the rule repository

- Select **Admin > Service Health > Repositories > Business Rules > New Rule**. For details on adding rules, see the *OMi Administration Guide*.
 - In the **Name** field, type the name of the rule you want to create (mandatory).
 - In the **Class Name** field, type **groovy: <file name>**. Note that the file name must be identical (case sensitive) to the file name in the **<Data Processing server root directory>\BLE\rules\groovy\rules** directory.
 - Create Rule parameters depending on your API rule type, as follows:
 - In the **Rule parameters** area, click **New**.
 - For API Sample rules:
 - In the **Name** field type **Maximum number of samples**. In the **Type** field, select **Integer**.
 - In the **Default Value** field, type **1**.
- Click **OK** to Save.

- For API Duration-Based Sample rules:
In the **Name** field type **duration**. In the **Type** field, select **Long**. In the **Default Value** field, type **990**.
Click **OK** to Save.

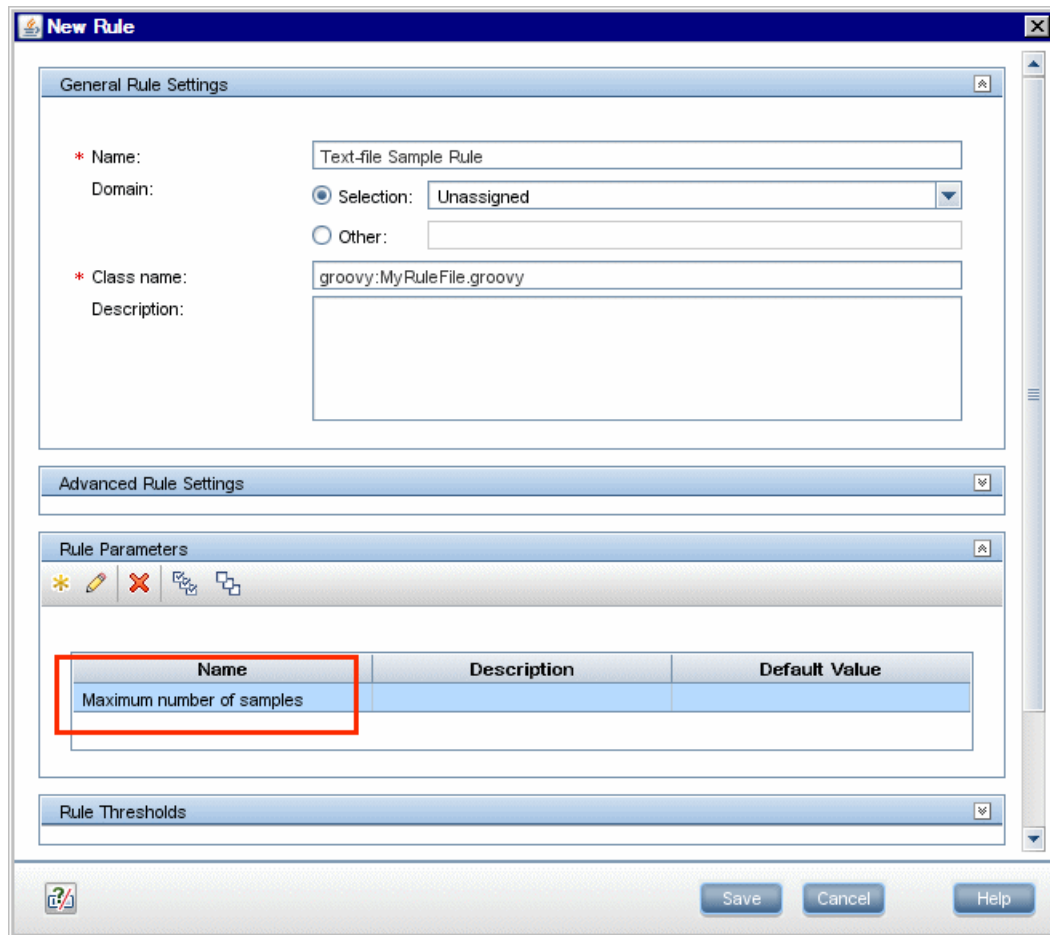
Repeat these steps to add the **No Data Timeout** rule parameter (Type: Long; Default Value = 990).

- e. Create Threshold parameters: critical, major, minor, warning, informational, and operator. (Skip this step if you are defining a Group and Sibling rule that does not have Thresholds, where status is calculated by the rule code.)

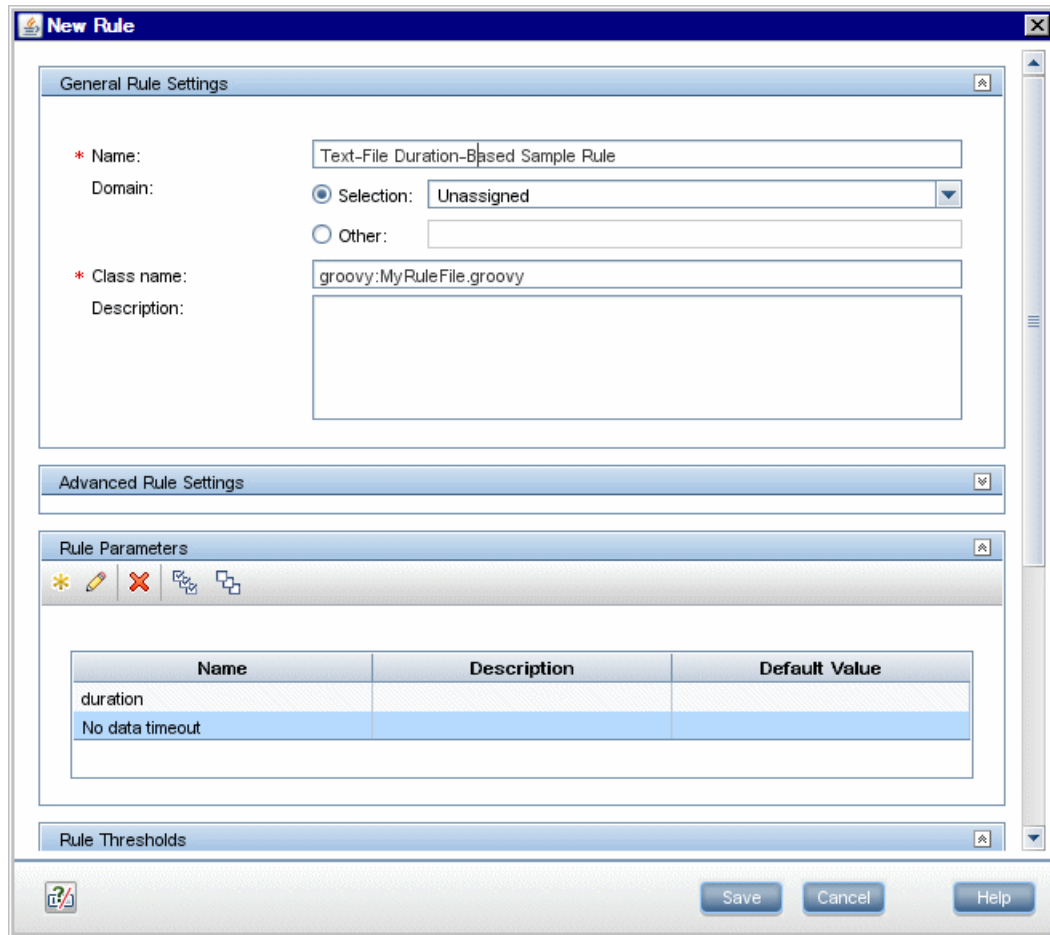
- In the **Threshold parameters** area, click **New**.
- In the **Name** field, type **critical**. In the **Type** field, select **Float**.
When defining the **operator** parameter, select **String** in the **Type** field.
- Click **OK** to save.

Repeat the above steps for each of the other Threshold parameters (major, minor, warning, informational, and operator).

The following image shows a Sample rule after the rule parameter has been added:



The following image shows a Duration-Based Sample rule after the rule parameters have been added:



3. Add the rule to the KPI's applicable rules list

Add the new rule to the list of applicable rules already attached to the relevant KPI. For details, see the Applicable Rules parameter in the *OMi Administration Guide*.

4. Add tooltip parameters to the new tooltip

When a rule is created using this procedure, a corresponding tooltip is with no tooltip parameters. For instructions on adding tooltip parameters to the new tooltip, see ["How to Work with Tooltip Entries" on page 384](#).

5. Reload rules after editing the text file

If you make changes to the text file at any time after the rule is created, perform the following steps to apply the changes.

- a. In the browser, access JMX port <29810 + workerID> (for example, 29811 for worker_1).
- b. Within **OMi-Platform**, select the service called **MarbleWorker** and invoke the **reloadRules** method. This method is applied to all the customers served by this worker.

How to Define an API Rule in the Rule Repository

Within the Business Rule Repository, create an API rule that can be applied to multiple KPIs. This is done by cloning one of the three API rules, and setting default rule values for specific rule parameters. After the rule is applied to a KPI, you can edit its script within the KPI and Health Indicator Customizations per CI page at any time to change the rule logic for the specific KPI.

1. Clone an API rule

Select **Administration > Service Health > CI Status Calculation > Business Rules**. In the Business Rule Repository page, clone one of the following rules: API Group and Sibling Rule, API Sample Rule, or API Duration-Based Sample Rule.

For details on cloning a rule, see the *OMi Administration Guide*.

2. Edit rule details

- a. Open the new rule for editing.
- b. In the **Name** field, rename the cloned rule.
- c. Edit the **KPI Calculation Script** rule parameter. In the **Default Value** field, enter the rule calculation script. The code that you enter is the default code for this rule, and appears in the KPI and Health Indicator Customizations per CI page for all KPIs assigned this rule. (Do not change any other fields.)
- d. If you are creating a Sample rule or Duration-Based Sample rule, edit the **Sample Fields** rule parameter. The sample fields that you enter are the default sample fields for this rule, and appear in the KPI and Health Indicator Customizations per CI page for all KPIs assigned this rule. (Do not change any other fields.)

For details on these rule parameters, see the following sections (depending on the type of rule you are creating):

- ["API Group and Sibling Rule" on page 374](#)
- ["API Sample Rule" on page 376](#)
- ["API Duration-Based Sample Rule" on page 377](#)

The Rules API classes are documented in Javadoc format in the *OMi Rules API Reference*. These files are located in the following folder:

```
<OMi_HOME_GW>/AppServer/webapps/site.war/amdocs/eng/API_docs/Rules_  
API/index.html
```

3. Add the rule to the KPI's applicable rules list

Add the new rule to the list of applicable rules already attached to the relevant KPI. For details, see the Applicable Rules parameter in the *OMi Administration Guide*.

How to Work with Tooltip Entries

The following section describes how to work with tooltip entries to display information calculated by the Rules API.

1. Go to:

Administration > Service Health > CI Status Calculation > Business Rules

In the Rule Repository page, add any required tooltip entries for the new rule. The following table lists common tooltip entries and their corresponding value sources and formatting methods:

Tooltip Parameter	Value Source	Formatting Method
Business Rule	NODE.DIM.RULE.ID_CUST	ruleIDtoString
CI name	NODE.PROPS.BamNodeNameKey	toLowerCase
Last Status Change	NODE.DIM.RESULT.LastStatusChange	returnDateAsString
Status	NODE.DIM.RESULT.Status	getStatusString
Value	NODE.DIM.RESULT.Value	returnNumOfDigitAfterPoint

For details, see the *OMi Administration Guide*.

2. If you have used the `kpi.setTooltip` method, you must set a corresponding tooltip entry in the Rule Repository as described above. In the **Value Source** field, type the name of the tooltip entry exactly as used in the code, and leave the **Formatting Method** field empty.

For example, if your code contains the method invocation `kpi.setTooltip("total_sales", value)`, type `NODE.DIM.RESULT.total_sales` in the **Value Source** field.

How to Write to Log Files From the Rules API Code

Within your API rules, you can write to log files from rule methods using a **logger** object. There are five log levels: debug, info, warn, error and fatal. Each of these uses a specific logger method.

By default, only log method invocations of error and fatal severity are written to the log files. You can modify this within the log configuration files.

To write to log files using the Rules API:

1. Within the rule method, implement one of the following methods (listed in ascending order of severity):
 - **logger.debug("<API rule name> : log message");**
 - **logger.info("<API rule name> : log message");**
 - **logger.warn("<API rule name> : log message");**
 - **logger.error("<API rule name> : log message");**
 - **logger.fatal("<API rule name> : log message");**

Type the name of your API rule inside the log message to identify each log message with its source rule.

2. The Rules API log files are found in the `<OMi_HOME_DPS>/log/marble_worker_<worker#>/RulesAPI` directory.

Open one of the following files to view the log messages (depending on your rule type):

- **groupAndSiblingRule.log** (for API Group and Sibling rules)
- **sampleRule.log** (for API Sample rules)
- **durationBasedSampleRule.log** (for API Duration-Based sample rules)

To modify the severity level written to a log file:

1. By default, only log method invocations of error and fatal severity are written to log files. To modify this setting, open the log configuration file located in `<OMi_HOME_DPS>/conf/core/Tools/log4j/marble_worker/dashboard_rules.properties`.
2. In the line corresponding with your rule type, replace the string **\${loglevel}** with the severity level you want logged (either DEBUG, INFO, WARN, ERROR, or FATAL). Edit one of the following lines, depending on your rule type:
 - Group and Sibling rules: `log4j.category.com.mercury.am.rules.dashboard.biDashboardRules.simplifiedRule.groupAndSiblingRule.DashboardGroupAndSiblingRule = ${loglevel}, bam.app.rules.api.group.appender`
 - Sample rules: `log4j.category.com.mercury.am.rules.dashboard.biDashboardRules.simplifiedRule.leaf.DashboardSimplifiedSampleBasedRule = ${loglevel},`

```
bam.app.rules.api.leafsample.append
```

- Duration-Based Sample rules:
log4j.category.com.mercury.am.rules.dashboard.blDashboardRules.
simplifiedRule.leaf.DashboardSimplifiedTimeBasedRule = **`\${loglevel}**},
bam.app.rules.api.leafduration.append

How to Include a CI Property in Rules API Calculations

Within your API rules, you can include CI properties using the CI class **getPropertyValue** method, and the KPI class **getCiProperty** method. Only CI properties with one of the following qualifiers can be accessed with this method:

- BLE_ATTRIBUTE
- BLE_ONLINE_ATTRIBUTE

To add this attribute to a CI class you must export the class, edit the class definition, and import it back to the server. When you open the exported class for editing, add the following xml to the required attribute:

```
<Attribute name="<attribute-name>" type="double" display-name="<attribute-  
display-name">  
  <Attribute-Qualifiers>  
    <Attribute-Qualifier name="BLE_ATTRIBUTE"/>  
  </Attribute-Qualifiers>  
</Attribute>
```

To retrieve the CI property value of a CI using the API Group and Sibling Rule, you must restart **marble_dashboard_tql**:

1. Access the RTSM JMX Console – `<OMi_HOME_DPS>:21212/jmx-console/HtmlAdaptor?action=inspectMBean&name=UCMDB:service=TQL%20Services`.
2. Invoke **retrieveTqlNames()**.
3. Search for **marble_dashboard_tql**, and restart the TQL.

Examples - API Sample Rule

This section provides examples of API Sample Rules. The following examples are described:

- ["Example - Average Availability Rule" below](#)
- ["Example - Average Performance Rule" on the next page](#)
- ["Example - Average Performance Rule Using a Rule Parameter Filter" on page 388](#)

Example - Average Availability Rule

The following rule calculates average availability of samples, based on the `u_iStatus` sample field.

The rule logic is (available samples / total samples) * 100.

```
// This rule uses the u_iStatus sample field.
def sampleFields = ["u_iStatus"];
public void calculateKPI(CI ci, KPI kpi, List<Sample> samples) {
    // Keep total number of samples for this calculation cycle.
    def totalSamples = samples.size();
    // Create a variable to count available samples.
    def availableSamples = 0;
    /**
     * Go over the given samples. If a sample's u_iStatus is equal to 0,
     * the sample is considered available.
     */
    samples.each {Sample currentSample->
        if (currentSample.u_iStatus == 0) {
            // Increase the count of available samples.
            availableSamples++;
        }
    }
    if (totalSamples > 0) {
        // Set KPI value, converted to percentage.
        kpi.setValue ((availableSamples/totalSamples)*100.0);
    }
}
```

Example - Average Performance Rule

The following rule calculates average performance in seconds, based on the dResponseTime and u_iStatus sample fields.

Only samples with a u_iStatus value of 0 (available samples) are used in the calculation. The rule logic is: $\text{sum}(\text{dResponseTime}) / \text{available samples}$.

```
// This rule uses the u_iStatus and dResponseTime sample field.
def sampleFields = ["u_iStatus", "dResponseTime"];
public void calculateKPI(CI ci, KPI kpi, List<Sample> samples) {
    // Create a variable to count available samples.
    def availableSamples = 0;
    // Create a variable to sum response times of available samples.
    def totalResponseTime = 0;
    /**
     * Go over the given samples. If a sample's u_iStatus is equal to 0,
     * the sample is considered available.
     */
    samples.each {Sample currentSample ->
        if (currentSample.u_iStatus == 0) {
            // Increase the count of available samples.
            availableSamples++;
        }
    }
}
```

```
        // Add the current sample's dResponseTime value to
totalResponseTime.
        totalResponseTime += currentSample.dResponseTime
    }
}
if (availableSamples > 0) {
    // Set KPI value, converted to percentage.
    kpi.setValue((totalResponseTime / availableSamples))
}
}
```

Example - Average Performance Rule Using a Rule Parameter Filter

The following rule calculates average performance in seconds, based on the `dResponseTime` and `u_iStatus` sample fields.

Only samples with a `u_iStatus` value of 0 (available samples) are used in the calculation.

The rule uses an optional rule parameter: Response time limit. If this rule parameter value has been set in the Service Health Admin, samples with a `dResponseTime` value greater than the rule parameter value are not used in the calculation.

Note: A rule parameter with the same name must be set for the rule in the Rule Repository. For details, see the *OMi Administration Guide*.

The rule logic is: `sum(dResponseTime) / available samples`.

```
/ This rule use the u_iStatus and dResponseTime sample fields.
def sampleFields = ["u_iStatus", "dResponseTime"];
public void calculateKPI(CI ci, KPI kpi, List<Sample> samples) {
    // Create a variable to count available samples.
    def availableSamples = 0;
    // Create a variable to sum response times of available samples.
    def totalResponseTime = 0;
    /**
     * Get the value of the rule parameter named "Response time limit"
     * from the KPI, as defined for the KPI in Service Health Admin.
     * This rule parameter is optional, so responseTimeLimit can be null.
     */
    Long responseTimeLimit = kpi.getRuleParameter("Response time limit")
    /**
     * Go over the given samples. If a sample's u_iStatus is equal to 0,
     * the sample is considered available.
     */
}
```

```
samples.each {Sample currentSample ->
  if (currentSample.u_iStatus == 0) {
    /**
     * Check the value of the rule parameter.
     * If it is not null (meaning the user has set a value),
     * and the sample's dResponseTime is greater than the
     * rule parameter value, the value is not valid.
     */
    boolean isSampleValid = true;
    if (responseTimeLimit != null) {
      // Check if ResponseTime exceeds the rule parameter value.
      if (currentSample.dResponseTime > responseTimeLimit) {
        // The sample is not valid.
        isSampleValid = false;
      }
    }
    if (isSampleValid) {
      // Increase the count of available samples.
      availableSamples++;
      // Add the sample's dResponseTime value to totalResponseTime.
      totalResponseTime += currentSample.dResponseTime
    }
  }
}
if (availableSamples > 0) {
  // Set KPI value, converted to percentage.
  kpi.setValue((totalResponseTime / availableSamples))
}
}
```

Examples - API Group and Sibling Rule

This section provides examples of API Group and Sibling Rules. The following examples are described:

- ["Example - Worst Child Rule" on the next page](#)
- ["Example - Worst Sibling Status Rule" on page 391](#)
- ["Example - Specific Child CI Group Rule" on page 392](#)
- ["Example - Sibling Rule Based on Availability and Performance KPIs" on page 392](#)
- ["Example - Group Average Value by CI Type" on page 393](#)
- ["Example - Worst Health Indicator Rule" on page 394](#)
- ["Example - Using Groovy Closure" on page 395](#)

Example - Worst Child Rule

The following rule finds the worst status from all of the KPIs of the calculated CI's child CIs, which are of the same type as the calculated KPI, based on active statuses only. Active statuses are **Critical**, **Major**, **Minor**, **Warning**, and **OK**.

```
public void calculateKPI(CI ci, KPI kpi) {
    // Get the calculated KPI's type ID (as defined in the Service Health KPI
    Repository).
    int kpiId = kpi.getType();
    // Get a list of all of the KPIs of the calculated CI's child CIs, which are
    of the same
    // type as the calculated KPI.
    List<KPI> childKpiList = ci.getChildrenKPIsByID(kpiId);
    // Create a variable to set the status of the calculated KPI,
    // only if an active status is found.
    boolean isActiveStatusFound = false;
    // Set the current worst status to OK; if a worse status is found this will
    be updated.
    Status worstStatus = Status.OK;
    // Go over the list of child KPIs.
    childKpiList.each{KPI childKPI->
        // Get the child KPI's status.
        Status childKpiStatus = childKPI.status;
        // Check if the child KPI's status is an active status.
        if(childKpiStatus.isActive()){
            // Mark that an active status was found.
            isActiveStatusFound = true;
            // Check if the child KPI's status is worse than the current worst
            status.
            if(childKpiStatus.isWorse(worstStatus)){
                // Update the worst status.
                worstStatus = childKpiStatus;
            }
        }
    }
    // Check if an active status was found in the child KPI.
    if(isActiveStatusFound){
        // Set the calculated KPI status.
        kpi.setStatus(worstStatus);
    }
}
```

Example - Worst Sibling Status Rule

The following rule finds the worst status from sibling KPIs, based on active statuses only. Active statuses are **Critical**, **Major**, **Minor**, **Warning**, and **OK**.

```
public void calculateKPI(CI ci, KPI kpi) {
    // Get a list of all the KPIs for the CI.
    List<KPI> ciKpiList = ci.getAllKPIs();
    /**
     * Create a variable to set the status of the calculated KPI,
     * only if an active status is found.
     */
    boolean isActiveStatusFound = false;
    // Set the current worst status to OK; if a worse status is found this will
    be updated.
    Status worstStatus = Status.OK;
    // Go over the list of the CI's KPIs.
    ciKpiList.each {KPI ciKPI ->
        /**
         * Check that the CI's KPI is not the calculated KPI.
         * This is needed because getAllKPIs method returns all the KPIs for the
        CI.
         */
        if (ciKPI != kpi) {
            /**
             * The ciKPI represents a sibling KPI of the calculated KPI.
             * Get the sibling KPI's status.
             */
            Status siblingKpiStatus = ciKPI.status;
            // Update worstStatus if necessary.
            if (siblingKpiStatus.isActive()) {
                isActiveStatusFound = true;
                if (siblingKpiStatus.isWorse(worstStatus)) {
                    worstStatus = siblingKpiStatus;
                }
            }
        }
    }
    // Check if an active status was found in the sibling KPI.
    if (isActiveStatusFound) {
        // Set the calculated KPI's status.
        kpi.setStatus(worstStatus);
    }
}
```

Example - Specific Child CI Group Rule

The following rule calculates KPI status based on the Availability KPI of a specific child CI (RTSM ID = "96c2df2b544683c7f79bb382d1d7b3a9").

If the child CI's Availability KPI value is 100, the calculated KPI's status is set to OK. All other values set the KPI's status to **Critical**.

Status is set only if the child CI exists, has the Availability KPI, and its Availability KPI has value.

```
public void calculateKPI(CI ci, KPI kpi) {
    /**
     * Get the Availability KPI for the child CI "tx_10 from virtual_host_3".
     * The RTSM ID of "tx_10 from virtual_host_3" is
     * "96c2df2b544683c7f79bb382d1d7b3a9".
     *
     * Note: Within the UI, the following line can be written as
     * KPI childKPI = "tx_10 from virtual_host_3"."Availability"
     */
    KPI childKPI = ci.getChildKpiByChildId(KpiType.Availability,
        "96c2df2b544683c7f79bb382d1d7b3a9");

    // Check if childKPI is not null. It is null if no child CI with this RTSM
    ID exists, or if this CI does not have the Availability KPI.
    if (childKPI != null) {

        // Check if the child KPI has a value.
        if (childKPI.valueExist) {
            if (childKPI.value == 100.0) {
                kpi.status = Status.OK
            }
            else {
                kpi.status = Status.CRITICAL
            }
        }
    }
}
```

Example - Sibling Rule Based on Availability and Performance KPIs

The following rule calculates KPI status based on the status of sibling Availability and Performance KPIs.

If these KPIs do not exist or do not have active status, no status is set.

If these sibling KPIs exist and are both OK, the calculated KPI status is set to OK. Otherwise, its status is set to **Critical**. (Active statuses are **Critical**, **Major**, **Minor**, **Warning**, and **OK**.)

```
public void calculateKPI(CI ci, KPI kpi) {
    /**
     * Get the sibling KPI of type Availability.
     * If Availability KPI does not exist, null will be returned.
     */
    KPI availabilityKPI = ci.getKPI(KpiType.Availability);
    // Get the sibling KPI of type Performance.
    KPI performanceKPI = ci.getKPI(KpiType.Performance);
    if (availabilityKPI != null && performanceKPI != null) {
        // Both KPIs exist for this CI. Check if the KPIs status is active.
        if (availabilityKPI.status.isActive() && performanceKPI.status.isActive
        ()) {
            // Check the KPI's status.
            if (availabilityKPI.status == Status.OK &&
                performanceKPI.status == Status.OK) {
                /**
                 * Both statuses are active and both are OK. Set this KPI's
                status to OK.
                 */
                kpi.status = Status.OK
            }
            else {
                /**
                 * Both statuses are active, and not both are OK.
                 * Set this KPI's status to CRITICAL
                 */
                kpi.status = Status.CRITICAL
            }
        }
    }
}
```

Example - Group Average Value by CI Type

The following rule calculates the average status of the KPIs of child CIs, which are of the same CI type as the calculated KPI.

Only child CIs of type "unix" are used in the calculation. If there are no child CIs of this type, or no child CI KPIs have value, no value is set for the KPI.

```
public void calculateKPI(CI ci, KPI kpi) {
    // Get the calculated KPI's type ID (as defined in the Service Health KPI
    Repository).
    int kpiId = kpi.getType();
```

```
// Get a list of the KPIs of the child CIs, which are of the same CI type as
the calculated
// KPI, whose CI type is "unix".
List<KPI> unixChildKpiList = ci.getChildrenKPIsByIDAndCiType(kpiId, "unix")
// Create a variable to sum the total values from child KPIs.
// If no child exists or no child has value the variable will remain null.
Double totalChildValue = null;
// Write information to the log file.
logger.debug("DashboardGroupAvgValueByCiTypeRule : number of child CIs with
type unix: " + unixChildKpiList.size())
// Go over the list of child KPIs.
unixChildKpiList.each {KPI childKPI ->
    // Sum values of the child KPIs using the Utils class, which handles
null values.
    totalChildValue = Utils.sum(totalChildValue, childKPI.value);
}
// Set the calculated KPI's value to the average value, using the Utils
class.
// If totalChildValue is null, null value will be set.
kpi.value = Utils.divide(totalChildValue, unixChildKpiList.size());
}
```

Example - Worst Health Indicator Rule

The following rule finds the worst status from all of the health indicators (HIs) of the calculated CI, based on active statuses only. Active statuses are **Critical**, **Major**, **Minor**, **Warning**, and **OK**.

```
public void calculateKPI(CI ci, KPI kpi) {
    // Get all health indicators.
    List<HI> his = ci.getHIs();
    // Create a variable to set the status of the calculated KPI,
    // only if an active status is found.
    boolean isActiveStatusFound = false;
    // Set the current worst status to OK;
    // if a worse status is found this will be updated.
    Status worstHiStatus = Status.OK;
    his.each {HI hi ->
        Status hiStatus = hi.getStatus();
        // Check if the current HI status is an active status.
        if (hiStatus.isActive()) {
            // Mark that an active status was found.
            isActiveStatusFound = true;
            // Check if the child KPI's status is worse than the current worst
status.
            if (hiStatus.isWorse(worstHiStatus)) {
```

```
        // Update the worst status.
        worstHiStatus = hiStatus;
    }
}
}
// Check if an active status was found in the child KPI.
if (isActiveStatusFound) {
    // Set the calculated KPI status.
    kpi.setStatus(worstHiStatus);
}
}
```

Example - Using Groovy Closure

The following rule sets the calculated KPI's status to Critical, if at least one Availability KPI with Major status exists for the calculated CI's child CIs.

This rule illustrates Groovy Closure. Refer to <http://groovy.codehaus.org/Closures> for more information.

```
public void calculateKPI(CI ci, KPI kpi) {
    /**
     * Use Groovy Closure with the CI class getChildrenKPIs method,
     * to get List of KPIs from the CI's child CIs, where
     * 1. KPI type is Availability
     * 2. Status is MAJOR
     *
     * Closure description:
     * { KPI childKPI ->
     *     childKPI.type == KpiType.Availability.getID("DASHBOARD") &&
     *     childKPI.status == Status.MAJOR
     * }
     * The Closure defines one parameter named childKPI of type KPI.
     * Each KPI from the CI's child CIs will be passed to the Closure by the
     * getChildrenKPIs method.
     * The Closure body returns a boolean value based on the logical expression
     * result.
     * Each KPI that the Closure body will return true for, will be part of the
     * returned List
     * The expression KpiType.Availability.getID("DASHBOARD") returns an int
     * representing the Availability KPI ID from the Service Health KPI Repository.
     */
    List<KPI> kpiList = ci.getChildrenKPIs {KPI childKPI ->
        childKPI.type == KpiType.Availability.getID("DASHBOARD") &&
        childKPI.status == Status.MAJOR
    }
```

```
    }  
    // Check if such a KPI exists.  
    if (kpiList.isEmpty()) {  
        // No such KPI exists.  
        // Write to a log file at debug level.  
        logger.debug "Closure Rule: no Availability KPI with MAJOR status  
exist"  
    }  
    else {  
        // At least one Availability KPI with MAJOR status exists.  
        logger.debug("Closure Rule: At least one Availability KPI with MAJOR  
status exist")  
        // Set calculated KPI status to CRITICAL.  
        kpi.status = Status.CRITICAL;  
    }  
}
```

Chapter 41: Service Health External APIs

This section includes:

- ["Retrieve Indicator Data API" below](#). You can use this API to access KPI over time statuses, KPI definitions, and indicator statuses.
- ["Reset Health Indicator State API" on page 402](#). In certain event flows, you might have an HI showing that a problem has occurred but no event has closed the problem, even though the problem was fixed. After dealing with the problem, you can use this API to reset the HI's state to **Normal**.
- ["Service Health Database Query API" on page 403](#). You can use this API to query the database and return a list of views in XML format.

Retrieve Indicator Data API

The following external API can be used to access KPI over time statuses, KPI definitions, and indicator statuses.

This section includes:

- ["Get KPI Over Time Statuses" below](#)
- ["Get KPI Definitions" on page 399](#)
- ["Get Indicator Statuses" on page 400](#)

The service log file is located under: `<OMi_HOME_GW>/log/jboss/serviceHealthExternalAPI.log`.

Return values are supported in XML and JSON formats.

Authentication should be done using basic access authentication method. For details and examples refer to http://en.wikipedia.org/wiki/Basic_access_authentication.

Get KPI Over Time Statuses

You can use the following to get KPI over time statuses.

API Syntax

```
https://<Gateway Server>/topaz/servicehealth/customers/<Customer Id>/kpiOverTime?ciIds=<CI ID>&startDate=<Start Date>&endDate=<End Date>
```

The API uses the following parameters:

- **customerId**. Customer ID (use **1** for non-HPE SaaS deployment).
- **ciId**. Mandatory; use comma-separated CI IDs.
- **startDate**. Mandatory; start time for the KPI status (value representing the date in milliseconds since January 1 1970).
- **endDate**. Mandatory; end time for the KPI status (value representing the date in milliseconds since January 1 1970).

- **view.** Optional; retrieve the results in the context of a local impact view (default is global view).
- **kpild.** Optional; use comma separated KPI internal IDs as in the repository UI (default is empty for all KPIs). For details, see the *OMi Administration Guide*.

The following is an example of the API and its output:

```
https://omi.example.com/topaz/servicehealth/customers/1/kpiOverTime?  
ciIds=0b656ce308022a6739e3e726497fda6a&startDate=1296499370000  
&endDate=1296501466000
```

```
<kpiStatuses>  
  <kpiStatus>  
    <ciId>0b656ce308022a6739e3e726497fda6a</entityId>  
    <ciDisplayLabel>ATM 1610</ciDisplayLabel>  
    <kpiType>6</kpiType>  
    <kpiDisplayName>Application Performance</kpiDisplayName>  
    <timeStamp>1296499370000</timeStamp>  
    <status>20</status>  
    <statusDisplayName>OK</statusDisplayName>  
    <duration>311000</duration>  
  </kpiStatus>  
  <kpiStatus>  
    <ciId>0b656ce308022a6739e3e726497fda6a</entityId>  
    <ciDisplayLabel>ATM 1610</ciDisplayLabel>  
    <kpiType>6</kpiType>  
    <kpiDisplayName>Application Performance</kpiDisplayName>  
    <timeStamp>1296499681000</timeStamp>  
    <status>-2</status>  
    <statusDisplayName>No Data</statusDisplayName>  
    <duration>1785000</duration>  
  </kpiStatus>  
  <kpiStatus>  
    <ciId>0b656ce308022a6739e3e726497fda6a</entityId>  
    <ciDisplayLabel>ATM 1610</ciDisplayLabel>  
    <kpiType>6</kpiType>  
    <kpiDisplayName>Application Performance</kpiDisplayName>  
    <timeStamp>1296501466000</timeStamp>  
    <status>20</status>  
    <statusDisplayName>OK</statusDisplayName>  
    <duration>13334000</duration>  
  </kpiStatus>  
  <kpiStatus>  
    <ciId>0b656ce308022a6739e3e726497fda6a</entityId>  
    <ciDisplayLabel>ATM 1610</ciDisplayLabel>  
    <kpiType>7</kpiType>  
    <kpiDisplayName>Application Availability</kpiDisplayName>  
    <timeStamp>1296428400000</timeStamp>  
    <status>0</status>
```

```
<statusDisplayName>Critical</statusDisplayName>  
<duration>69663000</duration>  
</kpiStatus>  
</kpiStatuses>
```

The output fields are as follows:

Field	Description
cild	CI ID
ciDisplayLabel	CI display label
kpiType	KPI ID (see "Get KPI Definitions" below below)
kpiDisplayName	KPI display name
timeStamp	Start time for the KPI status; value representing the date in milliseconds since January 1 1970
status	KPI status (see Get Indicator Statuses below)
statusDisplayName	KPI status display name
duration	Duration of the KPI's status in milliseconds.

Return Codes

The API returns the following return codes:

Name	Error Code	Description
BAD_REQUEST	400	<ul style="list-style-type: none">Start date is after the end dateStart date is in the futurestartDate, endDate or cilds are missing
UNAUTHORIZED	401	User has no permission for the selected view
INTERNAL_SERVER_ERROR	500	<ul style="list-style-type: none">Result size has exceeded the maximum quotaGeneral failure

Get KPI Definitions

You can use the following to retrieve the KPIs defined in the system.

API Syntax

```
https://<Gateway Server>/topaz/servicehealth/customers/<CustomerId>/  
repositories/indicators/kpis/<kpiId>
```

The API uses the following parameters:

- **customerId**. Customer ID (use **1** for non-HPE SaaS deployment).
- **kpiIds**. Optional; leave empty for all KPIs (default), or enter a KPI internal ID as in the repository UI, to select a specific KPI. For details, see the *OMi Administration Guide*.

The following is an example of the API and its output:

```
https://host.devlab.ad/topaz/servicehealth/customers/1/repositories/  
indicators/kpis/
```

```
<kpis>  
  <kpi>  
    <id>1</id>  
    <name>Legacy System</name>  
  </kpi>  
  <kpi>  
    <id>1311</id>  
    <name>Value</name>  
  </kpi>  
  <kpi>  
    <id>1310</id>  
    <name>Exceptions</name>  
  </kpi>  
</kpis>
```

The output fields are as follows:

Field	Description
id	KPI internal ID as in the repository UI; for details, see the <i>OMi Administration Guide</i> .
name	KPI name

Return Codes

The API returns the following return codes:

Name	Error Code	Description
NOT_FOUND	404	KPI not found
INTERNAL_SERVER_ERROR	500	General failure

Get Indicator Statuses

You can use the following to retrieve indicator statuses.

API Syntax

`https://<Gateway Server>/topaz/servicehealth/customers/<CustomerId>/repositories/indicators/statuses`

The API uses the following parameter:

customerId. Customer ID (use **1** for non-HPE SaaS deployment).

The following is an example of the API and its output:

```
https://host.devlab.ad/topaz/servicehealth/customers/1/repositories/indicators/statuses
```

```
<targets>
  <target>
    <id>20</id>
    <name>OK</name>
  </target>
  <target>
    <id>15</id>
    <name>Warning</name>
  </target>
  <target>
    <id>10</id>
    <name>Minor</name>
  </target>
  <target>
    <id>5</id>
    <name>Major</name>
  </target>
  <target>
    <id>0</id>
    <name>Critical</name>
  </target>
  <target>
    <id>-1</id>
    <name>Info</name>
  </target>
  <target>
    <id>-2</id>
    <name>No Data</name>
  </target>
  <target>
    <id>-4</id>
    <name>Downtime</name>
  </target>
</targets>
```

The output fields are as follows:

Field	Description
id	KPI status internal ID
name	KPI status name

Return Codes

The API returns the following return codes:

Name	Error Code	Description
INTERNAL_SERVER_ERROR	500	General failure

Reset Health Indicator State API

In certain event flows, you might have an HI showing that a problem has occurred but no event has closed the problem, even though the problem was fixed. After dealing with the problem, you might want to reset the HI's state to **Normal** (default). For details on resetting HI state within Service Health, see the *OMi User Guide*.

The Reset HI State API enables users outside of the OMi user interface to reset event-based HIs to their default state, using the HTTP-based REST protocol.

You can reset all HIs on a specific CI, or reset a specific HI.

This REST API is case-sensitive, and uses the **PUT** method.

Note: This API can impact the overall performance of your system; consult with HPE Professional Services before using the API.

API Syntax

- To reset all HIs related to a CI:

```
https://<Gateway Server>/topaz/servicehealth/customers/<CustomerId>/cis/<CI ID>/his/reset
```

- To reset a specific HI:

```
https://<Gateway Server>/topaz/servicehealth/customers/<CustomerId>/cis/<CI ID>/his/<HI name>/reset
```

- To reset a specific subcomponent of an HI:

```
https://<Gateway Server>/topaz/servicehealth/customers/<CustomerId>/cis/<CI ID>/his/<HI name>/reset?subcomponent=<subcomponent name>
```

HI name refers to the name of the HI as defined in the indicator repository, and not to the HI's display label.

Return Codes and Log File

The API returns the following return codes:

Name	Error Code	Description
OK	200	Success
UNAUTHORIZED	401	The user is not authorized for the customer
NOT_FOUND	404	<ul style="list-style-type: none">• CI not found• HI not found• Bad request (syntax error)
INTERNAL_SERVER_ERROR	500	<ul style="list-style-type: none">• RTSM error• Repositories error• Online engine error

The service log file is located under: **<OMi_HOME_GW>/log/jboss/serviceHealthExternalAPI.log**.

In addition, the service writes to the Audit log on each HI reset.

Service Health Database Query API

You can use the Service Health API to query the database and return a list of views in XML format.

Tip: You can use XSLT to convert the XML output into any other format (commonly text or HTML). For example, using basic XSLT transformations, you can produce HTML reports that are formatted to fit on mobile devices. These reports can be served via a mobile portal to display critical Operations Manager i views on users' mobile phones.

Query Syntax

The basic syntax of the query is as follows:

```
https://<Gateway Server>/topaz/bam/BAMOpenApi?customerId=<customer ID>&userName=<user name>&password=<password>&command=<command parameter>
```

Depending on the **command** parameter defined, additional parameters may also be included.

Main Parameters Used in the Query

The following table lists the parameters that must be defined in the query.

Parameter	Description
customerID	OMi customers should specify 1.

Parameter	Description
userName	Specify a user name defined in OMi. The query does not encrypt the login credentials.
password	Specify the password for the user name provided. The query does not encrypt the login credentials.
command	Specify one of the following values: getView s – Specify to retrieve all views from the Run-time Service Model (RTSM). No other parameters are required. getNode s – Specify to retrieve all child nodes of a specified view (you must also specify the view for which to retrieve child nodes in the viewName parameter); if using this command parameter you can also set the following parameters: showTooltip , depth , layout , xsltURL , responseContentType
viewName	If the getNode s command parameter is defined, include this parameter in the query and specify the view to retrieve. You can set the value to ticker_all_views to retrieve all views and their nodes.
showTooltip	If the getNode s command parameter is defined, you can include this parameter in the query to specify whether to display Service Health's KPI tooltip data, either true to display data or false to not. The default value is false .
depth	If the getNode s command parameter is defined, you can include this parameter in the query to specify the number of levels in the view to display. The default value is 1 .
layout	If the getNode s command parameter is defined, you can include this parameter in the query to specify the layout for the query results, either hierarchical or flat . In flat mode all nodes are retrieved in a flat list, and in hierarchical mode nodes are retrieved in the same hierarchy as in the view. The default value is flat .
xsltURL	If the getNode s command parameter is defined, you can include this parameter in the query to specify a URL to an .xslt file that transforms the .xml-format result of the query.
responseContentType	If the getNode s command parameter is defined, and the xsltURL parameter is included in the query, you can include this parameter in the query to specify the response MIME type.

Query Examples

Below are examples of queries and the data they return.

The following query returns a flat list of all views in the Run-time Service Model (RTSM):

```
https://myserver/topaz/bam/BAMOpenApi?customerId=1  
&userName=admin&password=admin&command=getViews
```

The following query returns a hierarchical tree showing KPI status and tooltip information for the Service Measurements view, to a depth of three child nodes:

```
https://myserver/topaz/bam/BAMOpenApi?customerId=1&userName=  
admin&password=admin&command=getNodes&viewName=Service%20  
Measurements&showTooltip=true&depth=3&layout=hierarchical
```

Send Documentation Feedback

If you have comments about this document, you can [contact the documentation team](#) by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on OMi Extensibility Guide (Operations Manager i 10.11)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to ovdoc-asm@hpe.com.

We appreciate your feedback!



Go OMi!