



Hewlett Packard
Enterprise

HPE Application Performance Management

Software Version: 9.30

APM Extensibility Guide

Document Release Date: July 2016
Software Release Date: July 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 2005-2016 Hewlett Packard Enterprise Development LP

Trademark Notices

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

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

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.

iPod is a trademark of Apple Computer, Inc.

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

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

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

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/group/softwaresupport/search-result?keyword=>

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

PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format.

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 HPE Passport user and sign in. Many also require a support contract. To register for an HPE Passport ID, go to <https://softwaresupport.hpe.com> and click **Register**.

To find more information about access levels, go to:

<https://softwaresupport.hpe.com/web/softwaresupport/access-levels>

HPE Software Integrations, Solutions and Best Practices

Access the Hewlett Packard Enterprise Software Support site (<https://softwaresupport.hpe.com/manuals>) to search for a wide variety of best practice documents and materials.

Contents

Extensibility Guide Overview	8
Part 1: Service Health	9
Chapter 1: Service Health Rules API	10
API Group and Sibling Rule	11
API Sample Rule	13
API Duration-Based Sample Rule	14
Creating Rules with the Rules API	15
How to Define an API Rule in the CI Indicators Tab	15
How to Create a Text File-Based API Rule	16
How to Define an API Rule in the Rule Repository	20
How to Work with Tooltip Entries	20
How to Write to Log Files From the Rules API Code	21
How to Include a CI Property in Rules API Calculations	22
Examples - API Sample Rule	23
Example - Average Availability Rule	23
Example - Average Performance Rule	24
Example - Average Performance Rule Using a Rule Parameter Filter	24
Examples - API Group and Sibling Rule	26
Example - Worst Child Rule	26
Example - Worst Sibling Status Rule	27
Example - Specific Child CI Group Rule	28
Example - Sibling Rule Based on Availability and Performance KPIs	29
Example - Group Average Value by CI Type	30
Example - Worst Health Indicator Rule	30
Example - Using Groovy Closure	31
Chapter 2: Service Health External APIs	33
Retrieve Indicator Data API	33
API Syntax	33
Return Codes	35
API Syntax	35
Return Codes	36
API Syntax	37
Return Codes	38
Reset Health Indicator State API	38
Service Health Database Query API	39
Part 2: Service Level Management	42

Chapter 3: SLM External API	43
Get SLA Configuration Data	43
API Syntax	43
Return Codes	45
Get SLA Calculation Results	45
API Syntax	45
Return Codes	47
Get Calendars	47
API Syntax	47
Return Codes	48
Get Tracking Periods	48
API Syntax	48
Return Codes	49
Get KPIs	50
API Syntax	50
Return Codes	51
Get Indicator Statuses	51
API Syntax	51
Return Codes	52
Chapter 4: SLM Rules API	53
API Simplified Average Rules	54
API Group and Sibling Rule	55
Accessing a Specific Child KPI in the KPI Definition Page	56
Sample Rule Calculation Mechanism - Overview	57
Sample Rules: Calculating the KPI Based on Samples	57
Sample Rules: Calculating the KPI's Aggregated Results	58
When to Use Sample or Duration-Based Sample Rules	59
Example of Average Response Time Calculation	59
API Sample Rule	59
API Duration-Based Sample Rule	61
Duration-Based Sample Continuity	62
Filtering with the Duration-Based Sample Rule	63
API Outage by Samples Rule	64
Creating Rules with the Rules API	66
How to Define an API Rule for a Specific KPI or Outage	66
How to Create a Text File-Based API Rule	67
How to Define an API Rule Within the Rule Repository	70
How to Work with Tooltip Entries	70
How to Write to Log Files From the Rules API Code	72
How to Include a CI Property in Rules API Calculations	73
Examples - API Group and Sibling Rule	73

Examples - API Sample Rule	73
Example - Sample-Based Average Response Time Rule	74
Calculation - Sample-Based Average Response Time Rule	75
Example - Sample-Based Average Response Time Rule with Filter	76
Calculation - Sample-Based Average Response Time Rule with Filter	77
Example - Sample-Based Maximum Response Time Rule	77
Calculation - Sample-Based Maximum Response Time Rule	78
Examples - API Duration-Based Sample Rule	78
Example - Duration-Based Average Response Time Rule	79
Calculation - Duration-Based Average Response Time Rule	80
Example - Duration-Based Average Response Time Rule with isSampleValid Method Filter ..	81
Calculation - Duration-Based Average Response Time Rule with isSampleValid Method	
Filter	81
Example - Duration-Based Average Response Time Rule with isSampleAndDurationValid	
Method Filter	82
Calculation - Duration-Based Average Response Time Rule with	
isSampleAndDurationValid Method Filter	82
Example - Duration-Based Average Response Time Rule with isSampleAndDurationValid	
and isSampleValid Method Filters	83
Examples - API Outage by Samples Rule	84
Example - Outage by Samples Rule and Calculation with Default Rule Parameters	84
Example - Outage by Sample Calculation with Minimum Duration of 900 Seconds	86
Example - Outage by Sample Calculation with Maximum Duration of One Hour	86
Example - Outage by Sample Calculation with a Sample Representing Two Failures	87
Chapter 5: SLM Web Services API	88
Using the SLM Web Services	88
SLM Web Services' Operations	89
Part 3: User Management	93
Chapter 6: User Admin External API	94
Get All Groups/Users	94
API Syntax	94
Return Codes	95
Post New User	96
API Syntax	96
Return Codes	97
Get Specific User	97
API Syntax	97
Return Codes	98
Get Specific Group	99
API Syntax	99
Return Codes	100

Delete User from Group	100
API Syntax	100
Return Codes	101
Post Existing User to a Group	101
API Syntax	101
Return Codes	101
Part 4: End User Management	103
Chapter 7: EUM Admin Open API	104
Part 5: SiteScope	105
Chapter 8: SiteScope Public API	106
Part 6: Service Health Analyzer	107
Chapter 9: Groovy	108
Groovy For Anomaly Detection	108
Groovy for Anomaly Severity	110
Groovy for Custom Drilldowns from the SHA Investigation UI	112
Part 7: Downtime	117
Chapter 10: Downtime REST Service	118
Downtime Schedule Examples	120
Example of a Downtime Schedule with One Occurrence	120
Example of a Weekly Downtime Schedule	121
Example of a Monthly Downtime Schedule	121
Downtime REST Examples using Java Code	121
Downtime REST Example Using Groovy	122
Import Example	122
Part 8: Reporting in APM	124
Chapter 11: Generic Reporting Engine API	125
Data Returned	126
Querying with a Browser	126
Using the Web Service	127
Supported SQL Syntax	127
Supported Functions	128
Query Limitations	129
Date-Time Values	130
byTime Function	130
Query Examples	131
Send Documentation Feedback	133

Extensibility Guide Overview

This guide describes how to customize Application Performance Management applications, extend their functionality, and use APIs to perform operations. The guide is intended for administrators and integrators who need to establish advanced configurations and extensions. If you are an administrator and need to set up APM, first refer to the APM Application Administration Guide.

This guide provides instructions for working with the following:

- **Service Health.** For details, see ["Service Health Rules API" on page 10](#) and ["Service Health External APIs" on page 33](#).
- **Service Level Management.** For details, see ["SLM External API " on page 43](#), ["SLM Rules API" on page 53](#), and ["SLM Web Services API" on page 88](#).
- **End User Management.** For details, see ["EUM Admin Open API" on page 104](#).
- **SiteScope.** For details, see ["SiteScope Public API" on page 106](#).
- **Downtime.** For details, see ["Downtime REST Service" on page 118](#)
- **Reports.** For details, see ["Generic Reporting Engine API" on page 125](#).

Part 1: Service Health

Chapter 1: Service Health Rules API

Note: In APM versions 9.00 and later, the rules that calculate indicator statuses and values based on samples ("[API Sample Rule](#)" on page 13 and "[API Duration-Based Sample Rule](#)" on page 14) 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 APM 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. The default Service Health rules appear in the section Understanding the Service Health Calculation Rules in the APM Application 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 APM administration and applications.

The Rules API classes are documented in Javadoc format in the *HP Rules API Reference*. These files are located in the following folder:

```
\\< Gateway Server root directory>
```

```
\AppServer\webapps\site.war\amdocs\eng\doc_lib\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 13.
- **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 14.

Creating API Rules

Rules can be created using the Rules API in these ways:

- Using the CI Indicators tab 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 15.

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 20.

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 21](#).

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 KPIs Repository page in the APM Application Administration Guide.

Group and Sibling Rule Methods and Fields

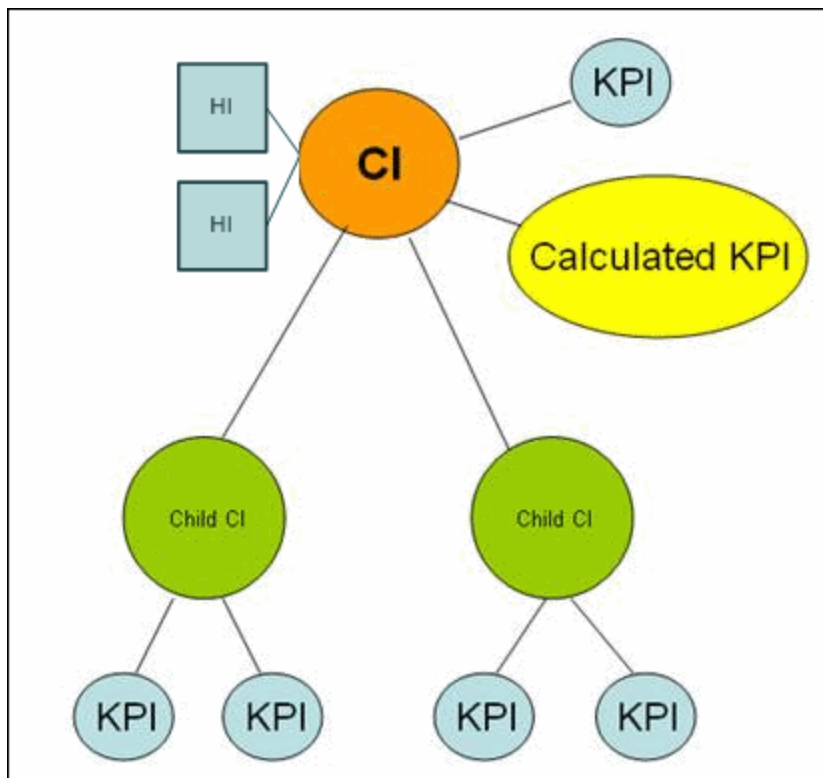
The Group and Sibling rule implements the Rules API Interface **GroupAndSiblingCalculator**, using the following guidelines:

- 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 *HP Rules API Reference*. These files are located in the following folder:

\\<Gateway Server root directory>\

AppServer\webapps\site.war\amdocs\eng\doc_lib\API_docs\Rules_API\index.html.

For detailed examples of Group and Sibling rules, see ["Examples - API Group and Sibling Rule" on page 26](#).

API rules can be defined within the Service Health CI Indicators tab or Rule Repository, or using a text file template, as described in ["Creating Rules with the Rules API" on page 15](#).

Defining a Group and Sibling Rule in the CI Indicators tab or Rule Repository

To define a Group and Sibling rule using the CI Indicators tab 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 CI Indicators Tab" on page 15](#) or ["How to Define an API Rule in the Rule Repository" on page 20](#).

Accessing a Specific Child KPI in the CI Indicators Tab

When creating a Group rule for a specific KPI in the CI Indicators tab, 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 28](#) in ["Examples - API Group and Sibling Rule" on page 26](#).

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 16](#).

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 23](#).

API rules can be defined within the Service Health CI Indicators tab or the Rule Repository, or using a text file template, as described in ["Creating Rules with the Rules API" on page 15](#).

The Rules API classes are documented in Javadoc format in the *HP Rules API Reference*. These files are located in the following folder:

```
\\<Gateway Server root directory>
```

```
AppServer\webapps\site.war\amdocs\eng\doc_lib\API_docs\Rules_API\index.html.
```

Defining a Sample Rule in the CI Indicators tab or Rule Repository

To define a Sample rule using the CI Indicators tab 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 CI Indicators Tab" on page 15](#) or ["How to Define an API Rule in the Rule Repository" on page 20](#).

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 16](#).

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 23](#).

API rules can be defined using the Service Health CI Indicators tab, using a text file, or within the Rule Repository, as described in ["Creating Rules with the Rules API" on the next page](#).

The Rules API classes are documented in Javadoc format in the *HP Rules API Reference*. These files are located in the following folder:

```
\\<Gateway Server root directory>\
AppServer\webapps\site.war\amddocs\eng\doc_lib\API_docs\Rules_API\index.html.
```

Defining a Duration-Based Sample Rule in the CI Indicators tab or Rule Repository

To define a Duration-Based Sample rule using the CI Indicators tab 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 CI Indicators Tab"](#) below or ["How to Define an API Rule in the Rule Repository"](#) on page 20.

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 the next page.

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 CI Indicators tab

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 CI Indicators tab, 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 CI Indicators tab at any time to change the rule logic for the KPI.

For details, see ["How to Define an API Rule in the CI Indicators Tab"](#) 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 CI Indicators tab 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 20.

How to Define an API Rule in the CI Indicators Tab

Each KPI has three applicable API rules. Within the CI Indicators tab, 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 **Admin > Service Health > CI Indicators**. Select **New KPI** to assign a new KPI to the CI, or **Edit KPI** to modify an existing KPI. For details on this process, see Adding KPIs to CIs, or Modifying KPI Settings - KPIs Tab in the APM Application 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 10.

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 11
- "[API Sample Rule](#)" on page 13
- "[API Duration-Based Sample Rule](#)" on page 14

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 11
- "[API Sample Rule](#)" on page 13
- "[API Duration-Based Sample Rule](#)" on page 14

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**

- a. Select **Admin > Service Health > Repositories > Business Rules > New Rule**. For details on adding rules, see Customizing KPI and HI Calculation Rules in the APM Application Administration Guide.

- b. In the **Name** field, type the name of the rule you want to create (mandatory).
- c. 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.
- d. Create Rule parameters depending on your API rule type, as follows:
 - o In the **Rule parameters** area, click **New**.
 - o 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.
 - o 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**.

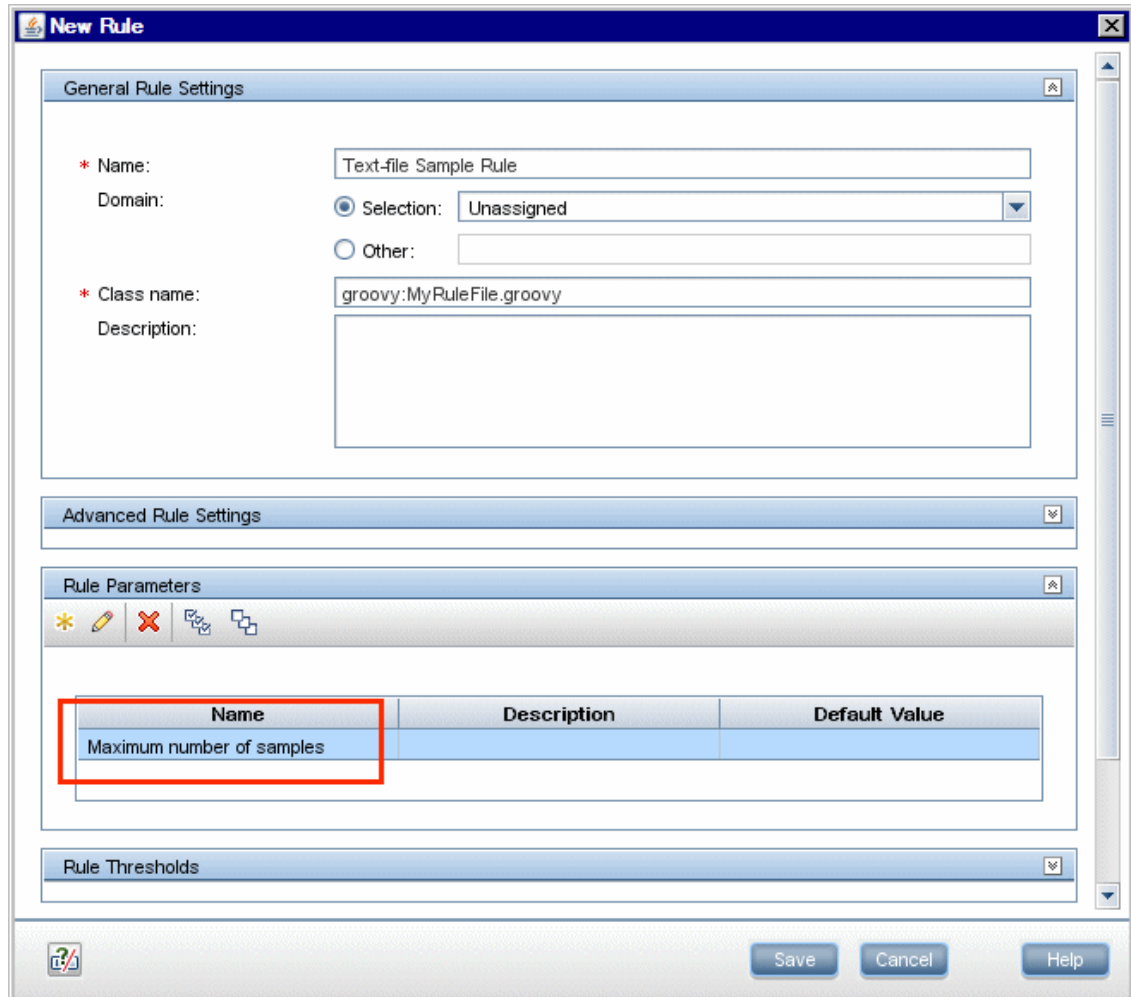
Click **OK** to Save.

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.)
 - o In the **Threshold parameters** area, click **New**.
 - o In the **Name** field, type **critical**. In the **Type** field, select **Float**.

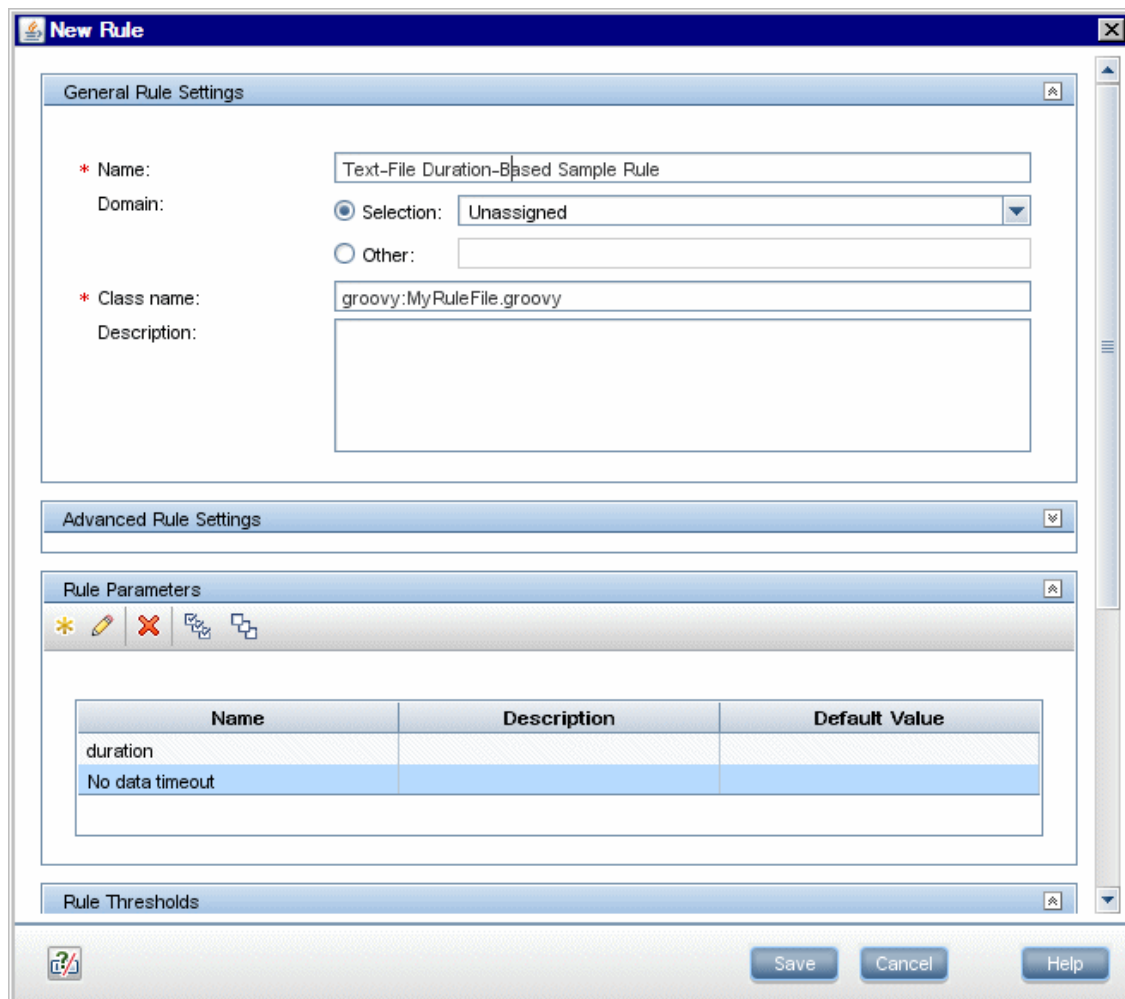
When defining the **operator** parameter, select **String** in the **Type** field.
 - o Click **OK** to save.

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 Main Settings Area > Applicable Rules parameter in New KPI/Edit KPI Dialog Box in the APM Application 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 the next page.](#)

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.

- In the browser, access JMX port <29810 + workerID> (for example, 29811 for worker_1).
- Within **BSM-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 CI Indicators tab at any time to change the rule logic for the specific KPI.

1. **Clone an API rule**

Select **Admin > Service Health > Repositories > 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 Customizing KPI and HI Calculation Rules in the APM Application 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 CI Indicators tab 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 CI Indicators tab 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):

- o ["API Group and Sibling Rule" on page 11](#)
- o ["API Sample Rule" on page 13](#)
- o ["API Duration-Based Sample Rule" on page 14](#)

The Rules API classes are documented in Javadoc format in the *HP Rules API Reference*. These files are located in the following folder:

```
\\<Gateway Server root directory>\
AppServer\webapps\site.war\amdocs\eng\doc_lib\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 New KPI/Edit KPI Dialog Box in the APM Application 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. Select **Admin > Service Health > Repositories > 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 Customizing Tooltips in the APM Application 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 **<Data Processing server root directory>\HPBSM\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 **<Data Processing server root directory>HPBSM\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.blDashboardRules.simplifiedRule.groupAndSiblingRule.DashboardGroupAndSiblingRule = ${loglevel}, bam.app.rules.api.group.appender`
 - Sample rules: `log4j.category.com.mercury.am.rules.dashboard.blDashboardRules.simplifiedRule.leaf.DashboardSimplifiedSampleBasedRule = ${loglevel}, bam.app.rules.api.leafsample.appender`
 - Duration-Based Sample rules: `log4j.category.com.mercury.am.rules.dashboard.blDashboardRules.simplifiedRule.leaf.DashboardSimplifiedTimeBasedRule = ${loglevel}, bam.app.rules.api.leafduration.appender`

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** - SLM and Service Health
- **BLE_ONLINE_ATTRIBUTE** - Service Health only

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**. Access the RTSM JMX Console – <APM Data Processing>:21212/jmx-console/HtmlAdaptor?action=inspectMBean&name=UCMDB:service=TQL%20Services, and invoke **retrieveTqlNames()**. 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 the next page](#)

Example - Average Availability Rule

The following rule calculates average availability of samples, based on the `u_iStatus` sample field.

The rule logic is $(\text{available samples} / \text{total samples}) * 100$.

```
// 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 Customizing Rule Parameters and Thresholds in the APM Application Administration Guide.

The rule logic is: $\text{sum}(\text{dResponseTime}) / \text{available samples}$.

```
/ 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" below](#)
- ["Example - Worst Sibling Status Rule" on the next page](#)
- ["Example - Specific Child CI Group Rule" on page 28](#)
- ["Example - Sibling Rule Based on Availability and Performance KPIs" on page 29](#)
- ["Example - Group Average Value by CI Type" on page 30](#)
- ["Example - Worst Health Indicator Rule" on page 30](#)
- ["Example - Using Groovy Closure" on page 31](#)

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 "bpm_tx_from_location" 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 "bpm_tx_from_location".  
    List<KPI> bpmTxFromLocationChildKpiList = ci.getChildrenKPIsByIDAndCiType(kpiId,  
"bpm_tx_from_location")  
    // 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  
bpm_tx_from_location: " + bpmTxFromLocationChildKpiList.size())  
    // Go over the list of child KPIs.  
    bpmTxFromLocationChildKpiList.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, bpmTxFromLocationChildKpiList.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 2: 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 38. 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 39. 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 the following topics:

- "Get KPI Over Time Statuses" below
- "Get KPI Definitions" on page 35
- "Get Indicator Statuses" on page 36

The service log file is located under: **<Gateway server root directory>\log\EJBContainer\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

```
http://<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 seconds since January 1 1970).
- **endDate**. Mandatory; end time for the KPI status (value representing the date in seconds since January 1 1970).
- **view**. Optional; retrieve the results in the context of a local impact view (default is global view). For details,

see Local Impact Views in the APM User Guide.

- **kpiId.** Optional; use comma separated KPI internal IDs as in the repository UI (default is empty for all KPIs). For details, see List of Service Health KPIs in the APM Application Administration Guide.

The following is an example of the API and its output:

```
http://host.devlab.ad/topaz/servicehealth/customers/1/kpiOverTime?  
ciIds=0b656ce308022a6739e3e726497fda6a&startDate=1296499370  
&endDate=1296501466
```

```
<kpiStatuses>  
  <kpiStatus>  
    <ciId>0b656ce308022a6739e3e726497fda6a</entityId>  
    <ciDisplayLabel>ATM 1610</ciDisplayLabel>  
    <kpiType>6</kpiType>  
    <kpiDisplayName>Application Performance</kpiDisplayName>  
    <timeStamp>1296499370</timeStamp>  
    <status>20</status>  
    <statusDisplayName>OK</statusDisplayName>  
    <duration>311</duration>  
  </kpiStatus>  
  <kpiStatus>  
    <ciId>0b656ce308022a6739e3e726497fda6a</entityId>  
    <ciDisplayLabel>ATM 1610</ciDisplayLabel>  
    <kpiType>6</kpiType>  
    <kpiDisplayName>Application Performance</kpiDisplayName>  
    <timeStamp>1296499681</timeStamp>  
    <status>-2</status>  
    <statusDisplayName>No Data</statusDisplayName>  
    <duration>1785</duration>  
  </kpiStatus>  
  <kpiStatus>  
    <ciId>0b656ce308022a6739e3e726497fda6a</entityId>  
    <ciDisplayLabel>ATM 1610</ciDisplayLabel>  
    <kpiType>6</kpiType>  
    <kpiDisplayName>Application Performance</kpiDisplayName>  
    <timeStamp>1296501466</timeStamp>  
    <status>20</status>  
    <statusDisplayName>OK</statusDisplayName>  
    <duration>13334</duration>  
  </kpiStatus>  
  <kpiStatus>  
    <ciId>0b656ce308022a6739e3e726497fda6a</entityId>  
    <ciDisplayLabel>ATM 1610</ciDisplayLabel>  
    <kpiType>7</kpiType>  
    <kpiDisplayName>Application Availability</kpiDisplayName>  
    <timeStamp>1296428400</timeStamp>  
    <status>0</status>  
    <statusDisplayName>Critical</statusDisplayName>
```

```
<duration>69663</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 seconds 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 seconds.

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 ciIDs 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

```
http://<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 List of Service Health KPIs in the APM Application Administration Guide.

The following is an example of the API and its output:

```
http://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 List of Service Health KPIs in the APM Application Administration Guide.
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

http://<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:

```
http://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 Health Indicator Component in the APM User Guide.

The Reset HI State API enables users outside of the APM 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:

```
http://<Gateway Server>/topaz/servicehealth/customers/<CustomerId>/cis/<CI ID>/his/reset
```

- To reset a specific HI:

```
http://<Gateway Server>/topaz/servicehealth/customers/<CustomerId>/cis/<CI ID>/his/<HI name>/reset
```

- To reset a specific subcomponent of an HI:

```
http://<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: **<Gateway server root directory>\log\EJBContainer\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 Application Performance Management views on users' mobile phones.

Query Syntax

The basic syntax of the query is as follows:

```
http://<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.

Supported Parameters Used in the Query

The following table lists the parameters that must be defined in the query.

Parameter	Description
customerID	APM customers should specify 1. HPE Software-as-a-Service customers should specify their unique customer ID.

Parameter	Description
userName	Specify a user name defined in APM. 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):

```
http://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:

```
http://myserver/topaz/bam/BAMOpenApi?customerId=1&userName=admin&password=admin&command=getNodes&viewName=Service%20Measurements&showTooltip=true&depth=3&layout=hierarchical
```

Part 2: Service Level Management

Chapter 3: SLM External API

You can use the SLM external API to retrieve SLA configuration properties and SLA calculation results, in order to consume this data in external applications.

The API enables you to access and process SLM data from outside APM. For details, see:

- "Get SLA Configuration Data" below
- "Get SLA Calculation Results" on page 45
- "Get Calendars" on page 47
- "Get Tracking Periods" on page 48
- "Get KPIs" on page 50
- "Get Indicator Statuses" on page 51

The service log file is located under: **<Gateway server root directory>\log\EJBContainer\slmExternalAPI.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 SLA Configuration Data

You can use the following to get SLA configuration data.

API Syntax

```
http://<Gateway Server>/topaz/slm/customers/<CustomerId>/sla/<slaId>
```

The API uses the following parameters:

- **customerId.** Customer ID (use **1** for non-HPE SaaS deployment).
- **slaid.** SLA RTSM ID (non-mandatory parameter; leave empty to get all SLAs in the system).

The following is an example of the API output:

```
<sla>
  <id>c808d3ddce3d849c1cc0d667bec7f6cc</id>
  <name>TestSLA3</name>
  <description></description>
  <details></details>
  <timezone>Central Standard Time</timezone>
  <type>SLA</type>
  <creator>administrator</creator>
  <url></url>
  <classification>Formal</classification>
  <customer>
```

```

    <customerId>e35cded4b3a628a8da4b9cf352007467</customerId>
    <customerName>IT Department</customerName>
  </customer>
  <provider>
    <providerId>ae9ea545c5f99adb52d7edd641bbf8ef</providerId>
    <providerName>Customers</providerName>
  </provider>
  <state>Running</state>
  <startDate>1290938400</startDate>
  <endDate>1608631200</endDate>
  <targets>
    <target>0</target>
    <target>5</target>
    <target>20</target>
    <target>10</target>
    <target>15</target>
  </targets>
  <trackingPeriods>
    <trackingPeriod>3ef2dfe04fa07c349e72ca9e7b04e2af</trackingPeriod>
    <trackingPeriod>bffe0ea9334ff7f309e969eec3c9c266</trackingPeriod>
    <trackingPeriod>a562b09777271425835abadb12f32e73</trackingPeriod>
    <trackingPeriod>f9f77b20f986fbb2eb064b0a30ece93d</trackingPeriod>
    <trackingPeriod>4ecf36e0eb88816745a8849db029c73f</trackingPeriod>
  </trackingPeriods>
  <calendars>
    <calendar>ecf840eef788851986195301aba206fd</calendar>
  </calendars>
</sla>

```

The output fields are as follows:

Field	Description
slald	SLA ID
name	SLA name
description	SLA description
details	SLA details
type	OLA, SLA, or UC
creator	Username of the user which created the SLA
timezone	SLA time zone
url	Link to the contract details from an external source.
classification	Formal or informal
customer	CI ID and name of the organization specified as the customer of the SLA.

Field	Description
provider	CI ID and name of the organization specified as the provider of the SLA.
state	Preliminary, Running, or Terminated
startDate	SLA start date, long value representing the date in milliseconds since January 1, 1970. 00:00:00 GMT.
endDate	SLA end date, long value representing the date in milliseconds since January 1, 1970.
targets	IDs of the SLA targets
trackingPeriods	IDs of the tracking periods
calenders	IDs of the SLA calendars

Return Codes

The API returns the following return codes:

Name	Error Code	Description
UNAUTHORIZED	401	User has no view permission for selected SLA
NOT_FOUND	404	SLA not found
INTERNAL_SERVER_ERROR	500	General failure

Get SLA Calculation Results

You can use the following to retrieve the SLA calculation results; this API supports filtering by CI, KPI, tracking periods, and calendars.

API Syntax

```
http://<Gateway Server>/topaz/slm/customers/  
<customerId>/ciSummary/<view>/sla/ <slaId>?  
calendar=<calendarId>&startDate=<startDate>&endDate=<endDate>&ciIds=<ciIds>&kpiId=<kpiId>
```

The API uses the following parameters:

- **customerId.** Customer ID (use **1** for non-HPE SaaS deployment).
- **view.** Mandatory; report view. Use one of the following: hour, day, week, month, quarter, year, weekToDate, monthToDate, quarterToDate, yearToDate or slaToDate.
- **slaid.** SLA ID (use **"all"** for all SLAs).
- **calendar.** Optional, SLA calendar ID (leave empty for all SLA defined calendars, see ["Get Calendars" on page 47](#)).
- **startDate.** Mandatory for closed periods only, in seconds.

- **endDate**. Mandatory for closed periods only, in seconds.
- **cilds**. Optional. Comma separated CI IDs; leave empty for all SLA CIs (or SLA nodes when slaid is "all").
- **kpiid**. Optional. KPI internal ID (as it appears in the KPI repository UI); leave empty for all KPIs. This API does not support Outage KPI.

The following is an example of the API output:

```
<slaStatus>
  <slaid>2c4d69f2784021a6a94b7a40100ba31f</slaid>
  <ciid>b8ae7539b0cb338f3bce84b4866647eb</id>
  <startDate>1293832800</startDate>
  <endDate>1294225200</endDate>
  <ciName>someName</ciName>
  <trackingPeriod>a562b09777271425835abadb12f32e73</trackingPeriod>
  <calendar>dd9ef8826c553d3e217fa0b3bf03f0a0</calendar>
  <kpiType>220</kpiType>
  <kpiDisplayName>SLM Status</kpiDisplayName>
  <kpiStatus>0</kpiStatus>
  <statusDisplayName>Failed</statusDisplayName>
  <kpiValue>0</kpiValue>
</slaStatus>
```

The output fields are as follows:

Field	Description
slaid	SLA ID
cild	CI ID
startDate	Period start time, long value representing the date in seconds since January 1 1970
endDate	Period end time, long value representing the date in seconds since January 1 1970
ciName	If you make a request for a single SLA, the CI name is displayed along with the CI ID. The CI name is not displayed if the CI ID and SLA ID are the same.
trackingPeriod	Tracking period of the sample: hour, day, week, month, quarter, year, SLA period (see "Get Tracking Periods" on page 48).
calendar	Calendar of the sample (see "Get Calendars" on the next page).
kpiType	KPI ID (see "Get KPIs" on page 50).
kpiDisplayName	KPI display name
kpiStatus	KPI status ID (see "Get Indicator Statuses" on page 51).
statusDisplayName	KPI status display name
kpiValue	KPI numeric value

Return Codes

The API returns the following return codes:

Name	Error Code	Description
BAD_REQUEST	400	<ul style="list-style-type: none">• Unsupported view• Start date is after the end date• startDate, endDate are missing for closed view (closed period)
UNAUTHORIZED	401	User has no view permission for selected SLA
INTERNAL_SERVER_ERROR	500	<ul style="list-style-type: none">• Result size has exceeded the maximum quota• General failure

Get Calendars

You can use the following to retrieve the calendars defined in the system.

API Syntax

```
http://<Gateway Server>/topaz/slm/customers/  
<CustomerId>/repositories/calendars/<calendarId>
```

The API uses the following parameters:

- **customerId.** Customer ID (use **1** for non-HPE SaaS deployment).
- **calendarId.** Calendar ID (leave empty for all calendars in the system).

The following is an example of the API output:

```
<calendars>  
  <calendar>  
    <id>5c6c09ec3d61db775333bb5beb5ea863</id>  
    <name>testcalc</name>  
    <description></description>  
    <scheduleDescription>Friday 3:00 AM - 3:30 AM, Wednesday 2:30 PM - 3:00  
    PM, Thursday 5:30 PM - 6:00 PM, Wednesday 3:30 AM - 4:30 AM, Tuesday 11:00 AM  
    - 11:30 AM, Friday 10:00 AM - 10:30 AM, Monday 2:00 AM - 2:30 AM, Wednesday 7:00 AM -  
    7:30 AM, Thursday 10:30 AM - 11:00 AM, Sunday, Monday 6:30 AM - 7:00 AM, Friday 2:00  
    PM - 2:30 PM, Monday 8:00 PM - 8:30 PM, Thursday 9:00 PM - 9:30 PM  
    </scheduleDescription>  
  </calendar>  
  <calendar>  
    <id>ecf840eef788851986195301aba206fd</id>  
    <name>24x7</name>  
    <description>24 hours, 7 days a week</description>  
    <scheduleDescription>Sunday, Monday, Tuesday, Wednesday, Thursday, Friday,
```

```
Saturday 12:00 AM - 12:00 AM
  </scheduleDescription>
</calendar>
<calendar>
  <id>86594b724cb03e2647b2a86b08103a28</id>
  <name>Business Hours</name>
  <description>8:00-17:00, Monday-Friday</description>
  <scheduleDescription>Monday, Tuesday, Wednesday, Thursday, Friday 8:00 AM -
5:00 PM
  </scheduleDescription>
</calendar>
</calendars>
```

The output fields are as follows:

Field	Description
id	Calendar ID
name	Calendar display name
description	Calendar description
scheduleDescription	A string representing the calendar scheduling configuration.

Return Codes

The API returns the following return codes:

Name	Error Code	Description
NOT_FOUND	404	Calendar not found
INTERNAL_SERVER_ERROR	500	General failure

Get Tracking Periods

You can use the following to retrieve the tracking periods defined in the system.

API Syntax

```
http://<Gateway Server>/topaz/slm/customers/
<CustomerId>/trackingPeriods/<trackingPeriodId>
```

The API uses the following parameters:

- **customerId.** Customer ID (use **1** for non-HPE SaaS deployment).
- **trackingPeriodId.** Tracking period ID (leave empty for all tracking periods in the system).

The following is an example of the API output:

```
<trackingPeriods>
  <trackingPeriod>
    <id>a562b09777271425835abadb12f32e73</id>
    <name>Month</name>
  </trackingPeriod>
  <trackingPeriod>
    <id>dd613d616284c14ec848a4fa5d939655</id>
    <name>Hour</name>
  </trackingPeriod>
  <trackingPeriod>
    <id>2e57ee4d9c4f6b50a1d01385a861aa4b</id>
    <name>Day</name>
  </trackingPeriod>
  <trackingPeriod>
    <id>4ecf36e0eb88816745a8849db029c73f</id>
    <name>Week</name>
  </trackingPeriod>
  <trackingPeriod>
    <id>3ef2dfe04fa07c349e72ca9e7b04e2af</id>
    <name>Quarter</name>
  </trackingPeriod>
  <trackingPeriod>
    <id>f9f77b20f986fbb2eb064b0a30ece93d</id>
    <name>Year</name>
  </trackingPeriod>
  <trackingPeriod>
    <id>bffe0ea9334ff7f309e969eec3c9c266</id>
    <name>SLA period</name>
  </trackingPeriod>
</trackingPeriods>
```

The output fields are as follows:

Field	Description
id	Tracking period ID
name	Tracking period display name

Return Codes

The API returns the following return codes:

Name	Error Code	Description
INTERNAL_SERVER_ERROR	500	General failure

Get KPIs

You can use the following to retrieve the KPIs defined in the system.

API Syntax

```
http://<Gateway Server>/topaz/slm/customers/  
<CustomerId>/repositories/indicators/kpis/<kpiId>
```

The API uses the following parameters:

- **customerid.** Customer ID (use **1** for non-HPE SaaS deployment).
- **kpiid.** KPI ID (leave empty for all KPIs in the system).

The following is an example of the API output:

```
<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>  
  <kpi>  
    <id>615</id>  
    <name>Operational Status</name>  
  </kpi>  
  <kpi>  
    <id>6</id>  
    <name>Application Performance</name>  
  </kpi>  
  <kpi>  
    <id>7</id>  
    <name>Application Availability</name>  
  </kpi>  
  <kpi>  
    <id>1500</id>
```

```
    <name>Generic</name>  
  </kpi>  
</kpis>
```

The output fields are as follows:

Field	Description
id	KPI ID
name	KPI display 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

```
http://<Gateway Server>/topaz/slm/customers/  
<CustomerId>/repositories/indicators/targets
```

The API uses the following parameter:

customerId. Customer ID (use **1** for non-HPE SaaS deployment).

The following is an example of the API output:

```
<targets>  
  <target>  
    <id>20</id>  
    <name>Exceeded</name>  
  </target>  
  <target>  
    <id>15</id>  
    <name>Met</name>  
  </target>
```

```
<target>
  <id>10</id>
  <name>Minor Breached</name>
</target>
<target>
  <id>5</id>
  <name>Breached</name>
</target>
<target>
  <id>0</id>
  <name>Failed</name>
</target>
<target>
  <id>-4</id>
  <name>Downtime</name>
</target>
<target>
  <id>-2</id>
  <name>No Data</name>
</target>
</targets>
```

The output fields are as follows:

Field	Description
id	Target/Status ID (as it appears in the calculation results)
name	Target/Status display name

Return Codes

The API returns the following return codes:

Name	Error Code	Description
INTERNAL_SERVER_ERROR	500	General failure

Chapter 4: SLM Rules API

Note: In APM versions 9.00 and later, the rules that calculate indicator statuses and values based on samples ("[API Sample Rule](#)" on page 59 and "[API Duration-Based Sample Rule](#)" on page 61) 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 APM 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 business rules to calculate Key Performance Indicators (KPIs). The default Service Level Management rules appear in the section List of Service Level Management Business Rules in the APM Application Administration Guide.

The recommended way to create new rules is with the Rules API. The Rules API enables you to create rules using the Groovy dynamic scripting language with Groovy runtime environment 1.7.3 and earlier. Users of this API should be familiar with Groovy and Java syntax, and with APM administration and applications.

The Rules API classes are documented in Javadoc format in the *HP Rules API Reference*. These files are located in the following folder:

```
\\<Gateway Server root directory>\
AppServer\webapps\site.war\amddocs\eng\doc_lib\API_docs\Rules_API\index.html.
```

Note: Extensive use of rules which use the Rules API will affect the performance of SLA calculation.

Types of API Rules

The following are the API rules in Service Level Management:

- **Simplified Average Rule and Simplified Duration-Based Average Rule.** These rules calculate health indicator average values based on data taken from sample fields, and enables you to manipulate these values. These rules are simpler to define than other API rules, and only require you to enter the sample field you want used in the calculation. For details, see "[API Simplified Average Rules](#)" on the next page.
- **Group and Sibling Rule.** This rule calculates KPIs based on aggregated values received from other KPIs, rather than from original sample data. The received data can come from the KPIs of child CIs (group), or from another KPI associated with the same CI (sibling). For details, see "[API Group and Sibling Rule](#)" on page 55.
- **Sample Rule.** This rule calculates KPIs based on data taken from sample fields. For details, see "[API Sample Rule](#)" on page 59.
- **Duration-Based Sample Rule.** This rule calculates KPIs based on data taken from sample fields, and uses both the sample's value and its duration within the rule calculation (for example, availability over time or backlog over time). For details, see "[API Duration-Based Sample Rule](#)" on page 61.
- **Outage by Samples Rule.** This rule calculates outages based on data received from samples. For details, see "[API Outage by Samples Rule](#)" on page 64.

For details on how sample rules are calculated, see "[Sample Rule Calculation Mechanism - Overview](#)" on page 57.

Creating API Rules

Rules can be created using the Rules API in three ways:

- Using the KPI Definition 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 template in the rule repositories.

These methods are described in ["Creating Rules with the Rules API" on page 66](#).

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 70](#).

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 72](#).

API Simplified Average Rules

The API Simplified Average Rule and API Simplified Duration-Based Average Rule calculate health indicator average values based on data taken from sample fields, and enable you to easily manipulate these calculations.

API Simplified Average Rule

The Simplified Average Rule calculates an average of sample field values, uses the rule parameters: Enumerator and Denomination.

- In the **Enumerator** area, enter the name of the sample field you want to use in the calculation in the following format: **sample.<field name>**.

To calculate the average value of the sample field `dValue`, type `sample.dValue` in the Enumerator. The rule takes the values of this field from the samples that come in during the calculation cycle, and calculates their average. This average becomes the value of the health indicator for the cycle.

You can also use mathematical expressions to manipulate these calculations. For example, suppose you have an EMS integration sending samples with two values: `attr1` representing sent text messages, and `attr2` representing failed text messages. To calculate the average number of successful messages (`attr1-attr2`), type `sample.attr1-sample.attr2` in the Enumerator.

- You can use the **Denomination** area to perform actions on the above value; the number or value you enter in this area is used to divide the Enumerator result.

For example, if you enter `sample.dValue` in the Enumerator, and `2` in the Denomination, the average of the `dValue` fields on the samples is divided by two, and the health indicator value is half the average of the sample fields.

API Simplified Duration-Based Average Rule

The Simplified Duration-Based Average Rule calculates an average of sample field values, taking the duration values from the samples themselves.

In the **Enumerator** field, enter the name of the sample field you want to use in the calculation in the following format: **Sample.<field name>**.

The rule takes the durations from the samples and calculates the average of sample field value, divided by their durations.

For example, if the value of the specified field was 10 for 40 minutes and 5 for 20 minutes, the rule calculates $(10 \times 40) + (5 \times 20) / 60 = 8.33$. The health indicator's average value for this hour will be 8.33.

API Group and Sibling Rule

An API Group and Sibling Rule calculates KPIs based on data received from other KPIs, 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 New KPI/Edit KPI Dialog Box in the APM Application Administration Guide.

The KPI is calculated based on the aggregated values of the group or sibling KPIs or HIs. The calculated KPI results represent the aggregated results.

Group and Sibling Rule Methods and Fields

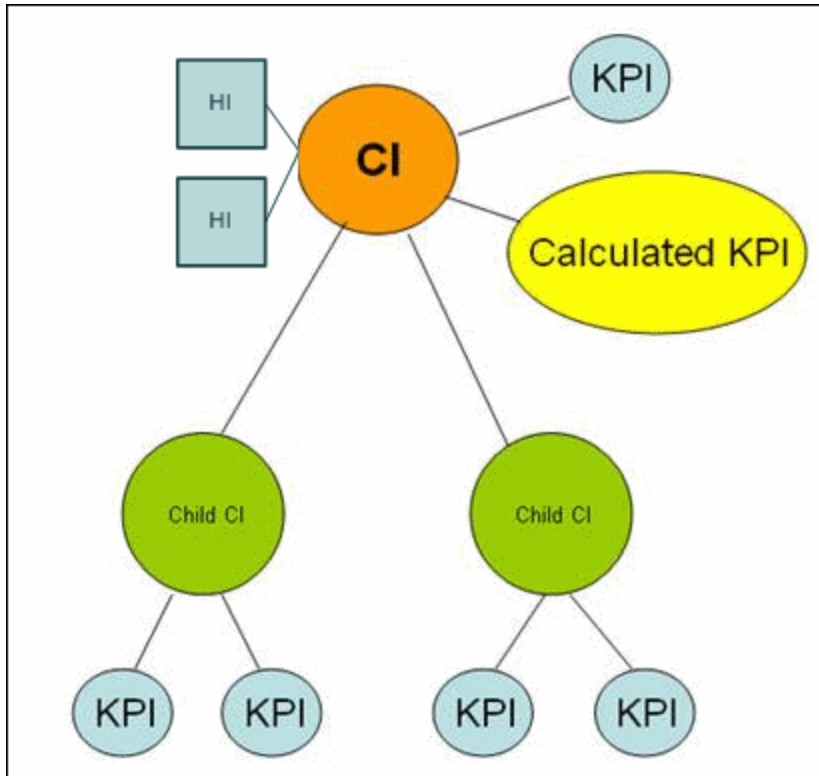
The Group and Sibling rule implements the Rules API Interface **GroupAndSiblingCalculator**, using the following guidelines:

- 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 or HI 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** objects returned by **CI** are used to get the aggregated values of these KPIs or HIs.
- 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 and HIs.



The Rules API classes are documented in Javadoc format in the *HP Rules API Reference*. These files are located in the following folder:

\\<Gateway Server root directory>\

AppServer\webapps\site.war\amdocs\eng\doc_lib\API_docs\Rules_API\index.html.

For examples of Group and Sibling rules, see ["Examples - API Group and Sibling Rule" on page 73](#).

API rules can be defined within the KPI Definition page or Rule Repository, or using a text file template, as described in ["Creating Rules with the Rules API" on page 66](#).

Defining a Group and Sibling Rule in the KPI Definition Page or Rule Repository

To define a Group and Sibling rule using the KPI Definition 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 for a Specific KPI or Outage" on page 66](#) or ["How to Define an API Rule Within the Rule Repository" on page 70](#).

Accessing a Specific Child KPI in the KPI Definition Page

When creating a Group rule for a specific KPI in the KPI Definition 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 ["Examples - API Group and Sibling Rule" on page 73](#).

Defining a Group and Sibling Rule Using a Text File

To define a Group and Sibling rule using a text file, use the **SlmGroupAndSiblingTemplate.groovy** template as described in ["How to Create a Text File-Based API Rule" on page 67](#).

Within the text file, enter the **calculateKPI** method body.

Sample Rule Calculation Mechanism - Overview

The API Sample rule and Duration-Based Sample rule calculate KPIs based on sample values, for each combination of calendar and tracking period. This calculation process is divided into calculation cycles.

Each calculation cycle, the following steps occur:

1. The samples that are in the Profile database for the calculation cycle time are retrieved. For example, if a KPI is calculated for 10:00-11:00 and the cycle duration is 5 minutes, the samples that are in the database with the timestamp 10:00-10:05 are used in the first cycle's calculation.
2. The samples go through a filtering mechanism which determines which samples are included in the calculation.
3. The KPI is calculated for the cycle based on the samples that passed through the filtering mechanism. For details, see ["Sample Rules: Calculating the KPI Based on Samples" below](#).
4. Aggregated KPI results are calculated based on the results for the cycle, and the previous aggregated results for the full calculation period. These aggregated KPI results are displayed in the Service Level Management reports. For details, see ["Sample Rules: Calculating the KPI's Aggregated Results" on the next page](#).

Sample Rules: Calculating the KPI Based on Samples

KPI results are calculated for each calculation cycle by the **calculateKPI** method, based on the **samples** parameter. The **samples** parameter is a **List** of **Sample** objects, which hold sample field values.

The Rules API classes are documented in Javadoc format in the *HP Rules API Reference*. These files are located in the following folder:

```
\\<Gateway Server root directory>\
AppServer\webapps\site.war\amdocs\leng\doc_lib\API_docs\Rules_API\index.html.
```

The **calculateKPI** method can be used to set KPI value, keys, and tooltips.

The value, keys, and tooltips set by the **calculateKPI** method are available to the **calculateAggregatedKPI** method, as described in ["Sample Rules: Calculating the KPI's Aggregated Results" on the next page](#).

Setting KPI Value

KPI value should be used when calculating KPI value directly from the sample fields. For details on setting the value, refer to the **setValue** method documentation.

For an example which uses KPI value, see ["Example - Sample-Based Maximum Response Time Rule" on page 77](#).

Setting KPI Calculation Keys

Each KPI can hold keys which are used as helpers to calculate value or tooltips. For example, when calculating average response time (total response time / total number of samples), the **setKey** method is used to set two keys for the current cycle: total response time, and total number of samples. These keys are aggregated and used to calculate the aggregated value of the KPI.

For an example which uses KPI calculation keys, see ["Example - Sample-Based Average Response Time Rule" on page 74](#).

Setting KPI Tooltips

KPI tooltips should be used when calculating KPI tooltips directly from the sample fields. For details on setting the tooltips, refer to the **setTooltip** method documentation.

Sample Rules: Calculating the KPI's Aggregated Results

Using the values, tooltips, and keys set by the **calculateKPI** method, the **calculateAggregatedKPI** method calculates aggregated values, keys, and tooltips.

The aggregated results are based on the KPI calculation results for the cycle, and the aggregated calculation results from the previous cycles.

The Rules API classes are documented in Javadoc format in the *HP Rules API Reference*. These files are located in the following folder:

```
\\<Gateway Server root directory>\
AppServer\webapps\site.war\amdocs\eng\doc_lib\API_docs\Rules_API\index.html.
```

The **calculateAggregatedKPI** method can be used to set KPI aggregated value, keys, and tooltips.

Setting KPI Aggregated Value

The KPI aggregated value can be calculated based on KPI value (for example, minimum calculation), or based on aggregated keys (for example, average response time). The KPI aggregated value is displayed in the Service Level Management report. For details on setting the aggregated value, refer to the **setAggregatedValue** method documentation.

For an example which calculates KPI aggregated value based on KPI value, see ["Example - Sample-Based Maximum Response Time Rule" on page 77](#).

For an example which calculates KPI aggregated value based on KPI aggregated keys, see ["Example - Sample-Based Average Response Time Rule" on page 74](#).

Setting KPI Aggregated Calculation Keys

The aggregated keys are used as helpers to calculate aggregated value or aggregated tooltips. The aggregated keys are calculated based on aggregated keys and keys from the current cycle.

For example, when calculating average response time (total response time / total number of samples), the **setAggregatedKey** method is used to set two aggregated keys: total response time, and total number of samples. These value are then used to calculate the aggregated KPI value.

For details on setting the aggregated keys, refer to the **setAggregatedKey** method documentation.

For an example which uses KPI aggregated calculation keys, see "[Example - Sample-Based Average Response Time Rule](#)" on page 74.

Setting KPI Status

If your rule sets KPI status, use the **setStatus** method within the **calculateAggregatedKPI** method.

Setting KPI Aggregated Tooltips

The KPI aggregated tooltips can be calculated based on KPI tooltips, or aggregated keys. The KPI aggregated tooltips are displayed in the Service Level Management report.

For details on setting the aggregated tooltips, refer to the **setAggregatedTooltip** method documentation.

When to Use Sample or Duration-Based Sample Rules

Use a Sample rule to calculate KPIs based on sample values and number of samples (when needed), without using sample duration in the calculation.

Use a Duration-based Sample rule to calculate KPIs based on sample values and sample duration, without using the number of samples in the calculation.

Example of Average Response Time Calculation

Average response time can be calculated using a Sample rule or a Duration-Based Sample rule.

At 10:00 a sample is received with 100 seconds as response time, and at 10:55 a sample is received with 50 seconds as response time. The rule calculation is for 10:00-11:00.

The API Sample rule calculates average response time (total response time / number of samples) = 75 seconds.

The API Duration-Based Sample rule calculates weighted response time based on the value of the samples, and on their durations. The duration of the first sample is 55 minutes and the duration of the second sample is 5 minutes. The weighted average response time for the period is $(55 \times 100 + 5 \times 50) / 60 = 95.83$ seconds.

API Sample Rule

A Sample rule calculates KPIs based on aggregated data taken from sample fields. For details on when to use the Sample rule, see "[When to Use Sample or Duration-Based Sample Rules](#)" above.

For details on how the rule is calculated, see "[Sample Rule Calculation Mechanism - Overview](#)" on page 57.

Sample Rule Methods and Fields

The Sample rule implements the Rules API Interface **SlmSamplesAggregatedCalculator**, which contains

the following methods:

```
public void calculateKPI(CI ci, SlmKPI kpi, List<Sample> samples)
public void calculateAggregatedKPI(CI ci, SlmKPI kpi)
public boolean isSampleValid(CI ci, SlmKPI kpi, Sample sample)
```

- The **calculateKPI** method calculates the KPI for each calculation cycle. This 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.
- The **kpi** parameter type is **SlmKPI**, and is used to set calculation results.
- The **samples** parameter is a **List** of **Sample** objects, which hold sample field values.
- The **calculateAggregatedKPI** method calculates the aggregated KPI. This method includes the parameters **ci** and **kpi**.
- The **isSampleValid** method is used to filter samples. If a sample is not valid, is not included in the calculation. Samples that are valid are included in the **samples** parameter of the **calculateKPI** method.
- 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.

The Rules API classes are documented in Javadoc format in the *HP Rules API Reference*. These files are located in the following folder:

```
\\<Gateway Server root directory>\
AppServer\webapps\site.war\amdocs\eng\doc_lib\API_docs\Rules_API\index.html.
```

For detailed examples of Sample rules, see ["Examples - API Sample Rule" on page 73](#).

API rules can be defined within the KPI Definition page or the Rule Repository, or using a text file template, as described in ["Creating Rules with the Rules API" on page 66](#).

Defining a Sample Rule in the KPI Definition Page or Rule Repository

To define a Sample rule using the KPI Definition page or within the Rule Repository, fill in the fields as follows:

- **Sample Fields.** List the sample fields whose values can be included in the calculation; 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.
- **Aggregated Calculation Script.** Enter the **calculateAggregatedKPI** method implementation; do not enter the method signature. The parameters **ci** and **kpi** of the **calculateAggregatedKPI** method are available for use in this script.
- **Sample Filter Script.** This field contains the default implementation of the **isSampleValid** method (by default, all samples are included in the calculation). You can edit this field to exclude samples from the calculation.

For detailed instructions, see ["How to Define an API Rule for a Specific KPI or Outage" on page 66](#) or ["How to Define an API Rule Within the Rule Repository" on page 70](#).

Defining a Sample Rule Using a Text File

To define a Sample rule using a text file template, use the **SlmSampleRuleTemplate.groovy** template file as described in ["How to Create a Text File-Based API Rule" on page 67](#).

Within the text file, define the **sampleFields** field, the **calculateKPI** method body, and the **calculateAggregatedKPI** method body.

The **isSampleValid** method has a default implementation of return true (all samples are included in the calculation). To override, uncomment the method and enter your implementation.

API Duration-Based Sample Rule

A Duration-Based Sample rule calculates KPIs based on aggregated data taken from sample fields. The duration-based rule uses each sample's duration within the rule calculation (for example, availability over time or backlog over time).

For details on when to use the Duration-Based Sample rule, see ["When to Use Sample or Duration-Based Sample Rules" on page 59](#). For details on how the rule is calculated, see ["Sample Rule Calculation Mechanism - Overview" on page 57](#).

Duration-Based Sample Rule Methods and Fields

The Duration-Based Sample rule implements the Rules API Interface **SlmSamplesTimeBasedAggregatedCalculator**, which contains the following methods:

```
public void calculateKPI(CI ci, SlmKPI kpi, List<Sample> samples)
public void calculateAggregatedKPI(CI ci, SlmKPI kpi)
public boolean isSampleValid(CI ci, SlmKPI kpi, Sample sample)
public boolean isSampleAndDurationValid(CI ci, SlmKPI kpi, Sample sample)
```

- The **calculateKPI** method calculates the KPI for each calculation cycle. This 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.
- The **kpi** parameter type is **SlmKPI**, and is used to set calculation results.
- The **samples** parameter is a **List** of **Sample** objects, which hold sample field values and sample durations. The **samples** parameter contains the samples that passed through a filter mechanism, as described in ["Filtering with the Duration-Based Sample Rule" on page 63](#). The last sample used in the previous calculation cycles can also be included, as described in ["Duration-Based Sample Continuity" on the next page](#).
- A sample's duration is defined as the interval from the sample timestamp to one of the following (whichever event occurs first):
 - The timestamp of the next sample, if the next sample is filtered using the **isSampleAndDurationValid** method.
 - The timestamp of the next sample within the cycle.
 - The end of the cycle.
- The **calculateAggregatedKPI** method calculates the aggregated KPI. This method includes the parameters **ci** and **kpi**.
- The **isSampleValid** and **isSampleAndDurationValid** method are used for filtering, as described in ["Filtering with the Duration-Based Sample Rule" on page 63](#).

The Rules API classes are documented in Javadoc format in the *HP Rules API Reference*. These files are located in the following folder:

\\<Gateway Server root directory>\

AppServer\webapps\site.war\amdocs\eng\doc_lib\API_docs\Rules_API\index.html.

- 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 Duration-Based Sample rules, see ["Examples - API Duration-Based Sample Rule" on page 78](#).

API rules can be defined using the KPI Definition page or the Rule Repository, or using a text file template, as described in ["Creating Rules with the Rules API" on page 66](#).

Defining a Duration-Based Sample Rule in the KPI Definition Page or Rule Repository

To define a Duration-Based Sample rule using the KPI Definition page or within the Rule Repository, fill in the fields as follows:

- **Sample Fields.** List the sample fields whose values can be included in the calculation; 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.
- **Aggregated Calculation Script.** Enter the **calculateAggregatedKPI** method implementation; do not enter the method signature. The parameters **ci** and **kpi** of the **calculateAggregatedKPI** method are available for use in this script.
- **Sample Filter Script** and **Sample and Duration Filter Script.** These fields contain the default implementation of the **isSampleValid** and **isSampleAndDurationValid** methods (by default, all samples are included in the calculation). You can edit these fields to exclude samples from the calculation.

For detailed instructions, see ["How to Define an API Rule for a Specific KPI or Outage" on page 66](#) or ["How to Define an API Rule Within the Rule Repository" on page 70](#).

Defining a Duration-Based Sample Rule Using a Text File

To define a Duration-Based Sample rule using a text file template, use the **SlmDurationBasedSampleRuleTemplate.groovy** template file as described in ["How to Create a Text File-Based API Rule" on page 67](#).

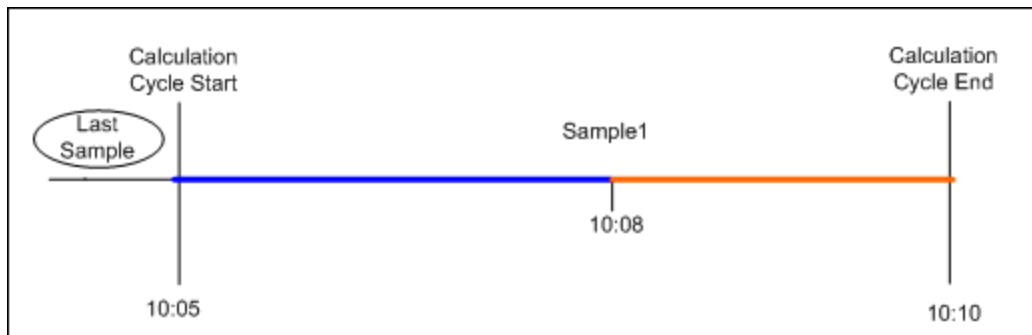
Within the text file, define the **sampleFields** field, the **calculateKPI** method body, and the **calculateAggregatedKPI** method body.

The **isSampleValid** and **isSampleAndDurationValid** methods have a default implementation of return true (all samples are included in the calculation). To override, uncomment the method and enter your implementation.

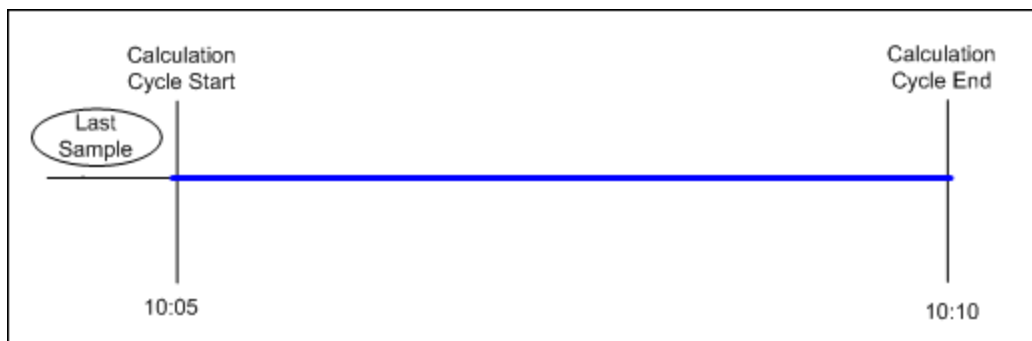
Duration-Based Sample Continuity

In the Duration-Based Sample rule, the first sample included in the **samples** parameter (for the **calculateKPI** method), is the last sample from the previous cycle's calculation. This enables a sample's value to be used over more than one calculation cycle.

For example, for the calculation cycle 10:05-10:10 there is one sample in the database (Sample1) with the timestamp 10:08. The **samples** parameter contains two samples: the last sample from the previous cycle (duration=3 minutes), and the sample from the current cycle (duration=2 minutes).



If there are no samples in the current cycle, the **samples** parameter contains the last sample from the previous cycle (duration=5 minutes).



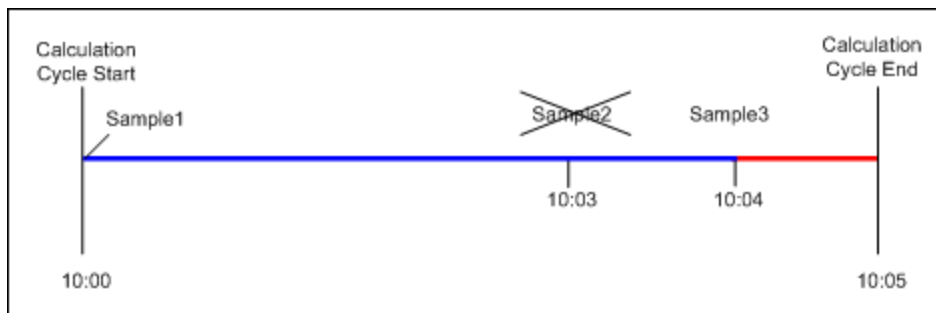
The last sample does not continue to the next calculation cycle if one of the following conditions are true:

- **No data timeout.** The last sample's timestamp has reached the no data timeout limit. For example, if the last sample's timestamp is 09:00 and the no data timeout is one hour, the sample is not included in the 10:00-10:05 calculation cycle, and all subsequent calculation cycles.
- **Downtime event.** A downtime event has occurred in the time between the last sample's timestamp and the current calculation cycle. For example, if the last sample's timestamp is 09:00 and a downtime is configured to 10:00-10:30, the sample is not part of the 10:30-10:35 calculation cycle, and all subsequent calculation cycles.

Filtering with the Duration-Based Sample Rule

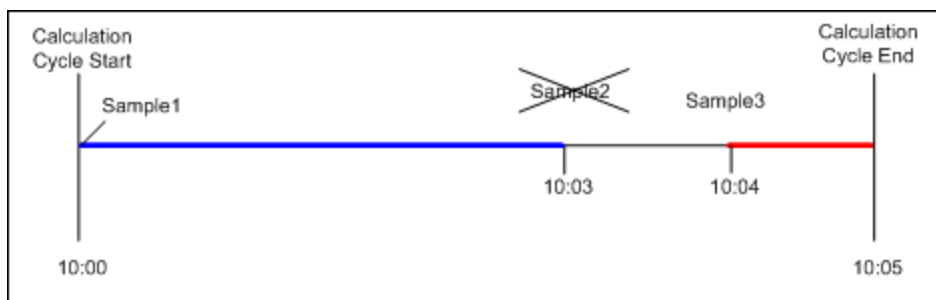
By default, the API Duration-Based Sample rule includes all samples in rule calculations. The **isSampleValid** and **isSampleAndDurationValid** methods enable filtering of samples when using this rule.

- When using the **isSampleValid** method, the duration of a filtered sample is added to the duration of the previous valid sample.



In the above example, Sample2 is filtered out using the **isSampleValid** method. The **samples** parameter contains Sample1 with 4 minutes duration and Sample3 with 1 minute duration. The total duration of all the samples in the cycle is 5 minutes.

- When using the **isSampleAndDurationValid** method, the duration of a filtered sample is not added to the duration of the previous valid sample, and it is therefore not included in the calculation.



In the above example, Sample2 is filtered out using the **isSampleAndDurationValid** method. The **samples** parameter contains Sample1 with 3 minutes duration and Sample3 with 1 minute duration. The total duration of all the samples in the cycle is 4 minutes.

API Outage by Samples Rule

An API Outage by Samples rule calculates outages based on data received from samples. An outage begins when a specific number of consecutive failures is calculated based on consecutive samples, and ends when a sample representing no failures is used in calculation.

Rule parameters (minimum number of failures, minimum duration, and maximum duration) define when an outage begins and when it ends, as described in List of Service Level Management Business Rule Parameters in the APM Application Administration Guide. The Outage by Samples rule parameters are demonstrated in ["Examples - API Outage by Samples Rule" on page 84](#).

Each sample can represent a number of failures (zero or more). Using the API Outage by Samples rule, you can calculate how many failures a sample represents based on the sample values. The outage is then calculated using the outage rule parameters.

Outage by Samples Rule Methods and Fields

The Outage by Samples rule implements the Rules API Interface **OutageBySamplesCalculator**, which contains the following methods:

```
public void calculateOutage(Outage outage, Sample sample)
```



```
public boolean isSampleValid(Sample sample)
```

- The **calculateOutage** method calculates the number of failures represented by a **Sample**. This method includes the parameters **outage** and **sample**.
- The **outage** parameter type is **Outage**, and is used to set the number of failures for the given **sample**.
- The **sample** parameter type is **Sample**, which holds sample field values.
- The **isSampleValid** method is used to filter samples. If a sample is not valid, it is not included in the **calculateOutage** method calculation.

The Rules API classes are documented in Javadoc format in the *HP Rules API Reference*. These files are located in the following folder:

```
\\<Gateway Server root directory>\
```

```
AppServer\webapps\site.war\amddocs\eng\doc_lib\API_docs\Rules_API\index.html.
```

- 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 examples of the Outage by Samples rule, see ["Examples - API Outage by Samples Rule" on page 84](#).

API rules can be defined using the KPI Definition page or the Rule Repository, or using a text file template, as described in ["Creating Rules with the Rules API" on the next page](#).

Defining an Outage by Samples Rule in the KPI Definition Page or Rule Repository

To define an Outage by Samples rule using the KPI Definition page or within the Rule Repository, fill in the fields as follows:

- **Sample Fields.** List the sample fields whose values can be included in the calculation; separate between the sample names with a comma (for example: "u_iStatus").
- **Outage Calculation Script.** Enter the **calculateOutage** method implementation; do not enter the method signature. The parameters **outage** and **sample** of the **calculateOutage** method are available for use in this script.
- **Sample Filter Script.** This field contains the default implementation of the **isSampleValid** method (by default, all samples are included in the calculation). You can edit this field to exclude samples from the calculation.

For detailed instructions, see ["How to Define an API Rule for a Specific KPI or Outage" on the next page](#) or ["How to Define an API Rule Within the Rule Repository" on page 70](#).

Defining an Outage by Samples Rule Using a Text File

To define an Outage by Samples rule using a text file template, use the **SlmOutageBySampleTemplate.groovy** template file as described in ["How to Create a Text File-Based API Rule" on page 67](#).

Within the text file, define the **sampleFields** field and the **calculateOutage** method body.

The **isSampleValid** method has a default implementation of return true (all samples are included in the calculation). To override, uncomment the method and enter your implementation.

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.

Note: The following three ways are also applicable to defining Outage calculations, using the API Outage by Samples Rule.

Define a rule for a specific KPI

Service Level Management KPIs can have the following API rules: API Group and Sibling Rule, API Sample Rule, and API Duration-Based Sample Rule.

Using the **New SLA/Edit SLA Wizard > Configure SLA Indicators** page, you can assign one of the API rules to a KPI, and enter rule details to define rule logic for that KPI.

You can then edit the rule details in the **Configure SLA Indicators** page at any time to change the rule logic for the KPI.

For details, see ["How to Define an API Rule for a Specific KPI or Outage" below](#).

Create a rule for multiple KPIs using a text file

For each of the API rules 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 Level Management, 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 restart the offline engine.

For details, see ["How to Create a Text File-Based API Rule" on the next page](#).

Defining a rule within the rule repository

The Rule Repository contains the following API rules: API Group and Sibling Rule, API Sample Rule, and API Duration-Based Sample Rule. You can use the Rule Repository to clone an API rule and enter rule details to define the rule logic.

After the rule is applied to a KPI, you can edit rule details within the **Configure SLA Indicators** page at any time to change the rule logic for a specific KPI.

For details, see ["How to Define an API Rule Within the Rule Repository" on page 70](#).

How to Define an API Rule for a Specific KPI or Outage

Each KPI or Outage has applicable API rules. Using the KPI Definition page, assign one of the API rules to a KPI or Outage, and enter the rule details to define the rule logic.

1. **Assign an API rule to a KPI or outage**

- To assign an API rule for a specific KPI assigned to a CI, edit an SLA using the **Agreements Manager > New SLA/Edit SLA Wizard > Configure SLA Indicators** page. Select **Add KPI** to assign a new KPI to the CI, or modify an existing KPI. For details, see Add KPI for CI/Edit KPI for CI Dialog Box in the APM Application 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. (API Sample Rule and API Duration-Based Sample Rule are only applicable for monitor CIs.) For a description of the rule types see "[SLM Rules API](#)" on page 53.

- To assign an API Outage rule to a CI's Outage, edit an SLA using the **Agreements Manager > New SLA/Edit SLA Wizard > Configure SLA Indicators** page. For details, see Add Outages KPI/Edit Outages KPI Dialog Box in the APM Application Administration Guide.

Click the Edit button to edit the Outage. From the list of applicable business rules, select API Outage by Samples Rule. (This rule is only applicable for monitored CIs.)

2. **Define the KPI or outage 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 55
- "[API Sample Rule](#)" on page 59
- "[API Duration-Based Sample Rule](#)" on page 61
- "[API Outage by Samples Rule](#)" on page 64

How to Create a Text File-Based API Rule

There are rule template files corresponding to the 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 Level Management, 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 restart the offline engine.

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 55
- "[API Sample Rule](#)" on page 59
- "[API Duration-Based Sample Rule](#)" on page 61
- "[API Outage by Samples Rule](#)" on page 64

Save the file to the **<Data Processing server root directory>\BLE\rules\groovy\rules** directory.

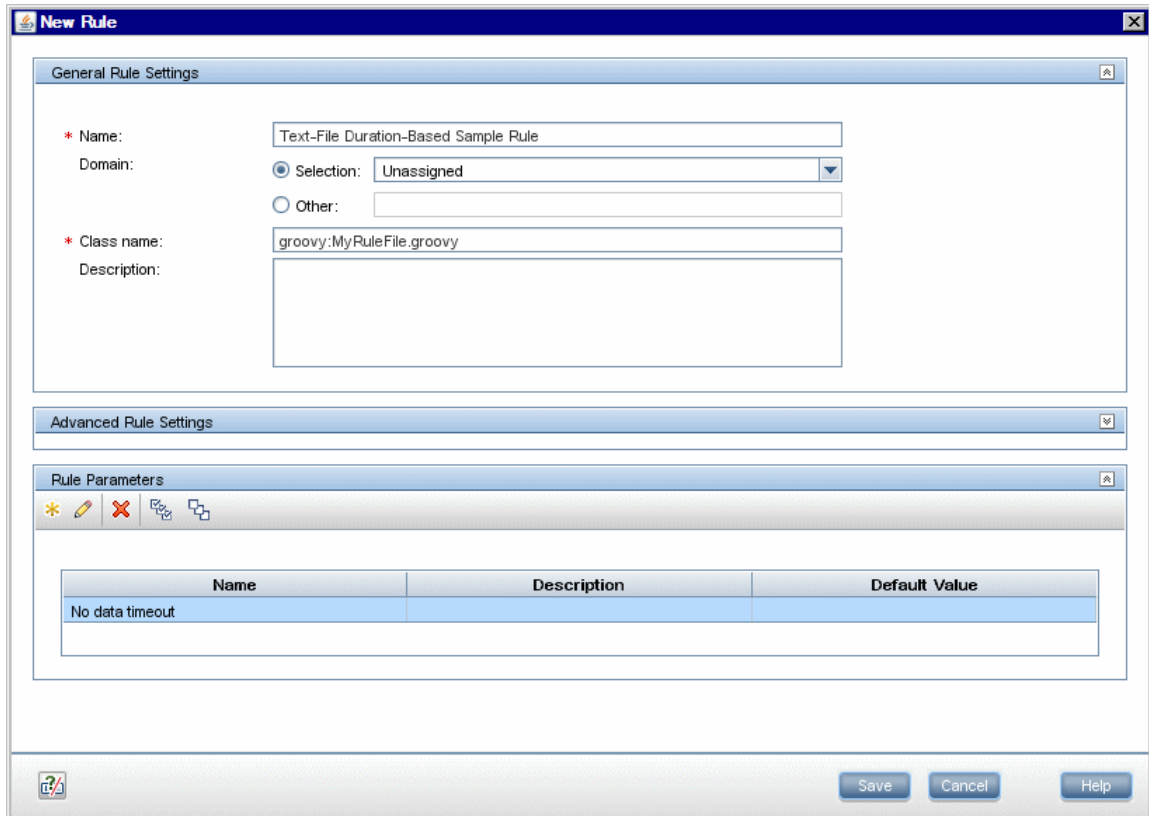
You must now add a rule in the Rule Repository that uses the rule logic in the text file.

- For rules other than Outage, follow the instructions in "Add a rule in the rule repository (for Group and Sibling, Sample, or Duration-Based Sample rules)" below.
- For Outage rules, follow the instructions in "Add an Outage rule in the rule repository" on the next page.

2. ***Add a rule in the rule repository (for Group and Sibling, Sample, or Duration-Based Sample rules)***

- a. Select **Admin > Service Level Management > Repositories > Business Rules > New Rule**. For details on adding rules, see Customizing KPI and HI Calculation Rules in the APM Application Administration Guide.
- b. In the **Name** field, type the name of the rule you want to create (mandatory).
- c. 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.
- d. For API Duration-Based Sample rules, create the following rule parameter:
 - In the **Rule parameters** area, click **New**.
 - In the **Name** field type **No Data Timeout**. In the **Type** field, select **Long**. In the **Default Value** field, type **3600**.
- e. Click **OK** to save.

The following image shows a Duration-Based Sample rule after the rule parameter has been added:



3. **Add an Outage rule in the rule repository**

- a. Within **Admin> Service Level Management > Repositories > Business Rules**, select the **Synthetic WS Outage** rule (316) and clone the rule.
- b. In the **Name** field, change the name of the rule (mandatory).
- c. 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.
- d. You can edit the **Description**, but do not change any other fields.
- e. Click OK to save.

4. **Add the rule to the list of applicable rules for a KPI or Outage**

Add the new rule to the list of applicable rules already attached to the relevant KPI. The relevant KPI for Outage rules is the **Outages** KPI (200).

For details, see the Main Settings Area > Applicable Rules parameter in New KPI/Edit KPI Dialog Box in the APM Application Administration Guide.

5. **Restart the offline engine 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, enter the following:

`http://<Application Performance Management Data Processing server name>:11021`

- b. Select **Foundations:NannyManager**.
- c. Invoke **showServiceInfoAsHTML**, and restart the offline engine.

How to Define an API Rule Within the Rule Repository

Within the Business Rule Repository, create an API rule that can be applied to multiple KPIs or Outages. This is done by cloning one of the four API rules, and setting default rule values for specific rule parameters. After the rule is applied to a KPI or Outage, you can edit its script within the KPI Definition page at any time to change the rule logic for the specific KPI or Outage.

1. **Clone an API rule**

Select **Admin > Service Level Management > Repositories > Business Rules**. In the Business Rule Repository page, clone one of the following rules: API Group and Sibling Rule, API Sample Rule, API Duration-Based Sample Rule, or API Outage by Samples Rule.

For details on cloning a rule, see *How to Customize a Business Rule Template in the APM Application Administration Guide*.

2. **Edit rule details**

- a. Open the new rule for editing.
- b. In the **Name** field, rename the cloned rule.
- c. Within the **Rule Parameters**, set the **Default value** for each rule parameter defining your rule logic, as described in the following sections:
 - o ["API Group and Sibling Rule" on page 55](#)
 - o ["API Sample Rule" on page 59](#)
 - o ["API Duration-Based Sample Rule" on page 61](#)
 - o ["API Outage by Samples Rule" on page 64](#)

For example, to define the **KPI Calculation Script**, edit the **KPI Calculation Script** rule parameter. In the **Default Value** field, enter the rule calculation script. The code that you enter becomes the default code for this rule, and appears in the KPI Definition page for all KPIs assigned this rule. (Do not change any other fields.)

3. **Add the rule to the list of applicable rules for a KPI or Outage**

Add the new rule to the list of applicable rules already attached to the relevant KPI. The relevant KPI for Outage rules is the **Outages** KPI (200).

For details, see the *Applicable Rules GUI parameter in New KPI/Edit KPI Dialog Box in the APM Application Administration Guide*.

How to Work with Tooltip Entries

If you have used the **kpi.setTooltip** method, you must set a corresponding tooltip entry in the Infrastructure Settings.

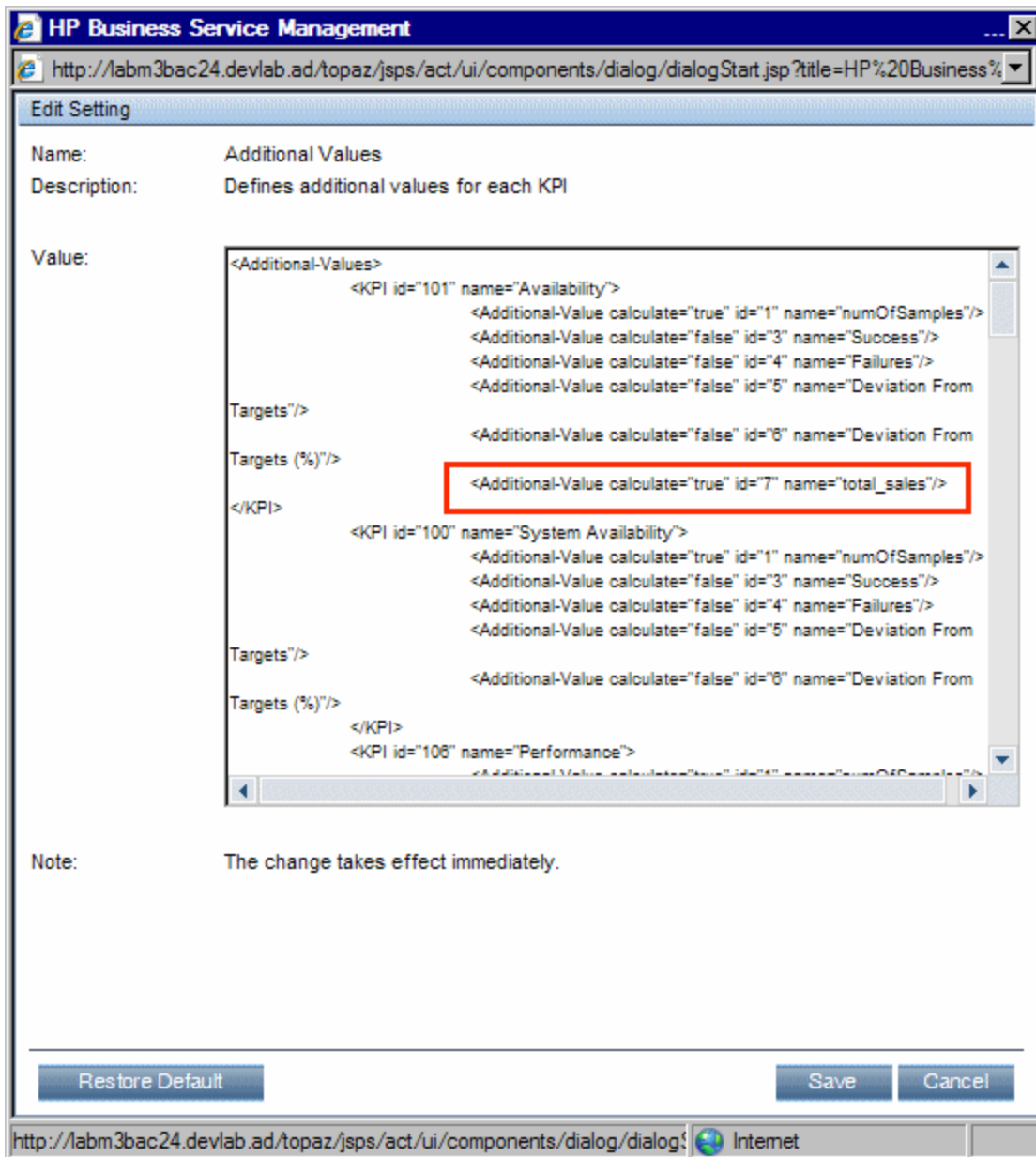
1. Select **Admin > Platform > Setup and Maintenance > Infrastructure Settings > Service Level Management**.

2. Edit the **Additional Values** as follows:

- a. Scroll down to the KPI where you want to add the tooltip.
- b. After the last entry for the KPI, add a line in the following format:
`<Additional-Value calculate="true" id="<id>" name="<name>"/>`

In the above format, `<id>` is one number higher than the current highest ID (if the highest ID is 5, type 6), and `<name>` is the name of the tooltip exactly as used in the code.

For example, if your code contains the method invocation `kpi.setTooltip("total_sales", value)`, and the KPI is **Availability**, type the following:



- c. Click **Save**.

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 **<Data Processing server root directory>\HPBSM\log\offline_engine\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 and API Duration-Based Sample rules)
- **OutageRule.log** (for API Outage by Samples 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 **<Data Processing server root directory>HPBSM\conf\core\Tools\log4j\offline_engine\slm_rules.properties**.

2. In the line corresponding with your rule type, replace the string **\${loglevel}** with the severity level you want logged (type one of the following: **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.bac.slm.rules.group.common.SlmGroupAndSiblingRule = ${loglevel}, slm.rules.api.group.appender`
- Sample rules and Duration-Based Sample rules:
`log4j.category.com.mercury.am.bac.slm.rules.leaf.simplified.SlmSimplifiedLeafRule = ${loglevel}, slm.rules.api.sample.appender`
- Outage by Samples rules:


```
log4j.category.com.mercury.am.bac.slm.rules.outages.simplified.SimplifiedOutageRule =  
${loglevel}, slm.rules.api.outage.appender
```

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. SLM rules can only access CI properties which have with the qualifier **BLE_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>
```

Examples - API Group and Sibling Rule

Both Service Level Management and Service Health implement the same interface to calculate API Group and Sibling rules.

For detailed examples showing API Group and Sibling rules, see ["Examples - API Group and Sibling Rule" on page 26](#).

Note that the statuses used in the **Status** class are different in Service Level Management and Service Health. For example, `Status.OK` in Service Health code is equivalent to `Status.EXCEEDED` in Service Level Management code. The following table shows the parallel statuses:

Service Health Status	Service Level Management Status
OK	EXCEEDED
WARNING	MET
MINOR	MINOR_BREACHED
MAJOR	BREACHED
CRITICAL	FAILED

Examples - API Sample Rule

This section provides examples of API Sample Rules. The following examples are described:

- ["Example - Sample-Based Average Response Time Rule" below](#)
- ["Example - Sample-Based Average Response Time Rule with Filter" on page 76](#)
- ["Example - Sample-Based Maximum Response Time Rule" on page 77](#)

Example - Sample-Based Average Response Time Rule

The following rule calculates average response time, based on the `dResponseTime` sample field. The rule result (aggregated value) is the average response time calculated based on all the samples that exist for the calculation period.

The rule logic is (total response time in seconds / total number of samples).

The rule uses the **SlmKPI** keys and aggregated keys to aggregate the total response time and the total number of samples, to calculate the rule result.

```
// Define the sample fields that will be used in the rule calculation.
def sampleFields = ["dResponseTime"];

/*
 * Implementation of the SlmSamplesAggregatedCalculator interface method.
 * For more information refer to the Rules API documentation.
 */
public void calculateKPI(CI ci, SlmKPI kpi, List<Sample> samples) {
/*
 * Sum the field dResponseTime from all given samples. This represents the total
 * response time for all the samples in the current calculation cycle.
 */
def totalTime = Utils.sumField(samples, "dResponseTime");
/*
 * Keep the total response time, converted to seconds, on a kpi key named
 * totalResponseTime. This key is used by the calculateAggregatedKPI method.
 */
kpi.key.totalResponseTime = Utils.divide(totalTime,1000.0);
/*
 * Keep the number of samples for the current calculation cycle on a kpi key named
 * totalSamples. This key is used by the calculateAggregatedKPI method.
 */
kpi.key.totalSamples = samples.size();
}

/*
 * Implementation of the SlmSamplesAggregatedCalculator interface method.
 */
public void calculateAggregatedKPI(CI ci, SlmKPI kpi) {
/*
 * Keep the aggregated response time on a kpi aggregated key named
 * totalResponseTime, by adding the current aggregated totalResponseTime from the
 * kpi aggregated key, and the current calculation cycle response time taken from the
 * kpi key named totalResponseTime (calculated in the calculateKPI method).
 */
}
```

```

kpi.agggregatedKey.totalResponseTime = Utils.sum(kpi.key.totalResponseTime,
kpi.agggregatedKey.totalResponseTime).
/*
 * Keep the aggregated total samples on a kpi aggregated key named totalSamples,
 * by adding the current aggregated total samples from the kpi aggregated key, and
 * the current calculation cycle total samples, taken from the kpi key named
 * totalSamples (calculated in the calculateKPI method).
 */
kpi.agggregatedKey.totalSamples = Utils.sum(kpi.key.totalSamples,
kpi.agggregatedKey.totalSamples)
/*
 * Set the aggregated value of the KPI by dividing the two aggregated values.
 * This aggregated value will be displayed in the SLA reports.
 */
kpi.agggregatedValue = Utils.divide(kpi.agggregatedKey.totalResponseTime,
kpi.agggregatedKey.totalSamples)
}

```

Calculation - Sample-Based Average Response Time Rule

The following calculation illustrates the Sample-Based Average Response Time rule. Between 10:00 and 11:00, 9 samples have arrived:

```

10:00 {Sample Fields: dResponseTime = 60}
10:04 {Sample Fields: dResponseTime = 30}
10:11 {Sample Fields: dResponseTime = 30}
10:25 {Sample Fields: dResponseTime = 25}
10:27 {Sample Fields: dResponseTime = 35}
10:35 {Sample Fields: dResponseTime = 10}
10:36 {Sample Fields: dResponseTime = 20}
10:38 {Sample Fields: dResponseTime = 30}
10:52 {Sample Fields: dResponseTime = 75}

```

Each calculation cycle is 5 minutes long. The rule calculation is as follows:

Cycle	Keys set by calculateKPI		Aggregated keys and value set by calculateAggregatedKPI		
	totalResponseTime	totalSamples	totalResponseTime	totalSamples	Aggregated Value
10:00-10:05	90	2	90	2	45
10:05-10:10	Null	0	90	2	45
10:10-10:15	30	1	120	3	40

Cycle	Keys set by calculateKPI		Aggregated keys and value set by calculateAggregatedKPI		
10:15-10:20	Null	0	120	3	40
10:20-10:25	Null	0	120	3	40
10:25-10:30	60	2	180	5	36
10:30-10:35	Null	0	180	5	36
10:35-10:40	60	3	240	8	30
10:40-10:45	Null	0	240	8	30
10:45-10:50	Null	0	240	8	30
10:50-10:55	75	1	315	9	35
10:55-11:00	null	0	315	9	35
					Final result: 35

Example - Sample-Based Average Response Time Rule with Filter

This rule is the same as the previous rule ("[Example - Sample-Based Average Response Time Rule](#)" on page 74), with the addition of a sample filter.

The code for the sample fields is as follows:

```
// This rule uses the dResponseTime sample field and u_iStatus sample field.
def sampleFields = ["dResponseTime", "u_iStatus"];
```

This rule uses an additional method, as follows:

```
/*
 * Override the default implementation of the SlmSamplesAggregatedCalculator
 * interface method.
 */
```

```
* Filter samples that hold u_iStatus field value which is not 0 or 2.
*/
public boolean isSampleValid(Sample sample) {
    //Get the value of the sample's u_iStatus field.
    def avialFieldValueObj = sample.u_iStatus;
    return (avialFieldValueObj == 0 || avialFieldValueObj == 2)
}
```

Calculation - Sample-Based Average Response Time Rule with Filter

The following calculation illustrates the Sample-Based Average Response Time rule with filter.

For 10:00 - 11:00, 9 samples exist on the Profile Database:

```
10:00 {Sample Fields: dResponseTime = 60, u_iStatus = 0}
10:04 {Sample Fields: dResponseTime = 30, u_iStatus = 1}
10:11 {Sample Fields: dResponseTime = 30, u_iStatus = 2}
10:25 {Sample Fields: dResponseTime = 25, u_iStatus = 1}
10:27 {Sample Fields: dResponseTime = 35, u_iStatus = 0}
10:35 {Sample Fields: dResponseTime = 10, u_iStatus = 1}
10:36 {Sample Fields: dResponseTime = 20, u_iStatus = 0}
10:38 {Sample Fields: dResponseTime = 30, u_iStatus = 2}
10:52 {Sample Fields: dResponseTime = 75, u_iStatus = 1}
```

The samples in bold do not pass the filter. The following 5 samples are taken into calculation:

```
10:00 {Sample Fields: dResponseTime = 60, u_iStatus = 0}
10:11 {Sample Fields: dResponseTime = 30, u_iStatus = 2}
10:27 {Sample Fields: dResponseTime = 35, u_iStatus = 0}
10:36 {Sample Fields: dResponseTime = 20, u_iStatus = 0}
10:38 {Sample Fields: dResponseTime = 30, u_iStatus = 2}
```

The calculation result is = $(60+30+35+20+30)/5 = 35$

Example - Sample-Based Maximum Response Time Rule

The following rule calculates maximum response time, based on the dResponseTime sample field. The rule result (aggregated value) is the maximum response time calculated based on all the samples that exist for the calculation period. The rule uses the **SlmKPI** value to keep the maximum value for each calculation cycle.

```
// Define the sample fields that will be used in the rule calculation.
def sampleFields = ["dResponseTime"];

/*
 * Implementation of the SlmSamplesAggregatedCalculator interface method. For more
 information refer to the Rules API documentation.
 */
public void calculateKPI(CI ci, SlmKPI kpi, List<Sample> samples) {
    /**
```

```
        * Find the maximum value of the dResponseTime field from all given samples,  
        * and set it as the KPI value for the current calculation cycle.  
        */  
        kpi.value = Utils.getMax(samples, "dResponseTime");  
    }  
  
    /**  
    * Implementation of the SlmSamplesAggregatedCalculator interface method.  
    */  
    public void calculateAggregatedKPI(CI ci, SlmKPI kpi) {  
        /**  
        * Keep the aggregated maximum response time on the KPI aggregated value,  
        * by replacing the current aggregated value with the maximum between the KPI  
        * aggregated value and the KPI value (calculated in the calculateKPI method).  
        * This aggregated value will be displayed in the SLA reports.  
        */  
        kpi.aggregatedValue = Utils.max(kpi.value, kpi.aggregatedValue)  
    }  
}
```

Calculation - Sample-Based Maximum Response Time Rule

The following calculation illustrates the Sample-Based Maximum Response Time rule.

Between 10:00 and 11:00, 9 samples have arrived:

```
10:00 {Sample Fields: dResponseTime = 60}  
10:04 {Sample Fields: dResponseTime = 30}  
10:11 {Sample Fields: dResponseTime = 30}  
10:25 {Sample Fields: dResponseTime = 25}  
10:27 {Sample Fields: dResponseTime = 35}  
10:35 {Sample Fields: dResponseTime = 10}  
10:36 {Sample Fields: dResponseTime = 20}  
10:38 {Sample Fields: dResponseTime = 30}  
10:52 {Sample Fields: dResponseTime = 75}
```

The result of the rule calculation is 75.

Examples - API Duration-Based Sample Rule

This section provides examples of API Duration-Based Sample Rules. The following examples are described:

- ["Example - Duration-Based Average Response Time Rule" on the next page](#)
- ["Example - Duration-Based Average Response Time Rule with isSampleValid Method Filter" on page 81](#)
- ["Example - Duration-Based Average Response Time Rule with isSampleAndDurationValid Method Filter" on page 82](#)
- ["Example - Duration-Based Average Response Time Rule with isSampleAndDurationValid and isSampleValid Method Filters" on page 83](#)

Example - Duration-Based Average Response Time Rule

The following rule calculates the weighted average response time based on the `dResponseTime` sample field and sample duration, for all the samples that exist for the calculation period.

The rule logic is $\text{sum}(\text{sample response time} * \text{sample duration}) / \text{sum}(\text{samples duration})$.

The rule uses **SlmKPI** keys and aggregated keys to aggregate the total weighted response time and the total duration of samples, in order to calculate rule results.

```
// Define the sample fields that will be used in the rule calculation.
def sampleFields = ["dResponseTime"];

/*
 * Implementation of the SlmSamplesTimeBasedAggregatedCalculator interface method.
 * For more information refer to the Rules API documentation.
 */
public void calculateKPI(CI ci, SlmKPI kpi, List<Sample> samples) {
    /**
     * Set the KPI key totalDuration to sum duration of all given samples.
     * This key is used by the calculateAggregatedKPI method.
     */
    kpi.key.totalDuration = Utils.sumDuration(samples)
    // Iterate over all samples that arrived in the current calculation cycle.
    samples.each {Sample sample ->
        /**
         * Calculate weighted response time for each sample by multiplying
         * sample duration with sample dResponseTime field value.
         */
        def weightedResponseTime = Utils.multiply(sample.duration,
sample.dResponseTime);
        /**
         * Keep the total weighted response time for all given samples
         * on a KPI key named totalResponseTime.
         * This key is used by the calculateAggregatedKPI method.
         */
        kpi.key.totalResponseTime = Utils.sum(kpi.key.totalResponseTime,
weightedResponseTime)
    }
}

/*
 * Implementation of the SlmSamplesTimeBasedAggregatedCalculator interface method.
 */
public void calculateAggregatedKPI(CI ci, SlmKPI kpi) {
    /**
     * Keep the aggregated response time on a kpi aggregated key named
     * totalResponseTime, by adding the current aggregated totalResponseTime, and the
     * current calculation cycle response time taken from the totalResponseTime kpi
     key
```

```

        * (calculated by the calculateKPI method).
        */
        kpi.aggregatedKey.totalResponseTime = Utils.sum(kpi.key.totalResponseTime,
        kpi.aggregatedKey.totalResponseTime)
        /**
        * Keep the aggregated total duration on a kpi aggregated key named
        totalDuration,
        * by adding the current aggregated total duration from the kpi aggregated key,
        * and the current calculation cycle total duration.
        */
        kpi.aggregatedKey.totalDuration = Utils.sum(kpi.key.totalDuration,
        kpi.aggregatedKey.totalDuration)
        /**
        * Set the aggregated value of the KPI by dividing the two aggregated values.
        * This aggregated value will be displayed in the SLA reports.
        */
        kpi.aggregatedValue = Utils.divide(kpi.aggregatedKey.totalResponseTime,
        kpi.aggregatedKey.totalDuration)
    }
    
```

Calculation - Duration-Based Average Response Time Rule

The following calculation illustrates the Duration-Based Average Response Time rule.

For 10:00 - 11:00, 5 samples exist on the Profile Database:

- 10:00 Sample1 {Sample Fields: dResponseTime = 60}
- 10:04 Sample2 {Sample Fields: dResponseTime = 30}
- 10:25 Sample3 {Sample Fields: dResponseTime = 25}
- 10:38 Sample4 {Sample Fields: dResponseTime = 30}
- 10:52 Sample5 {Sample Fields: dResponseTime = 75}

Each calculation cycle is 5 minutes long. The rule calculation is as follows:

Cycle	samples parameter		Keys set by calculateKPI		Aggregated keys and value set by calculateAggregatedKPI		
	Sample	Sample Duration	totalResponseTime	totalDuration	totalResponseTime	totalDuration	Aggregated Value
10:00-10:05	Sample1 Sample2	240 60	16200	300	16200	300	54.000
10:05-10:10	Sample2	300	9000	300	25200	600	42.000
10:10-10:15	Sample2	300	9000	300	34200	900	38.000
10:15-10:20	Sample2	300	9000	300	43200	1200	36.000
10:20-10:25	Sample2	300	9000	300	52200	1500	34.800
10:25-10:30	Sample3	300	7500	300	59700	1800	33.167
10:30-10:35	Sample3	300	7500	300	67200	2100	32.000
10:35-10:40	Sample3 Sample4	180 120	8100	300	75300	2400	31.375
10:40-10:45	Sample4	300	9000	300	84300	2700	31.222
10:45-10:50	Sample4	300	9000	300	93300	3000	31.100
10:50-10:55	Sample4 Sample5	120 180	17100	300	110400	3300	33.455
10:55-11:00	Sample5	300	22500	300	132900	3600	36.917
							Result: 36.917

Example - Duration-Based Average Response Time Rule with isSampleValid Method Filter

This rule is the same as ["Example - Duration-Based Average Response Time Rule"](#) on page 79, with the addition of the **isSampleValid** method filter.

The code for the sample fields is as follows:

```
// This rule uses the dResponseTime sample field and u_iStatus sample field.  
def sampleFields = ["dResponseTime", "u_iStatus"];
```

This rule uses an additional method, as follows:

```
/*  
 * Override default implementation of the SlmSamplesTimeBasedAggregatedCalculator  
 * interface method.  
 *  
 * Filter samples that hold u_iStatus field with value of 6.  
 */  
public boolean isSampleValid(Sample sample) {  
    //Get the value of the sample's u_iStatus field.  
    def avialFieldValueObj = sample.u_iStatus;  
    return (avialFieldValueObj != 6)  
}
```

Calculation - Duration-Based Average Response Time Rule with isSampleValid Method Filter

The following calculation illustrates the Duration-Based Average Response Time rule with **isSampleValid** method filter.

For 10:00 - 11:00, 5 samples exist on the Profile Database:

```
10:00 Sample1 {Sample Fields: dResponseTime = 60, u_iStatus = 0}  
10:04 Sample2 {Sample Fields: dResponseTime = 30, u_iStatus = 2}  
10:25 Sample3 {Sample Fields: dResponseTime = 25, u_iStatus = 6}  
10:38 Sample4 {Sample Fields: dResponseTime = 30, u_iStatus = 0}  
10:52 Sample5 {Sample Fields: dResponseTime = 75, u_iStatus = 2}
```

Sample3 did not pass the **isSampleValid** method filter, so the following 4 samples are taken into calculation:

```
10:00 Sample1 {Sample Fields: dResponseTime = 60, u_iStatus = 0}  
10:04 Sample2 {Sample Fields: dResponseTime = 30, u_iStatus = 2}  
10:38 Sample4 {Sample Fields: dResponseTime = 30, u_iStatus = 0}  
10:52 Sample5 {Sample Fields: dResponseTime = 75, u_iStatus = 2}
```

Note that after the filtering, the interval between Sample3 and Sample4 is considered part of the duration of Sample2.

The rule calculation is as follows:

Cycle	samples parameter		Keys set by calculateKPI		Aggregated keys and value set by calculateAggregatedKPI		
	Sample	Sample Duration	totalResponseTime	totalDuration	totalResponseTime	totalDuration	Aggregated Value
10:00-10:05	Sample1	240	16200	300	16200	300	54.000
	Sample2	60					
10:05-10:10	Sample2	300	9000	300	25200	600	42.000
10:10-10:15	Sample2	300	9000	300	34200	900	38.000
10:15-10:20	Sample2	300	9000	300	43200	1200	36.000
10:20-10:25	Sample2	300	9000	300	52200	1500	34.800
10:25-10:30	Sample2	300	9000	300	61200	1800	34
10:30-10:35	Sample2	300	9000	300	70200	2100	33.428
10:35-10:40	Sample2	180	9000	300	79200	2400	33
	Sample4	120					
10:40-10:45	Sample4	300	9000	300	88200	2700	32.666
10:45-10:50	Sample4	300	9000	300	97200	3000	32.4
10:50-10:55	Sample4	120	17100	300	114300	3300	34.636
	Sample5	180					
10:55-11:00	Sample5	300	22500	300	136800	3600	38
							Result: 38

Example - Duration-Based Average Response Time Rule with isSampleAndDurationValid Method Filter

This rule is the same as "Example - Duration-Based Average Response Time Rule" on page 79, but with the `isSampleAndDurationValid` method filter.

The code for the sample fields is as follows:

```
// This rule uses the dResponseTime sample field and u_iStatus sample field.
def sampleFields = ["dResponseTime", "u_iStatus"];
```

This rule uses an additional method, as follows:

```
/*
 * Override default implementation of the SlmSamplesTimeBasedAggregatedCalculator
 * interface method.
 *
 * Filter samples that hold u_iStatus field value which is not 0 or 2.
 */
public boolean isSampleAndDurationValid(CI ci, SlmKPI kpi, Sample sample) {
    //Get the value of the sample's u_iStatus field.
    def avialFieldValueObj = sample.u_iStatus;
    return (avialFieldValueObj == 0 || avialFieldValueObj == 2)
}
```

Calculation - Duration-Based Average Response Time Rule with isSampleAndDurationValid Method Filter

The following calculation illustrates the Duration-Based Average Response Time rule with `isSampleAndDurationValid` method filter.

For 10:00 - 11:00, 5 samples exist on the Profile Database:

10:00 Sample1 {Sample Fields: dResponseTime = 60, u_iStatus = 0}
 10:04 Sample2 {Sample Fields: dResponseTime = 30, u_iStatus = 2}
 10:25 Sample3 {Sample Fields: dResponseTime = 25, u_iStatus = 2}
10:38 Sample4 {Sample Fields: dResponseTime = 30, u_iStatus = 1}
 10:52 Sample5 {Sample Fields: dResponseTime = 75, u_iStatus = 2}

Sample4 did not pass the **isSampleAndDurationValid** method filter, so the following 4 samples are taken into calculation:

10:00 Sample1 {Sample Fields: dResponseTime = 60, u_iStatus = 0}
 10:04 Sample2 {Sample Fields: dResponseTime = 30, u_iStatus = 2}
 10:25 Sample3 {Sample Fields: dResponseTime = 25, u_iStatus = 2}
 10:52 Sample5 {Sample Fields: dResponseTime = 75, u_iStatus = 2}

Note that after the filtering, the interval between Sample4 and Sample5 is *not* considered part of the duration of Sample3, so the total duration for the hour is 46 minutes.

The rule calculation is as follows:

Cycle	samples parameter		Keys set by calculateKPI		Aggregated keys and value set by calculateAggregatedKPI		
	Sample	Sample Duration	totalResponseTime	totalDuration	totalResponseTime	totalDuration	Aggregated Value
10:00-10:05	Sample1	240	16200	300	16200	300	54.000
	Sample2	60					
10:05-10:10	Sample2	300	9000	300	25200	600	42.000
10:10-10:15	Sample2	300	9000	300	34200	900	38.000
10:15-10:20	Sample2	300	9000	300	43200	1200	36.000
10:20-10:25	Sample2	300	9000	300	52200	1500	34.800
10:25-10:30	Sample3	300	7500	300	59700	1800	33.167
10:30-10:35	Sample3	300	7500	300	67200	2100	32.000
10:35-10:40	Sample3	180	4500	180	71700	2280	31.477
10:40-10:45	empty			0	71700	2280	31.477
10:45-10:50	empty			0	71700	2280	31.477
10:50-10:55	Sample5	180	13500	180	85200	2460	34.634
10:55-11:00	Sample5	300	22500	300	107700	2760	36.917
							Result: 39.021

Example - Duration-Based Average Response Time Rule with isSampleAndDurationValid and isSampleValid Method Filters

This rule uses both the **isSampleAndDurationValid** and the **isSampleValid** method filters. These are described in the following sections:

- ["Example - Duration-Based Average Response Time Rule with isSampleValid Method Filter" on page 81](#)
- ["Example - Duration-Based Average Response Time Rule with isSampleAndDurationValid Method Filter" on the previous page](#)

For 10:00 - 11:00, 5 samples exist on the Profile Database:

10:00 Sample1 {Sample Fields: dResponseTime = 60, u_iStatus = 0}
 10:04 Sample2 {Sample Fields: dResponseTime = 30, u_iStatus = 2}
10:25 Sample3 {Sample Fields: dResponseTime = 25, u_iStatus = 6}
10:38 Sample4 {Sample Fields: dResponseTime = 30, u_iStatus = 1}
 10:52 Sample5 {Sample Fields: dResponseTime = 75, u_iStatus = 2}

Sample3 did not pass the **isSampleValid** method filter, and Sample4 did not pass the **isSampleAndDurationValid** method filter. The following 3 samples are taken into calculation:

10:00 Sample1 {Sample Fields: dResponseTime = 60, u_iStatus = 0}
 10:04 Sample2 {Sample Fields: dResponseTime = 30, u_iStatus = 2}
 10:52 Sample5 {Sample Fields: dResponseTime = 75, u_iStatus = 2}

After the filtering, the interval between Sample3 and Sample4 is considered part of the duration of Sample2, but the interval between Sample4 and Sample5 is not considered part of the duration of Sample4 (filtered by **isSampleAndDurationValid**).

The rule calculation is as follows:

Cycle	samples parameter		Keys set by calculateKPI		Aggregated keys and value set by calculateAggregatedKPI		
	Sample	Sample Duration	totalResponseTime	totalDuration	totalResponseTime	totalDuration	Aggregated Value
10:00-10:05	Sample1	240	16200	300			
	Sample2	60			16200	300	54
10:05-10:10	Sample2	300	9000	300	25200	600	42
10:10-10:15	Sample2	300	9000	300	34200	900	38
10:15-10:20	Sample2	300	9000	300	43200	1200	36
10:20-10:25	Sample2	300	9000	300	52200	1500	34.8
10:25-10:30	Sample2	300	9000	300	61200	1800	34
10:30-10:35	Sample2	300	9000	300	70200	2100	33.429
10:35-10:40	Sample2	180	5400	180	75600	2280	33.157
10:40-10:45					75600	2280	33.157
10:45-10:50					75600	2280	33.157
10:50-10:55	Sample5	180	13500	180	89100	2460	36.219
10:55-11:00	Sample5	300	22500	300	111600	2760	40.434
							Result: 40.434

Examples - API Outage by Samples Rule

This section provides examples of Outage by Samples Rules. The following examples are described:

- ["Example - Outage by Samples Rule and Calculation with Default Rule Parameters" below](#)
- ["Example - Outage by Sample Calculation with Minimum Duration of 900 Seconds" on page 86](#)
- ["Example - Outage by Sample Calculation with Maximum Duration of One Hour" on page 86](#)
- ["Example - Outage by Sample Calculation with a Sample Representing Two Failures" on page 87](#)

Example - Outage by Samples Rule and Calculation with Default Rule Parameters

The following section illustrates the Outage by Samples rule, based on the default Outage rule parameters:

Minimum number of failures: 2
 Minimum duration: 0
 Max duration: undefined

A sample represents one failure if the sample's u_iStatus field value is not 0 or 2. The rule also filters out samples whose u_iStatus field value is 6.

```
// Define the sample fields that will be used in the rule calculation.
```

```
def sampleFields = ["u_iStatus"];

/*
 * Implementation of the OutageBySamplesCalculator interface method.
 * If the sample's u_iStatus field value is not 0 or 2, the sample represents 1
 * failure.
 * In any other case the sample represents no failures.
 *
 * For more information refer to the Rules API documentation.
 */
public void calculateOutage(Outage outage, Sample sample) {
    // Take the sample field's u_iStatus value.
    def statusFieldValue = sample.u_iStatus;
    if(statusFieldValue != 0 && statusFieldValue != 2){
        outage.setNumberOfFailures 1;
    }
}

/*
 * Override default implementation of the OutageBySamplesCalculator interface method.
 *
 * If the sample's u_iStatus field value is not 0 or 2, the sample represents 1
 * failure.
 */
public boolean isSampleValid(Sample sample) {
    def statusFieldValue = sample.u_iStatus;
    return (statusFieldValue != 6)
}
}
```

The following calculation illustrates the above Outage by Samples rule.

For 10:00 - 11:00, 6 samples exist on the Profile Database:

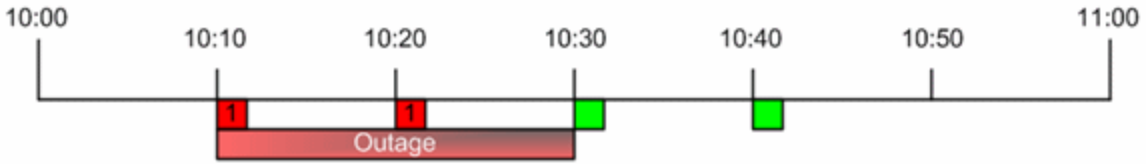
```
10:10 Sample1 {Sample Fields: u_iStatus = 1}
10:20 Sample2 {Sample Fields: u_iStatus = 1}
10:25 Sample3 {Sample Fields: u_iStatus = 6}
10:30 Sample4 {Sample Fields: u_iStatus = 0}
10:35 Sample5 {Sample Fields: u_iStatus = 6}
10:40 Sample6 {Sample Fields: u_iStatus = 2}
```

Sample3 and Sample5 did not pass the **isSampleValid** method. The following samples are passed to the **calculateOutage** method:

```
10:10 Sample1 {Sample Fields: u_iStatus = 1}
10:20 Sample2 {Sample Fields: u_iStatus = 1}
10:30 Sample4 {Sample Fields: u_iStatus = 0}
10:40 Sample6 {Sample Fields: u_iStatus = 2}
```

For Sample1 and Sample2, the following line is invoked inside the **calculateOutage** method:
outage.setNumberOfFailures 1;

The result of the calculation is the following:



An outage is reported with a duration of 20 minutes, from 10:10 - 10:30.

Example - Outage by Sample Calculation with Minimum Duration of 900 Seconds

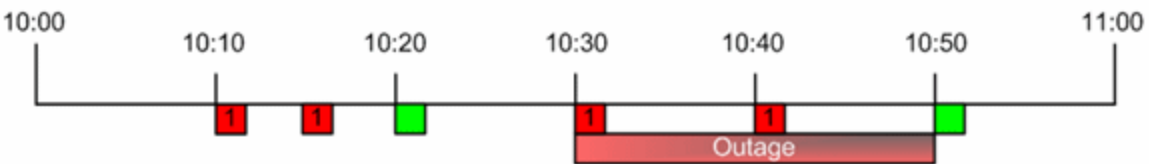
The following calculation illustrates the Outage by Samples rule, based on the following Outage rule parameters:

Minimum number of failures: 2
Minimum duration: 900 (seconds)
Max duration: undefined

For 10:00 - 11:00, 6 samples exist on the Profile Database:

- 10:10 - Sample representing 1 failure.
- 10:15 - Sample representing 1 failure.
- 10:20 - Sample representing no failures.
- 10:30 - Sample representing 1 failure.
- 10:40 - Sample representing 1 failure.
- 10:50 - Sample representing no failures.

The result of the calculation is the following:



An outage is reported with a duration of 20 minutes, from 10:30 - 10:50. There is no outage between 10:10 - 10:20 because the outage duration did not reach the minimum outage duration parameter.

Example - Outage by Sample Calculation with Maximum Duration of One Hour

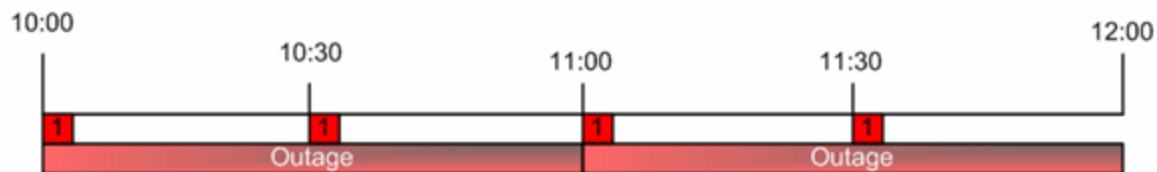
The following calculation illustrates the Outage by Samples rule, based on the following Outage rule parameters:

Minimum number of failures: 2
Minimum duration: 0 (default)
Max duration: 1 (hour)

For 10:00 - 12:00, 4 samples exist on the Profile Database:

- 10:00 - Sample representing 1 failure.
- 10:30 - Sample representing 1 failure.
- 11:00 - Sample representing 1 failure.
- 11:30 - Sample representing 1 failure.

The result of the calculation is the following:



Two outages are reported with a duration of 1 hour; the first outage from 10:00 - 11:00, and the second outage from 11:00 - 12:00.

Example - Outage by Sample Calculation with a Sample Representing Two Failures

The following calculation illustrates the Outage by Samples rule, based on the following Outage rule parameters:

Minimum number of failures: 2
Minimum duration: 0 (default)
Max duration: undefined

For 10:00 - 11:00, two samples exist on the Profile Database:

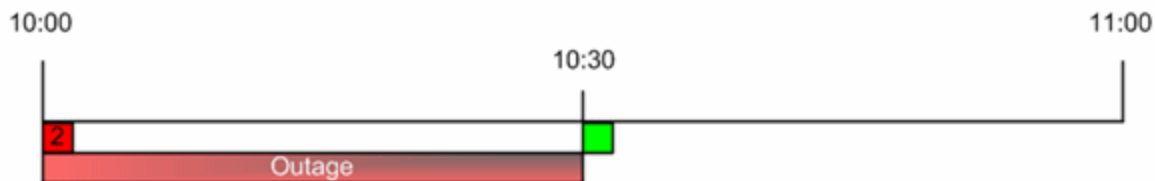
- 10:00 - Sample representing two failures.

The following line of code is invoked in the **outage** parameter in **calculateOutage** method:

```
outage.setNumberOfFailures 2;
```

- 10:30 - Sample representing no failures.

The result of the calculation is the following:



An outage is reported with a duration of 30 minutes, from 10:00 - 10:30.

Chapter 5: SLM Web Services API

The SLM Web Services API is an integration tool enabling administration of SLAs from an application either internal or external to APM. The SLM Web services support the creation and management of SLAs, also in an HPE Software-as-a-Service (HPE SaaS) deployment.

For details, see the following:

- ["Using the SLM Web Services" below](#)
- ["SLM Web Services' Operations " on the next page](#)

Prerequisite Knowledge

Users of the API should be familiar with Service Level Management administration and SOAP concepts.

Permissions

The administrator provides login credentials for connecting with the Web services. The credentials must be those of a user with Administrator permissions, or the SLA owner.

For details on setting permissions in the Permissions Manager, see Permissions Overview in the APM Platform Administration Guide.

Supported Operations

The following Web service operations are supported:

- Create SLA (with no CI)
- Start SLA
- Get SLA general properties
- Update SLA general properties
- Delete SLA
- Add Service to SLA
- Delete Service from SLA
- Get Services that are included in the SLA

Using the SLM Web Services

The SLM Web Services API enables submitting a service request. The engine returns an error description if it cannot parse the statement or does not run successfully. If there is no error, the results of the request are returned.

The Web services are described in a SOAP WSDL file, located at:

```
http://<server>:8080/slm_ws/services/SlmServices?wsdl
```

The port specification is only necessary for non-standard installations. Consult your system administrator for the correct port number.

Developers can use a development environment to generate code from WSDL for calling the Web services. The WSDL describes the interface operations and operation parameters.

SLM Web Services' Operations

This section describes the operations for the SLM Web services.

- ["createSLA" below](#)
- ["startSLA" below](#)
- ["updateSLA" on the next page](#)
- ["deleteSLA" on the next page](#)
- ["getSLAProperties " on the next page](#)
- ["addServicesForSLAWithOfferings" on page 91](#)
- ["deleteServiceFromSLA" on page 91](#)
- ["getSLAServicesFullPath" on page 92](#)
- ["getServiceSLAs" on page 92](#)

createSLA

Creates a new SLA with the specified SLA properties.

Operation signature:

```
String createSLA(String customerId, SlaPropertiesDTO properties)
```

Operation arguments:

Argument	Description
customerId	Typically 1 (apart from in an HPE SaaS environment).
properties	The properties used for initializing the new SLA, as follows: name,description,agreementDetails,type(OLA, SLA,UC), classification (internal, external); startDate,endDate,timeZoneld,customerId, providerId,trackingPeriods

Return-value: Returns the ID of the SLA in the RTSM.

Exception: Throws `SLMWebServiceException` if the user cannot create another SLA; for example, because the user does not have the necessary permissions to create the SLA, or if the number of allowed SLAs has been reached.

startSLA

Starts the specified SLA.

Operation signature:

```
startSLA(String customerId, String SLAId)
```

Operation arguments:

Argument	Description
customerId	Typically 1 (apart from in an HPE SaaS environment).
SLAId	The RTSM ID of the SLA you want to start.

Exception: Throws `SLMWebServiceException` when any type of error occurs.

updateSLA

Updates the SLA with the new properties.

Operation signature:

`updateSLA(String customerId, String SLAId, SLAPropertiesDTO properties)`

Operation arguments:

Argument	Description
customerId	Typically 1 (apart from in an HPE SaaS environment).
SLAId	The RTSM ID of the SLA to update.
properties	The properties to be updated.

Exceptions:

- Throws `SLADoesNotExistsException` if an SLA with the given ID does not exist.
- Throws `SLMWebServiceException` if a system problem occurs that prevents the operation to execute successfully.

deleteSLA

Deletes the specified SLA.

Operation signature:

`deleteSLA(String customerId, String SLAId)`

Operation arguments:

Argument	Description
customerId	Typically 1 (apart from in an HPE SaaS environment).
SLAId	The RTSM ID of the SLA you want to delete.

Exception: Throws `SLMWebServiceException` when any type of error occurs.

getSLAProperties

Retrieves the properties of the specified SLA.

Operation signature:

`SLAPropertiesDTO getSLAProperties(String customerId, String SLAId)`

Operation arguments:

Argument	Description
customerId	Typically 1 (apart from in an HPE SaaS environment).
SLAId	The RTSM ID of the SLA with the properties you want to retrieve.

Exception: Throws `SLMWebServiceException` when any type of error occurs.

addServicesForSLAWithOfferings

Adds the specified services and their impact sub-tree to the SLA. The matching service offering is used for each service.

Operation signature:

```
String[] addServicesForSLAWithOfferings(String customerId, String slaId,
ServiceWithOffering[] servicesWithOffering)
```

Operation arguments:

Argument	Description
customerId	Typically 1 (apart from in an HPE SaaS environment).
SLAId	The RTSM ID of the SLA to which you want to add services.
servicesWithOffering	The services to add to the SLA with matching service offering names. Each <code>ServiceWithOffering</code> includes: <code>serviceId</code> , <code>serviceName</code> , <code>offeringName</code> Note: <ul style="list-style-type: none"> • <code>offeringName</code> must be provided. • You must provide at least one of <code>serviceId</code> or <code>serviceName</code>. If <code>serviceId</code> is missing, the server will try to obtain it using the <code>serviceName</code>.

Return-value: Returns an array of validation error messages (empty array if none).

Exception: Throws `SLMWebServiceException` if a system problem occurs that prevents the operation from executing successfully.

deleteServiceFromSLA

Removes the specified services and their paths from the specified SLA.

Operation signature:

```
deleteServiceFromSLA(String customerId, String SLAId, Service[] services)
```

Operation arguments:

Argument	Description
customerId	Typically 1 (apart from in an HPE SaaS environment).
SLAId	The RTSM ID of the SLA from which you want to remove the services.

Argument	Description
services	The services you want to remove from the SLA. Each service includes: <code>serviceId</code> , <code>serviceName</code> If <code>serviceId</code> is missing, the server will try to obtain it using the <code>serviceName</code> .

Exceptions: Throws `SLMWebServiceException` when any type of error occurs.

getSLAServicesFullPath

Retrieves the services of the specified SLA.

Operation signature:

`ServiceFullPath[] getSLAServicesFullPath(String customerId, String SLAId)`

Example: Example of SLA with services 4, 5:

```
{id1, id2, id3, id4}  
{id1, id2, id5}
```

Operation arguments:

Argument	Description
customerId	Typically 1 (apart from in an HPE SaaS environment).
SLAId	The RTSM ID of the SLA

Return-value: Returns the full paths of the services in the SLA.

Exceptions: Throws `SLMWebServiceException` when any type of error occurs.

getServiceSLAs

Returns the SLAs that include the specified service.

Operation signature:

`String[] getServiceSLAs(String customerId, String serviceId)`

Operation arguments:

Argument	Description
customerId	Typically 1 (apart from in an HPE SaaS environment).
serviceId	The RTSM ID of the service.

Return-value: Returns an array of SLA IDs that include the specified service.

Exceptions: Throws `SLMWebServiceException` when any type of error occurs.

Part 3: User Management

Chapter 6: User Admin External API

You can use the User Admin External API to manage users.

The API enables you to get, add, and delete users. For details, see:

- "Get All Groups/Users" below
- "Post New User" on page 96
- "Get Specific User" on page 97
- "Get Specific Group" on page 99
- "Delete User from Group" on page 100
- "Post Existing User to a Group" on page 101

The service log file is located under: **<Gateway server root directory>\log\EJBContainer\acweb.log**.

Return values are supported in XML format.

Authentication should be done using basic access authentication method. For details and examples refer to http://en.wikipedia.org/wiki/Basic_access_authentication.

Get All Groups/Users

You can use the following to retrieve all groups/users.

Note: You must be a superuser or administrator to access this API.

API Syntax

GET Request

`http://<APM Host>/topaz/acweb/usermanagement/<customerId>/users`

The API uses the following parameters:

- **customerId**. Customer ID (use **1** for non-HPE SaaS deployment).

The following is an example of the response:

```
<all>
  <groups>
    <group>
      <name>root group</name>
      <groups>
        <group>
          <name>second-level group</name>
          <users>
            <user>
              <name>second-level user</name>
              <login-name>sl-user</login-name>
```

```
        </user>
      </users>
    </group>
  </groups>
  <users>
    <user>
      <name>first-level user</name>
      <login-name>fl-user</login-name>
    </user>
  </users>
</group>
<group>
  <name>root group2</name>
  <users>
    <user>
      <name>second-level user</name>
      <login-name>sl-user</login-name>
    </user>
  </users>
</group>
</groups>
<users>
  <user>
    <name>root user</name>
    <login-name>r-user</login-name>
  </user>
  <user>
    <name>user administrator</name>
    <login-name>uadmin</login-name>
  </user>
</users>
</all>
```

The output fields are as follows:

Field	Description
group	Group name
user	User name and login name

Return Codes

The API returns the following return codes:

Name	Error Code	Description
OK	200	Success
INTERNAL_SERVER_ERROR	500	Internal application error

Post New User

You can use the following to add new users. You can select the groups in which to add the new user and assign roles to the user. However, you cannot assign permissions to the user.

By default, the new user has the same default permissions as a user created using the UI:

- View:Enterprise
- View:Customer
- Add:Custom_Reports
- Add:Trend_Reports

Note: You must be a superuser or administrator to access this API.

API Syntax

POST Request

http://<APM Host>/topaz/acweb/usermanagement/<customerId>/users/user

The API uses the following parameters:

- **customerId.** Customer ID (use **1** for non-HPE SaaS deployment).

The following is an example of the request body:

```
<user>
  <name>rest-user1</name>
  <login>ruser1</login>
  <password>admin</password>
  <roles>
    <role>BPM Viewer</role>
  </roles>
  <parents>
    <group>root group</group>
  </parents>
</user>
```

or

```
<user>
  <name>rest-user1</name>
  <login>ruser1</login>
  <password>admin</password>
</user>
```

The output fields are as follows:

Field	Description
name	Name of the user (required)
login	Login name (required)
password	User's password (required)
role	User's role
group	The group in which to add the user

Return Codes

The API returns the following return codes:

Name	Error Code	Description
OK	200	Success
INTERNAL_SERVER_ERROR	500	Internal application error Some required fields are empty

Get Specific User

You can use the following to retrieve a specific user.

Note: You must be a superuser or administrator to access this API.

API Syntax

GET Request

http://<APM Host>/topaz/acweb/usermanagement/<customerId>/users/user/<user-name>

The API uses the following parameters:

- **customerId.** Customer ID (use **1** for non-HPE SaaS deployment).
- **user-name.** User name

The following is an example of the response:

```
<user>
  <id>5</id>
  <name>second-level user</name>
  <login-name>sl-user</login-name>
  <parents>
    <group>
      <name>root group2</name>
    </group>
```

```
<group>
  <name>second-level group</name>
</group>
</parents>
<roles>
  <role>BPM_Viewer</role>
  <role>RUM_Viewer</role>
  <role>DEFAULT</role>
</roles>
<permissions>
  <permission actionOn="RUM_Engine" action="VIEW"/>
  <permission actionOn="TREND_REPORTS" action="ADD"/>
  <permission actionOn="EUM_SR_Folder" action="VIEW"/>
  <permission actionOn="VIEW" action="REMOVE"/>
  <permission actionOn="CUSTOMER" action="VIEW"/>
  <permission actionOn="VIEW" action="VIEW"/>
  <permission actionOn="BPM_Agent" action="VIEW"/>
  <permission actionOn="VIEW" action="CHANGE"/>
  <permission actionOn="EUM_Alert" action="VIEW"/>
  <permission actionOn="EUM_SR_Folder_Content" action="VIEW"/>
  <permission actionOn="EUM_Application" action="VIEW"/>
  <permission actionOn="RUM" action="VIEW"/>
  <permission actionOn="ENTERPRISE" action="VIEW"/>
  <permission actionOn="CUSTOM_REPORTS" action="ADD"/>
  <permission actionOn="VIEW" action="FULLCONTROL"/>
  <permission actionOn="BPM" action="VIEW"/>
</permissions>
</user>
```

The output fields are as follows:

Field	Description
id	ID of user
name	Name of user
login-name	Login name of user
group	User's group
role	User's role
permission	User's permissions

Return Codes

The API returns the following return codes:

Name	Error Code	Description
OK	200	Success
INTERNAL_SERVER_ERROR	500	Internal application error User not found. (In this case, a message appears in the response body. For example, user[root-user11] not found.)

Get Specific Group

You can use the following to retrieve a specific group.

Note: You must be a superuser or administrator to access this API.

API Syntax

GET (Method) Request

http://<APM Host>/topaz/acweb/usermanagement/<customerId>/groups/group/<group-name>

The API uses the following parameters:

- **customerId.** Customer ID (use **1** for non-HPE SaaS deployment).
- **group-name.** Group name

The following is an example of the response:

```
<group>
  <name>root group</name>
  <users>
    <user>second-level user</user>
  </users>
  <permissions>
    <permission actionOn="ENTERPRISE" action="VIEW"/>
    <permission actionOn="CUSTOMER" action="VIEW"/>
  </permissions>
</group>
```

The output fields are as follows:

Field	Description
name	Group name
user	User name
permission	Group's permissions

Return Codes

The API returns the following return codes:

Name	Error Code	Description
OK	200	Success
INTERNAL_SERVER_ERROR	500	Internal application error Group not found. (In this case, a message appears in the response body. For example, group[root-grouppp]not found.)

Delete User from Group

You can use the following to remove an existing user from a group.

Note: You must be a superuser or administrator to access this API.

API Syntax

DELETE Request

http://<APM Host>/topaz/acweb/usermanagement/<customerId>/groups/user/<user-name>

The API uses the following parameters:

- **customerId.** Customer ID (use **1** for non-HPE SaaS deployment).
- **user-name.** user name

The following is an example of the request body:

```
<groups>  
  <group>  
    root group  
  </group>  
</groups>
```

Or

```
<groups/>
```

The output fields are as follows:

Field	Description
group	Group name

Return Codes

The API returns the following return codes:

Name	Error Code	Description
OK	200	Success
INTERNAL_SERVER_ERROR	500	Internal application error Group or user not found. (In this case, a message appears in the response body. For example, group[root-grouppp]not found.)

Post Existing User to a Group

You can use the following to add an existing user to a group.

Note: You must be a superuser or administrator to access this API.

API Syntax

POST Request

http://<APM Host>/topaz/acweb/usermanagement/<customerId>/groups/user/<user-name>

The API uses the following parameters:

- **customerId.** Customer ID (use **1** for non-HPE SaaS deployment).
- **user-name.** User name

The following is an example of the request body:

```
<groups>
  <group>
    root group
  </group>
</groups>
```

The output fields are as follows:

Field	Description
group	Group name

Return Codes

The API returns the following return codes:

Name	Error Code	Description
OK	200	Success
INTERNAL_SERVER_ERROR	500	Internal application error Group or user not found. (In this case, a message appears in the response body. For example, group[root-grouppp]not found.)

Part 4: End User Management

Chapter 7: EUM Admin Open API

The EUM Admin Open API enables you to perform operations on EUM configurations without using the EUM Administration user interface. The API supports retrieving, updating and creating Business Process Monitor and Real User Monitor configurations.

The EUM Admin Open API is a RESTful Web service. It contains an integrated server side module that is compliant to JAX-RS v1.0 standard. In addition, a java client based on the Apache Wink framework is provided.

The EUM Admin Open API classes are documented in Javadoc format in the APM EUM Administration API Reference. These files are located in the following folder:

\\< Gateway Server root directory>\AppServer\webapps\site.war\amdocs\eng\doc_lib\API_docs\EUM_API\EUM_Administration_API_CSH.htm

Part 5: SiteScope

Chapter 8: SiteScope Public API

The SiteScope Public API enables you to perform operations on SiteScope configurations without using the SiteScope or SAM Administration user interface in Business Service Management. The API supports retrieving, updating and creating SiteScope configurations, data acquisition, custom monitor scripting and creating the topology script in Jython.

The SiteScope Public API can be invoked by any known Web Services invocation framework such as Axis or WSIF, or by any client application.

For a list of supported configuration and data acquisition APIs included with SiteScope, see SiteScope APIs in Using SiteScope in the SiteScope Help. Sample scripts for SiteScope configuration and data acquisition APIs can be found in **<SiteScope installation directory>\examples\integrations\api**. For documentation, see the HPE SiteScope Public API Reference Guide located in **<SiteScope installation directory>\examples\integrations\api\doc\javadoc.zip**.

Sample scripts for all the custom monitors are available in the HPE SiteScope Custom Monitor API Reference, located in **<SiteScope installation directory>\examples\monitors\custom\doc\javadoc.zip** or from the Home page of the SiteScope Help.

Part 6: Service Health Analyzer

Chapter 9: Groovy


SHA is fully automated and can automatically detect metrics that have abnormal behavior and combine them into anomalies. SHA provides several Groovy scripts that you can activate and modify to control how SHA processes metrics.

Groovy For Anomaly Detection

There is a Groovy script that enables you to control when a specific metric should be considered problematic, or to suppress or escalate a specific anomaly. Following are examples of how you could use the Groovy script:

- Determine when a specific metric should be considered problematic.
- Suppress a specific metric. For example, you may want SHA to ignore a metric that is often abnormal, but does not have a big impact on your business.
- Escalate all instances of a specific metric. For example, you may consider all metrics that monitor cash withdrawals important, and need to be reported, even if there is only minor abnormal activity.
- Fine-tune when a specific metric should be considered abnormal. For example, a metric may have a regular spike at the beginning of a month or before holidays. Even though this spike is outside the baseline, it should be considered normal, and SHA should not generate an alert over this behavior.
- Set a different upper and lower coefficient for a metric type or each instance of a metric, and define a minimal value at which a metric should be considered abnormal.

How to implement a Groovy Script for Anomaly Detection

1. On the Gateway server, open the following directory:
\\<GW>\HPBSM\confanalytics_loader\
2. Make a copy of the file **shaAbnormalityFunction.groovy_example** named **shaAbnormalityFunction.groovy** (delete "*_example*" from the file extension).
3. In APM Infrastructure Settings, activate the access to the Groovy script:
 - a. In APM, select **Admin > Platform > Setup and Maintenance > Infrastructure Settings**.
 - b. In **Applications**, select **Service Health Analyzer**.
 - c. Under **Allow customization (using Groovy) of the default abnormal behavior detection per metric**, next to the required value, click **Edit Setting**  and change the value to **True**.
4. In the JMX Console, enable the Groovy script:
 - a. In a browser, enter the following to log into the JMX Console:
http://<GW>:29926
where <GW> is the FQDN of your APM Gateway server.
 - b. Click **SHA custom abnormality function**.
 - c. Under **void enable**, set a value of **True** and then click **Invoke**.
5. To test that SHA can read the Groovy file:

- a. In a browser, enter the following to log into the JMX Console:
http://<GW>:29926
- b. Click **SHA custom abnormality function**.
- c. Under **java.lang.String testGroovy** enter the required parameters and click **Invoke**.

java.lang.String testGroovy

validate and returns the /conf/analytics_loader/shaAbnormalityFunction.groovy result OR 'none' for usual calculation/ script compilation error (see advanced.analytics.loader.plugin.log). -1 denotes special use of the result

Parameters

Name	Class	Value	Description
customerId	java.lang.String	<input type="text"/>	leave empty for the default customer
metricId	int	<input type="text"/>	arbitrary id - not related to the real Id in the analytics-metric-def
bulkId	java.lang.String	<input type="text"/>	default is zero for test metric, or a different value for the real metric definition
dimensions	java.lang.String	<input type="text"/>	key=value;... for example Metric=CPU;Node=hp.corp.net
timestamp	java.lang.String	<input type="text"/>	time in seconds since 1970 - or empty for the current time
value	java.lang.String	<input type="text"/>	leave empty for 0
mean	java.lang.String	<input type="text"/>	-
deviation	java.lang.String	<input type="text"/>	-
deviationCheck	java.lang.String	<input type="text"/>	lower/ upper or empty for both
useGroovyState	boolean	<input type="radio"/> true <input type="radio"/> false	use the same state from the regular operation, leave false if not sure

Examples of how to Modify the Groovy File

Following are some of the ways that you can customize the **shaAbnormalityFunction.groovy** file:


Ignore a Specific CI or Metric

Add the CI or metric name to the following lines of the file:

```
boolean importantCI(Map params) {
Map<String, String> cis = params.get("CIs");
if (cis.values().contains("myDatabaseName"))
return false;
return true;
}
```

Create a Weekly or Yearly Calendar

You can modify the way SHA process data on specific date, for example you may want to double the coefficient over the weekend or in the week before major holidays.

1. In a browser, enter the following to open the Calendar application:
http://<GW>/topaz/jsps/slm/SLMAdminApplet.jsp?target=SwingSchedules
where <GW> is the FQDN of your APM Gateway server.
2. In the Calendar application, click the **New Calendar**  button.
3. Enter a name and description for the calendar and select a period type:
 - **Weekly** - Set a weekly pattern based on half-hour blocks of time, for example every Saturday 09:00 - 17:30

- **Yearly** - Set a yearly pattern based on dates, for example December 15 - 30.

Note: The **Compound** option is not currently supported.

4. Click **Next**, enter the required blocks of time or calendars to be combined, and click **Finish**.

Create a List of Public Holidays

SHA provides a file with calculations for several default public holidays which may have abnormal metric behavior. Many of these holidays are not on fixed calendar dates, for example they fall on the first Monday of the month.

After modifying and inserting the required holidays, you need to resave the **shaAbnormalityFunction.groovy** file to communicate the changes to SHA.

The file **shaAbnormalityFunction.groovy_example_holiday** contains calculations for the following holidays which can be applied or modified if needed.

- Day of week (weekend)
- New Years Day
- Independence Day Observed
- Christmas Day
- Valentine Day
- Thanksgiving Observed

How to Identify an Abnormality Based on Groovy

In APM, you can identify anomalies that are based on Groovy in the Metrics View tab and in the Anomaly Highlights window.

- In the Metrics View tab, the name of the metric is followed by **[C]**; the tooltip over the metric name includes the text "**Custom rule abnormality**".
- In the Anomaly Highlights window, under Possible Causes the "Abnormal Metric" field includes the text "**(using Groovy)**".

Groovy for Anomaly Severity

SHA automatically applies algorithms to define the severity of an anomaly. You can use a Groovy script to override the automated algorithm and manually configure the severity level for a specific anomaly.

Following are examples of how you can manually control the severity level:

- SHA should always regard anomalies about cash withdrawals as "Critical".
- A specific location may be down, which indicates a local problem, not related to the central location and therefore SHA should always regard a zero response time as "Major", not "Critical".

Example:

The following example from the `shaStatus.groovy` file changes a specific metric from "Critical" to "Minor".

```
if (ShaStatusUtils.hasAvailabilityMetrics(anomaly))  
return Severities.CRITICAL.getSeverity();
```

```
else  
return ShaStatusUtils.suppressToMinor(ShaStatusUtils.getSeverity(params));
```

How to implement a Groovy Script for Anomaly Severity

1. On the Gateway server, open the following directory:
`\\<GW>\HPBSM\conf\pi_engine\`
2. Make a copy of the file `shaStatus.groovy_example` named `shaStatus.groovy` (delete `"_example"` from the file extension).
3. In the JMX Console, confirm that SHA can read the Groovy script:
 - a. In a browser, enter the following to log into the JMX Console:
`http://<DPS>:29925`
where `<DPS>` is the FQDN of your BSM Data Processing Server.
 - b. Click **SHA Engine**.
 - c. Under `java.lang.String testGroovy`, click **Invoke**.

If SHA is interacting correctly with the Groovy Script, the following result appears:

Groovy status for the test anomaly is minor, severity: MINOR

4. Once you have confirmed that SHA can read the Groovy Script, open the file `shaStatus.groovy` in a text editor and modify the script as required. For a list of common APIs in the script, see "[shaStatus.groovy APIs](#)" below.

After each change to the `ShaStatusUtils.groovy` file, save the file to communicate the changes to SHA, and then retest the file using the JMX Console (as described above). The anomaly used for the test is bundled in the `ShaStatusUtils.groovy` file.

5. Once you have completed the process, you can monitor the Groovy status in the following log file:

`\\DPI\C$\HPBSM\log\pi_engine\pi_engine.log`

Following are example of a Groovy log entries

```
(GroovyStatusAdapter.java:154) INFO - Script reloaded /conf/pi_
engine/shaStatus.groovy, edit time: Tue Oct 08 14:56:12 IDT 2013, Content:
```

or

```
Groovy status for anomaly 1 is minor, severity: MINOR
```

shaStatus.groovy APIs

Following is a list of many of the common `shaStatus.groovy` APIs

API	Applications
<code>getSeverity</code>	Gets the original severity of the anomaly, as determined by the SHA engine.
<code>suppressToMinor</code>	Changes "Critical" severity to "Minor"; makes no change to a non-critical severity.

API	Applications
hasOnlyAvailabilityMetrics	Checks if all the metrics in the anomaly are abnormal due to an availability problem. Relevant only for BPM, SiteScope, BSM-Connector, MS-SCOM, or NNM.
hasAvailabilityMetrics	Checks if the anomaly contains a metric with an abnormality cause by an availability problem.
hasPerformanceMetrics	Checks if the anomaly contains a metric with performance breaches.
hasCustomMetrics	Checks if the anomaly contains a metric with customized Groovy results.
getAnomaly	Gets the anomaly.
getAbnormalMetrics	Gets anomaly metrics.
getTestAnomaly	Test anomaly for JMX testGroovy .

Groovy for Custom Drilldowns from the SHA Investigation UI

You can use a Groovy script to add custom drilldowns to SHA. The custom drilldown can open an *.exe file or open a URL.

To Create a Custom Drilldown Groovy Script

1. On the Gateway server, open the following directory:
\\<GW>\AppServer\webapps\site.war\conf\pi_drilldowns
This directory contains examples of each of the available out-of-the-box Groovy scripts.
2. Make a copy of the required script and change the extension from ***.groovy_example** to ***.groovy**. For information about out-of-the-box templates, see ["Out-of-the-box Templates" on the next page](#).
3. Open the file in a text editor and modify the file as required. For information about creating or modifying a template, see ["Modifying Groovy Script Files" on page 115](#).
4. In a browser, enter the following:
http://<GW>:8080/jmx-console/
where <GW> is the FQDN of your APM Gateway server.
5. Under **Topaz**, click **service=Service Health Analyzer Drilldown**.
6. In the **reloadScripts ()** section, click **Invoke** to load the script. Then invoke **displayScripts()** to confirm that it is valid.
7. Check that the custom drilldown works correctly from the user interface, or from the **testScript()** section.

Out-of-the-box Templates

Out-of-the-box Groovy templates are located on the Gateway server in the following directory:
`\\<GW>\AppServer\webapps\site.war\conf\pi_drilldowns.`

The following scripts open specific reports:

- `app_health.groovy`
- `bpm_error_summary.groovy`
- `bpm_transaction_invocation.groovy`
- `diagnostics.groovy`
- `highlights.groovy`
- `kpi_over_time.groovy`
- `nnm.groovy`
- `nnm_perfSpi.groovy_example`
- `om_performance_graph.groovy`
- `om_run_tool.groovy`
- `rum_events_summary.groovy`
- `rum_session_analyzer.groovy`
- `scom.groovy_example`
- `sitescope_metric.groovy`
- `sla_summary.groovy`

The following shows run books:

- `run_books.groovy`

The following script is used to demonstrate invocation of command line program under the Windows operating system. This script also demonstrates the usage of TQL from inside the script.

- `ping.groovy`

Example:

Following is an example of the `bpm_error_summary.groovy` file:

```
import Utils;
import EUMFilterUtils;

// You can import any JAR which is in \\HPBSM\lib,
// or \\HPBSM\AppServer\webapps\site.war\WEB-INF\lib
// or groovy script from \\HPBSM\AppServer\webapps\site.war\conf\pi_drilldowns\utils

String getDrilldownCommand(Map params) {

    return Utils.getGateway(params) +
        EUMFilterUtils.getGeneralPrefix
("application_error_summary") +
        EUMFilterUtils.getRfwTimeFilter(params) +
        EUMFilterUtils.getApplicationFilter(params) +
        EUMFilterUtils.getLocationFilter(params) +
```

```
        EUMFilterUtils.getTransactionFilter(params);
    }

    String getName (Locale locale) {
        // see \\HPBSM\AppServer\webapps\site.war\conf\pi_drilldowns\resources for i18n
        return new Utils().getString
        (locale,"menu.key.name.application_error_summary");
    }

    String getDescription (Locale locale) {
        return new Utils().getString
        (locale,"menu.key.description.application_error_summary");
    }

    // filters method

    String[] getDataProviders () {
        // Use NONE, ALL, or more data providers.
        // NONE indicating drilldown only from the topology view, where the pressed CI is
        known, in
        // contrary to drilldown from metrics view
        // (you have several dimensions and cannot know on which one to operate the
        drilldown)
        return [Utils.getDataProvider("BPM")];
    }

    String[] getCiTypes() {
        // You should return at list one CI type name (not the display name),
        // this can be the configuration_item.
        // The framework will match also the ancestores of these CI types
        return [
            "business_application",
            "business_transaction_flow",
            "business_transaction"];
    }

    // not mandatory
    String isAvailable(Map params) {

        if (DimensionsUtils.getApplication(params) == null)
            return new Utils().
        MsgBox(params, "drills.application.not.found");

        if (! EUMFilterUtils.
        isEUMConfigured(params).getBpmConfigured()) {
            CiDTO application =
                DimensionsUtils.getApplication(params);
            if (application != null)
                return new Utils().
        MsgBox(params, "drills.no.bpm.app.configured",
```

```
application.getDisplayName());
    else
        return new Utils().
MsgBox(params, "drills.no.bpm.configured");
    }

    return "";
}
```

Modifying Groovy Script Files

You can modify Groovy script files as follows:

Add additional URL parameters for the generation of the command.

You can use any of the following parameters:

- APM's GATEWAY_HOSTNAME.
- The anomaly's START_TIME in milliseconds.
- The anomaly's END_TIME.
- The anomaly's TIME_ZONE.
- DIMENSIONS – If the drilldown appears in the Metrics View, you can use any of the metric's dimensions (as they appear in the histogram). If the drilldown appears in the Topology view, then the only dimension available is the CI (ID, Display Name, and CI type).
- METRIC_DISPLAY_NAME – If the drilldown appears in the Metrics View, you can get the metric name.
- CUSTOMER_ID, USER_ID, USER_NAME, and LOCALE of the current user.
- GROOVY_UTILS – Following is the set of available functions:
 - getCmdbClassAncestors(ci_type) – Used by **diagnostics.groovy**.
 - getModelGraph(ci_ids) – Not in use.
 - getCi(ci_id), getCiProperties(ci_id, properties) – Used to get additional property on a CI ID. Used by **sitescope_monitor.groovy**.
 - isTransaction/ isLocation/ isHost(ci) – Used to get the transaction, location, and host name from the DIMENSIONS.
 - getCiIdByGlobalId(global_id) – Used to get the CI ID by its global ID.
 - runTQL(tql, ci_id, inputName, outputName, properties...) – Used to get properties on outputName CI, based on the TQL and single CI. Used by **ping.groovy** and **diagnostics.groovy** (to get the application CI).
 - getSlaIds(ci_ids, session) – Used to get SLA IDs. Used by **sla_summary.groovy**.

For examples of how these parameters are use, see the example Groovy files.

Specify the data provider

You can use **SiteScope**, **BPM**, **RUM**, **Diagnostics**, **NNMi** or **PA**. **ALL** for any data provider, and **NONE** for drilldowns from only the topology view.

Specify the CI Type.

Enter the CI Types (separated with commas) which are relevant for the drilldowns that you are creating (for example `node`, `business_transaction`).

Specify optional availability check

Checks the URL parameters needed for the command generation. Returns either an empty string for valid entry, or the reason why an entry is invalid (for example, **"This CI is not monitored by SiteScope"**).

Part 7: Downtime

Chapter 10: Downtime REST Service

You can use a RESTful web service running on the Gateway Server to retrieve, update/terminate, 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 further information about downtime, see Downtime Management Overview in Platform Administration.

Note: The permissions required to use downtime REST API are the same as the permissions configured for the Downtime entity in the User Management for specific users.

Supported HTTP Requests

The downtime REST service supports the following HTTP requests:

Note: CustomerID is always 1 except in the case of HPE SaaS customers.

Action	HTTP Command
Retrieve all downtimes	<code>http://<HPBSM server>/topaz/bsmservices/customers/[customerId]/downtimes</code>
Retrieve a specific downtime	<code>http://<HPBSM server>/topaz/bsmservices/customers/[customerId]/downtimes/[downtimeId]</code>
Update/Terminate a downtime using http PUT	<code>http://<HPBSM server>/topaz/bsmservices/customers/[customerId]/downtimes/[downtimeId] + XML of the downtime</code> Note: To terminate a downtime, redefine the end time of the downtime by adding or modifying the <code><endDate></code> tag.
Create downtime using http POST	<code>http://<HPBSM server>/topaz/bsmservices/customers/[customerId]/downtimes + XML of the downtime</code> Note: Successful creation of the downtime causes a return of the newly created downtime in XML format, including the downtime ID.
Delete downtime using http DELETE	<code>http://<HPBSM server>/topaz/bsmservices/customers/[customerId]/downtimes/[downtimeId]</code>

Allowed Downtime Actions

Use the XML commands listed for the following downtime actions:

Action Description	XML Command
Take no actions	<code><action name="REMINDER"/></code>
Suppress alerts and close events	<code><action name="SUPPRESS_NOTIFICATIONS"/></code>

Action Description	XML Command
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"/>
Enforce downtime on Reports and KPI calculations; suppress alerts and close events (continue monitoring), including all SLAs	<action name="ENFORCE_ON_REPORTS"> <propGroup name="SLA" value="ALL"/> </action>
Enforce downtime on Reports and KPI calculations; suppress alerts and close events (continue monitoring), including specific SLA	<action name="ENFORCE_ON_REPORTS"> <propGroup name="SLA" value="SELECTED"> <prop>dda3fb0b20c0d83e078035ee1c005201</prop> </propGroup> </action>
Stop active monitoring (BPM and SiteScope); enforce downtime on Reports & KPI calculations; suppress alerts and close events	<action name="STOP_MONITORING"/>

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.

Note: For this example, all fields are mandatory.

```
<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 should not:
 - 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.

Note: Although you can create, delete, or modify downtimes retroactively via the REST web service, it is not recommended.

- The date format of startDate/endDate is: **yyyy-MM-dd'T'HH:mm:ssZ**
- For weekly and monthly downtimes, the startDate and endDate can be defined for any time, but we recommend that it 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 using Java Code

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:

```
\\< Gateway Server root directory>\AppServer\webapps\site.war\amdocs\eng\doc_lib\  
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.

Downtime REST Example Using Groovy

If business management solutions (such as HPSM or third party software) create downtime events when integrating with APM, 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 APM.

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 APM downtime, the mandatory fields are:

- Downtime Name
- CI ID
- Schedule
- Action

Imported events must be translated to match APM downtime properties.

Import Mechanism

Importing downtimes into APM 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 APM 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:

```
\\< Gateway Server root directory>\AppServer\webapps\site.war\amdocs\eng\doc_lib\  
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 APM 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 APM 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 APM REST service to create downtime in APM. You can alter the contents of this file to create your own custom integrations with BSM 9.1 and higher or APM.

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 APM server, the external source, or any other server.
3. Edit the DTImport.bat file to update the APM 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 (**http://<Gateway or Data Processing Server name>:8080/jmx-console/**) 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.

Part 8: Reporting in APM

Chapter 11: Generic Reporting Engine API

The recommended method for creating API-level queries to the profile database is building queries using the Custom Query Builder. The Custom Query Builder enables the building of queries using a graphical user interface, and facilitates the generation of reports, extraction of data in different formats, and generation of query URLs that can be used with third-party or custom tools. For details, see Building a Custom Query Using Custom Query Builder in the APM User Guide.

The Generic Reporting Engine API also enables manual creation of queries using the following methods:

- **Web browser.** The request is sent as an HTML query and the data is returned as HTML or as a CSV (Comma Separated Values) file that can be opened with Microsoft Excel or processed with a custom tool.
- **Web Service.** The return object contains the data in CSV format.

The remainder of this chapter describes how to create queries manually.

Prerequisite Knowledge

Users of the API should be familiar with SQL syntax and APM administration and applications. Users of the API through the Web Service should also be familiar with the SOAP specification and an object-oriented programming language such as C++ or Java.

Permissions

For a query to access the data using the API query syntax described below, the user and password parameters passed in the query must be those of a user with either System Viewer or Superuser permissions. (For details on setting permissions in the Permissions Manager, see Permissions Overview in the APM Platform Administration Guide.)

Note: If a GDE custom query does not contain a user's credentials but indicates the type of report (such as `&resultType=csv`), you need to enable SSL.

Configuration

To configure the API options, select **Admin > Platform > Setup and Maintenance > Infrastructure Settings**:

- Select **Foundations**.
- Select **Generic Data Engine Open API**.
- In the Generic Data Engine Open API - Generic Data Engine Open API Settings table, locate:
 - **Maximum Rows.** Change the maximum number of data rows returned.
 - **Enable User Credentials in URL in Open API.** Open API requires basic authentication. If **Enable User Credentials in URL in Open AP** is set to true, Open API will also accept user credentials in the URL.

Getting Metadata on the Samples

When building queries, you must know the data representation of the sample. For information on commonly queried samples and descriptions of their fields, see Data Samples in the APM Application Administration

Guide.

Advanced Sample Retrieval

Users with special reporting needs can retrieve a list of all samples and their fields using the MBean Inspector. Access the MBean Inspector page by entering the following URL in your browser:

```
http://<server>[:port]/jmx-console/HtmlAdaptor?action=inspectMBean&name=Topaz%3AService%3DMeta-Data+Manager
```

The default port number is 8080. If this port is incorrect, consult your system administrator for the correct port number.

Enter your JMX console authentication credentials. If you do not know your authentication credentials, contact your system administrator.

On the MBean Inspector page, click the **Invoke** button next to the operation **showMetaDataDBMapping**. The bean returns the list of the fields in each sample.

Data Returned

The same data is returned whether the request is made from a browser or with the Web Service. With a browser, the data resides in the response body, and for the Web Service, the data resides in the return object.

Web Browser Response Body

When the query is submitted from a browser, the response CSV or HTML contains either the data, or an error code and message. If the number of rows to be returned exceeds the maximum, the last row of the data is `Returned X of Y rows`, where `X` is the number of rows returned and `Y` is the actual number of rows that fulfil the conditions of the query. If there is an error at the engine level, the HTTP success code is returned, but the body of the response is `<error code>`, `<error message>`.

Web Service Return Object

The Web Service return object contains the following:

- **retval**. The data or an error message.
- **errorCode**. The error code (type int). Possible error codes are:
 - 0 - Success
 - 100 - Authorization error
 - 101 - Processing error
 - 102 - Open API has been disabled
- **origRowCount**. The actual number of rows the query should have returned (type int). If the number of rows to be returned exceeds the maximum, the **origRowCount** field is set to the actual number of rows that the query would have returned had the maximum not been exceeded.

Querying with a Browser

When querying with a browser, the `getData` service is called with the URL:

```
http://<server>[:port]/topaz/gdeopenapi/GdeOpenApi?method=
getData&user=<username>&password=<password>&query=<query>
```

The URL can include an optional `resultType` parameter:

```
http://<server>[:port]/topaz/gdeopenapi/GdeOpenApi?method=getData&user=
<username>&password=<password>&query=<query>&resultType=csv
```

The port specification is only necessary for non-standard installations. Consult your system administrator for the correct port number.

The default return type is HTML. If `resultType=csv` is specified, a comma separated values file is returned.

Note: In an HPE SaaS environment, include the following additional parameter `customerID=<customer id>` .

Using the Web Service

The API Web Service enables submitting a query consisting of a username, password, and an SQL-like select statement. The engine returns an error description if it cannot parse the statement or if there is a problem running the query. If there is no error, the results of the query are returned.

The SOAP WSDL is at:

```
http://<server>[:port]/topaz/gdeopenapi/services/GdeWsOpenAPI?wsdl
```

The port specification is only necessary for non-standard installations. Consult your system administrator for the correct port number.

Supported SQL Syntax

The language supported is a subset of SQL and supports these keywords, modifiers, and operators:

- SELECT
- WHERE
- FROM
- TOP
- HAVING
- Aliasing with the AS keyword
- Logical operators OR , AND , NOT
- DISTINCT modifier (only supported for select list items)
- IN operator. Inner selects can be used to return the values for the IN operator.
- BETWEEN operator
- IS NULL (IS NOT NULL is not supported)
- LIKE. The wildcard character is the asterisk (*). Do not use the percent sign (%). The asterisk can not be used by itself (LIKE *). It must be used with other characters.
- Mathematical operators: +, -, *, /, (,)

- Comparators: =, IS, !=, <>, >, >=, <, <=
- ORDER BY and the ASC and DESC modifiers

Supported Functions

The supported functions and their descriptions are as follows:

Function	Field Formula	Description
MAX	MAX(<Data field>)	Returns the maximum value of the data represented in the selected data field.
MIN	MIN(<Data field>)	Returns the minimum value of the data represented in the selected data field.
SUM	SUM(<Data field>)	Returns the summed value of the data represented in the selected data field.
COUNT	COUNT(<Data field>)	Returns the number of records in the data represented in the selected data field. Note: COUNT does not count records in which the value of the selected data field is NULL.
AVG	AVG(<Data field>)	Returns the average value of the data represented in the selected data field.
STDDEV	STDDEV(<Data field>)	Returns the standard deviation of the data represented in the selected data field.
SUMOFSQR	SUMOFSQR(<Data field>)	Returns the sum of squares of the data represented in the selected data field.
LOG	LOG(<Data field>,?)	Returns the logarithm of ? where the data represented in the selected data field is the base.
CEIL	CEIL(<Data field>)	Returns the smallest integer value that is greater than or equal to the data represented in the selected data field.
FLOOR	FLOOR(<Data field>)	Returns the largest integer value that is equal to or less than the data represented in the selected data field.
MOD	MOD(<Data field>,?)	Returns the remainder (modulus) of the data represented in the selected data field divided by ?.

Function	Field Formula	Description
SQRT	SQRT(<Data field>)	Returns the square root of the data represented in the selected data field.
REPLACENULL	REPLACENULL(<Data field>,?)	Replaces null values in the data field with ?. Note: REPLACENULL is equivalent to Oracle's NVL and Microsoft SQL Server's ISNULL
IF	IF(<Data field>,?,?,?)	The function formula IF(<Data field>,a,b,c,d) contains a condition where a represents the relation between the <Data field> and b, c represents the result if the condition is true, and d represents the result if the condition is false. Example: You enter the following field formula: IF (dResponseTime,>,5000,pass,fail) . If the response time is greater than 5000, the result is pass; if the response time is less than 5000, the result is fail.
byTime	This function is not available for selection in the Custom Query Builder page.	For details, see "byTime Function" on the next page.

Query Limitations

The following limitations apply to queries submitted to the service:

- Only one monitor type can be selected in a single query.
- The asterisk (*) is not supported as a wildcard character except in combination with the LIKE operator. It is supported as the multiplication operator.
- Inner selects and joins are not supported, with one exception: an inner select can be used to return the values for an IN clause.
- The ORDER BY clause requires a column number, for example ORDER BY 1. ORDER BY column name is not supported.
- The engine requires that queries contain a time limitation (that is, a condition for the time_stamp field) in the WHERE clause.
- The GROUP BY clause is not supported. It is unnecessary because the engine treats all fields that do not have an aggregate function as GROUP BY fields.
- When manually defining a filter that consists of strings containing white space or special characters (for example, where bb_guid IN (a b, c)), you must enclose the white space or special character string with quotes (for example, where bb_guid IN (`a b', c)). When you create filters on the Filter Builder

page, APM automatically adds the quotes. Special characters are defined as any characters other than digits, letters, and the following characters: "_", "\$", "#".

- When defining a filter that consists of strings containing one or more single quote characters, you must add a second single quote character beside each instance. For example, change `szTransactionName = ('Login_to_0'Brien')` to `szTransactionName = ('Login_to_0''Brien')`.
- The columns in the returned data are labeled `Column 0`, `Column 1`, and so on. To return meaningful column labels, use the SQL AS operator. For example, `Select time_stamp as TimeStamp`. With this use of the AS operator, the column label is `TimeStamp`.
- The "COUNT (DISTINCT <field>)" syntax is not supported. Instead use the "COUNT DISTINCT (<field>)" syntax.

Date-Time Values

Time in queries and return data is specified in seconds since January 1, 1970. You can use Microsoft Excel to convert between time values in seconds and date-time.

Time is most commonly used for time stamp fields.

To get a GMT time for use in a query:

Enter the date and time in a Date-formatted cell and in another cell, formatted as General, enter the formula:
`=(<date cell> - 25569) * 86400`

To correct for a local time zone:

Add the time zone offset times 3600 seconds to the result. For example, for Central Europe (GMT + 2):
`=(<date cell> - 25569) * 86400 + (2 * 3600)`

To view a time value from a query as a GMT date in Excel:

Use a Date format for the cell and enter the formula:
`=<time stamp> / 86400 + 25569`

To correct for a local time zone:

Subtract the time zone offset times 3600 seconds from the time stamp. For example, for the Eastern United States, standard time (GMT - 3):
`=(<time stamp> - (-3 * 3600)) / 86400 + 25569`

byTime Function

The Generic Reporting Engine SQL supports the function **byTime**, which returns data grouped by time periods. For example, querying the average response time of a transaction for the past day without the **byTime** function returns one value. You can use the **byTime** function to view the average response time of the transaction for each hour of the past day. In this case, a value is returned for each hour of the past 24 hours.

The function syntax is:

byTime(<timefield>, <step value>, <number of step>, <offset>)

Argument	Description
<i>timefield</i>	Usually a timestamp field
<i>step value</i>	One of: 10 - Second 20 - Minute 30 - Hour 40 - Day 50 - Week 60 - Month 70 - Quarter 80 - Year
<i>number of step</i>	The number of the units specified in <i>step value</i> to group.
<i>offset</i>	Time zone offset from GMT in hours. Positive numbers indicate time zones East of GMT. Negative numbers indicate time zones west of GMT.

For example, to return one value for each 3 days, corrected to one hour east of GMT:

`byTime(time_stamp, 40, 3, 1)`

Query Examples

Below are several examples of query URLs that retrieve different types of data from the database.

Example of ss_t Sample

This example illustrates retrieving the average value for SiteScope samples on a given measurement and monitor:

`http://myServer/topaz/gdeopenapi/GdeOpenApi?method=getData&user=admin&password=admin&query=select szMeasurementName, szMonitorName, avg(dValue) from ss_t where u_iStatus=1 and time_stamp > 123456 and szMeasurementName = 'myMeasurmentName' and szMonitorName = 'myMonitorName'`

Example of trans_t Sample

This example illustrates retrieving the average response time, grouped by minutes and offset to GMT + 3 for Springfield_infra_ems_login transactions in the Springfield_Location application for a given period from BPM data:

`http://myServer/topaz/gdeopenapi/GdeOpenApi?method=getData&user=admin&password=admin&query=select byTime(time_stamp, 20, 1, 3.0), application_name as ApplicationName, szTransactionName as TransactionName, AVG(dResponseTime) from trans_t where time_stamp >= 1126594800.64 and time_stamp < 1126596000.64 and application_name='Springfield_Location' and szTransactionName='Springfield_infra_ems_login'`

Example of rum_action_t Sample

This example illustrates retrieving the total server time for each URL as measured by RUM:

```
http://myServer/topaz/gdeopenapi/GdeOpenApi?method=getData&user=admin&password=admin&query=select application_name as ApplicationName,action_name as ActionName,action_descriptor,AVG(tot_server_time) from rum_action_t where time_stamp>=1304197200.64 and time_stamp<1306702800.64 and application_name='EC2%20jpetstore' and action_name='Sign In'
```

Example of rum_application_stats_t Sample

This example illustrates retrieving the average server time of the application's actions on each of the servers serving the application as measured by RUM:

```
http://MyServer/topaz/gdeopenapi/GdeOpenApi?method=getData&user=admin&password=admin&query=select application_name as ApplicationName,server_host_name as hostName,Avg(tot_server_time) as serverTime from rum_application_stats_t where time_stamp>=1304197200.64 and time_stamp<1306702800.64 and application_name='EC2 jpetstore' group by application_name,server_host_name
```

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 APM Extensibility Guide (Application Performance Management 9.30)

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 SW-doc@hpe.com.

We appreciate your feedback!