



TIBCO ActiveMatrix® Service Grid

Service Performance Manager API Reference

*Software Release 3.4
April 2019*

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

ANY SOFTWARE ITEM IDENTIFIED AS THIRD PARTY LIBRARY IS AVAILABLE UNDER SEPARATE SOFTWARE LICENSE TERMS AND IS NOT PART OF A TIBCO PRODUCT. AS SUCH, THESE SOFTWARE ITEMS ARE NOT COVERED BY THE TERMS OF YOUR AGREEMENT WITH TIBCO, INCLUDING ANY TERMS CONCERNING SUPPORT, MAINTENANCE, WARRANTIES, AND INDEMNITIES. DOWNLOAD AND USE OF THESE ITEMS IS SOLELY AT YOUR OWN DISCRETION AND SUBJECT TO THE LICENSE TERMS APPLICABLE TO THEM. BY PROCEEDING TO DOWNLOAD, INSTALL OR USE ANY OF THESE ITEMS, YOU ACKNOWLEDGE THE FOREGOING DISTINCTIONS BETWEEN THESE ITEMS AND TIBCO PRODUCTS.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, the TIBCO logo, Two-Second Advantage, TIB, Information Bus, ActiveMatrix, Business Studio, Enterprise Message Service, Hawk, and Rendezvous are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. Please see the readme.txt file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 2010-2019. TIBCO Software Inc. All Rights Reserved.

Contents

Figures	5
TIBCO Documentation and Support Services	6
Introduction	8
TIBCO Service Performance Manager API	9
Model Overview	10
RTASchema	11
Attributes and Facts	11
Dimension	11
Time Dimension	11
Dimension Hierarchy	12
Dimension Hierarchy Attributes	13
Cube	13
Measurement	14
Retention Policy	15
Additional Properties in the Schema	15
Publishing the Facts to the Server	18
Executing the Query	19
Setting up the Rules	21
The Runtime Model API	23
Creating Custom Metric Functions	24
Creating a Class with a Metric Function	24
Creating a Function Catalog	24
Binding the Metric Parameters to the Fact Attributes in the Schema	26
Make the Metric Function Available to the Product	26
Creating Custom Actions	27
Creating a Class that Extends AbstractActionHandlerContext	27
Creating a Class that Extends AbstractActionImpl	27
Creating an Action Catalog	28
Making the Custom Action Available to the Product	29
Java API Reference Pages	30

Figures

The Model Objects and their Relationship 10

Binding Function Parameters to the Schema 25

TIBCO Documentation and Support Services

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit <https://docs.tibco.com>.

Product-Specific Documentation

Documentation for TIBCO ActiveMatrix® Service Grid is available on the <https://docs.tibco.com/products/tibco-activematrix-service-grid> page.

Use of the following features, installation profiles and development tools requires a TIBCO ActiveMatrix Service Grid license:



- TIBCO ActiveMatrix Policy Director Governance, TIBCO ActiveMatrix SPM Dashboard, and TIBCO ActiveMatrix SPM Runtime Server profiles; and
- TIBCO ActiveMatrix Service Grid development tools for Java, Webapp and Spring components.

Customers with only a TIBCO ActiveMatrix Service Bus license are not licensed to use these features, tools or profiles.

The following documents form the documentation set:

- *TIBCO ActiveMatrix Service Grid Concepts*: Read this manual before reading any other manual in the documentation set. This manual describes terminology and concepts of the platform. The other manuals in the documentation set assume you are familiar with the information in this manual.
- *TIBCO ActiveMatrix Service Grid Development Tutorials*: Read this manual for a step-by-step introduction to the process of creating, packaging, and running composites in TIBCO Business Studio.
- *TIBCO ActiveMatrix Service Grid Composite Development*: Read this manual to learn how to develop and package composites.
- *TIBCO ActiveMatrix Service Grid Java Component Development*: Read this manual to learn how to configure and implement Java components.
- *TIBCO ActiveMatrix Service Grid Mediation Component Development*: Read this manual to learn how to configure and implement Mediation components.
- *TIBCO ActiveMatrix Service Grid Mediation API Reference*: Read this manual to learn how to develop custom Mediation tasks.
- *TIBCO ActiveMatrix Service Grid Spring Component Development*: Read this manual to learn how to configure and implement Spring components.
- *TIBCO ActiveMatrix Service Grid WebApp Component Development*: Read this manual to learn how to configure and implement Web Application components.
- *TIBCO ActiveMatrix Service Grid REST Binding Development*: Read this manual to learn how to configure and implement REST components.
- *TIBCO ActiveMatrix Service Grid Administration Tutorials*: Read this manual for a step-by-step introduction to the process of creating and starting the runtime version of the product, starting TIBCO ActiveMatrix servers, and deploying applications to the runtime.
- *TIBCO ActiveMatrix Service Grid Administration*: Read this manual to learn how to manage the runtime and deploy and manage applications.

- *TIBCO ActiveMatrix Service Grid Hawk ActiveMatrix Plug-in*: Read this manual to learn about the Hawk plug-in and its optional configurations.
- *TIBCO ActiveMatrix Service Grid Policy Director Governance Custom Actions*: Read this manual to learn how you can configure and enforce policies for ActiveMatrix and external services hosted in third party containers, using TIBCO ActiveMatrix Policy Director Governance.
- *TIBCO ActiveMatrix Service Grid Service Performance Manager API Reference*: Read this manual to learn how to use the SPM APIs.
- *TIBCO ActiveMatrix Service Grid Error Codes*: Read this manual to know more about the error messages and how you could use them to troubleshoot a problem.
- *TIBCO ActiveMatrix Service Grid Installation and Configuration*: Read this manual to learn how to install and configure the software.
- *TIBCO ActiveMatrix Service Grid Security Guidelines*: Read this manual to learn more about security guidelines and recommendations for TIBCO ActiveMatrix Service Grid.
- *TIBCO ActiveMatrix Service Grid Release Notes*: Read this manual for a list of new and changed features, steps for migrating from a previous release, and lists of known issues and closed issues for the release.

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit <http://www.tibco.com/services/support>.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at <https://support.tibco.com>.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to <https://support.tibco.com>. If you do not have a user name, you can request one by clicking Register on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](https://community.tibco.com). For a free registration, go to <https://community.tibco.com>.

Introduction

TIBCO Service Performance Manager provides the capability to monitor and manage services and assets in your enterprise in real-time. The Service Performance Manager product suite consists of a central Service Performance Manager server which provides real-time aggregation capabilities.

The suite also provides the Service Performance Manager client API. Applications should use the Service Performance Manager Client API to connect to the server and submit monitoring related data to the server. You can use the Service Performance Manager Client API to model your aggregations in a flexible manner. This guide covers the various aspects of the Service Performance Manager Client API.

TIBCO Service Performance Manager API

The TIBCO Service Performance Manager is an API-driven product. You can use the TIBCO Service Performance Manager API to define rules and metrics that helps you monitor and manage a group of services and assets in the enterprise such as machines, resources, and so on. The API is broadly classified into Model, Client, Query, Rules, and Runtime API.

Model API

This part of the API set deals with the core model that is used across the client and the server. It consists of classes and interfaces that are used to model the aggregation rules and the overall aggregation schema. The Model API provides classes and interfaces to represent elements of a schema.

Client API

This part of the API lets client applications connect to the Service Performance Manager server and perform various operations. Some such operations are to get the aggregation schemas, submit facts for aggregations, perform queries on the aggregated sets, and so on. From this point onwards, Service Performance Manager Server will be referred as "the server" in this document.

Applications connect to the server and submit data or facts to the server. The server uses these facts as input to compute the aggregations as defined in the schema. The server also includes a Query API to perform snapshot and streaming queries.

Query API

This part of the API set lets client applications to define queries (snapshot and streaming) and execute them. It also includes a Filter API that helps the application build complex filters using AND/OR conditions.

Rule API

This part of the API set deals with Rules, Actions, and alerts. It allows applications such as the ActiveMatrix Dashboard to define certain conditions (using the Query API) and when they match, to define actions and alerts. Actions are of two types, Set actions and Clear actions. Set actions are those that are invoked when a rule condition evaluates to `true` and a Clear actions are those that are invoked when a previously set rule condition evaluates to `false`.

Runtime API

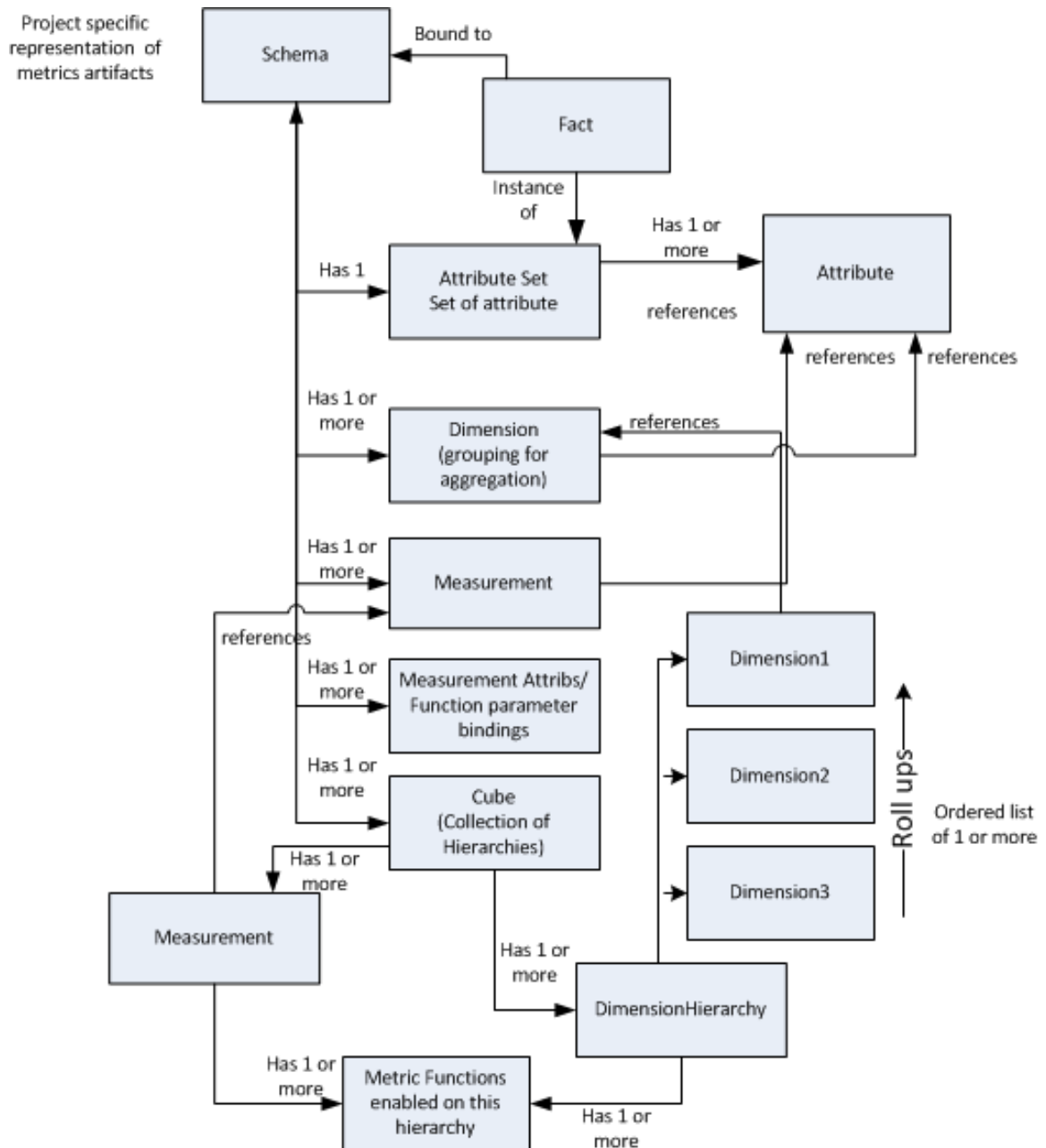
This part of the API exposes certain runtime classes and interfaces that lets you build custom actions and metric computation functions.

Model Overview

A high level summary of the Model API is explained. For details refer to the API documentation that ships with the product.

The figure depicts the model objects and their relationships.

The Model Objects and their Relationship



RTASchema

The RTASchema represents a logical grouping of all model elements. It is the parent object for all the schema elements defined under it. The XML equivalent of the RTASchema is the schema XML file.

For example, the Service Performance Manager server loads the `AMX_3_0_SPM_Schema.xml` at startup. While loading, the Service Performance Manager server reads from the schema file. The table shows attributes and their equivalent API for a Schema:

API for the Attributes of a Schema

Attributes	API Usage
name	String getName()
display-name	String getDisplayName()
description	String getDescription()

The next sections show example fragments of the *schema* file for illustration.

Attributes and Facts

An attribute represents a data element. It has a name and a data type. Client applications set values of these attributes using a fact. A *Fact* is a collection of related attributes that the client program uses to send to the server. A fact is always bound to a schema. Therefore, defining a set of attributes in the model lets the API and the server validate the attributes and their values that are set in the fact.

```
<attributes>
  <attribute datatype="STRING" name="host"/>
  <attribute datatype="STRING" name="environment"/>
  <attribute datatype="STRING" name="node"/>
</attributes>
```

The table shows attributes and their equivalent API for the Attribute element:

API for the Attribute Element

Attributes	API Usage
attribute	Attribute RTASchema.getAttribute(String name)
name	String getName()
description	String getDescription()

Dimension

An aggregation or a metric computation has several Dimensions. Each Dimension categorizes or qualifies the metric computation by providing additional information about the metric being computed. It can be considered as an attribute of a metric.

A dimension is defined at a global level in the schema. It is then referenced by dimension hierarchies. The name of the dimension can be different from that of the attribute.

Time Dimension

A TimeDimension is an extension of a Dimension. It categorizes the computation by a particular time-slot. For example, a TimeDimension can be defined to compute a certain metric such as an average

response time for each minute, or for every 2 hours or for every 2 weeks. A TimeDimension provides methods to define such a time-slot in terms of time units and time periods.

The following snippet is an example of a TIME dimension.

```
<time-dimension name="month" attribute-ref="timestamp" unit="MONTH" frequency="1"/>
<time-dimension name="weeks" attribute-ref="timestamp" unit="WEEK" frequency="1"/>
<time-dimension name="days" attribute-ref="timestamp" unit="DAY" frequency="1"/>
<time-dimension name="hours" attribute-ref="timestamp" unit="HOUR" frequency="1"/>
<time-dimension name="minutes" attribute-ref="timestamp" unit="MINUTE"
frequency="1"/>
```

There are some predefined units such as months, weeks, days, hours, and minutes. The frequency is the rate at which you want to compute the metrics. For example, if the frequency of the minutes dimension is set to 2, the aggregation is done every two minutes. The `qtrOffset` is only applicable when the unit is a quarter and indicates the start of the quarter.

Dimension Hierarchy

A *DimensionHierarchy* is an ordered list of Dimensions over which aggregations are computed and rollup computations are performed. For example, if you want to compute minute-wise, hourly and daily aggregations, you would add them to the API model in that order. Each dimension defined in the hierarchy has an implicit level; starting with zero for the first dimension in the hierarchy increasing by one as you traverse the ordered hierarchy.

A set of measurements or metrics to be computed for each of these levels are defined for the hierarchy. The API has methods to:

- Exclude certain metrics to be computed at specific levels, or
- Not compute measurements completely at a specific level.

By default, all metrics associated with the hierarchy are computed at all levels.

```
<cube>
  <hierarchy name="ByService">
    <properties>
      <property name="storage-schema" value="ByService"/>
    </properties>
    <dimensions>
      <dimension ref="service_name" compute="false"/>
      <dimension ref="application_name" compute="false"/>
      <dimension ref="environment" compute="false"/>
      <dimension ref="node" compute="false"/>
      <dimension ref="host" compute="false"/>
      <dimension ref="service_type" compute="false"/>
      <dimension ref="weeks"/>
      <dimension ref="days"/>
      <dimension ref="hours"/>
      <dimension ref="minutes"/>
    </dimensions>
    <measurement-refs>
      <measurement ref="HitCount"/>
      <measurement ref="SuccessCount"/>
      <measurement ref="FaultCount"/>
      <measurement ref="AvgResponseTime"/>
      <measurement ref="TP5ResponseTime"/>
      <measurement ref="TP95ResponseTime"/>
    </measurement-refs>
  </hierarchy>
</hierarchies>
</cube>
```

The dimension mentioned in the `<dimension-ref>` should be a part of the dimensions defined in the schema.

In the `Dimension` element, the value in the `compute` attribute indicates whether or not metrics are computed for a dimension. By default, this value is set to `true`. If set to `false`, the computation is not propagated upwards and it stops at the dimension where `compute` is set to `false`.

The measurement mentioned in the `<measurement-ref>` should be a part of the measurements defined in the schema.



Changes made to the schema are reflected only on restarting the Service Performance Manager server.

The table gives attributes and their equivalent API for the hierarchy element:

API for the Hierarchy Element

Attributes	API Usage
hierarchy	DimensionHierarchy Cube.getDimensionHierarchy(String name)
name	String getName()
display-name	String getDisplayName()
description	String getDescription()
enabled	boolean isEnabled()

Dimension Hierarchy Attributes

A dimension hierarchy can be enabled or disabled at runtime. When disabled, there are no computations performed for that hierarchy. By default, all hierarchies are enabled.

To disable a hierarchy, add the `enabled="false"` attribute to the hierarchy. For example:

```
<hierarchy name="ByService" enabled="false">
```

This value takes effect only when you restart the Service Performance Manager server.

The `DimensionHierarchy.setEnabled()` indicates whether or not the hierarchy is enabled. If the method returns true, the hierarchy is enabled. The API to disable the hierarchy is `MutableDimensionHierarchy.setEnabled(false)`.

Cube

Cubes are a logical grouping of related hierarchies.

```
<cube>
  <hierarchy name="ByService">
    <properties>
      <property name="storage-schema" value="ByService"/>
    </properties>
    <dimensions>
      <dimension ref="service_name" compute="false"/>
      <dimension ref="application_name" compute="false"/>
      <dimension ref="environment" compute="false"/>
      <dimension ref="node" compute="false"/>
      <dimension ref="host" compute="false"/>
      <dimension ref="service_type" compute="false"/>
      <dimension ref="weeks"/>
      <dimension ref="days"/>
      <dimension ref="hours"/>
      <dimension ref="minutes"/>
    </dimensions>
    <measurement-refs>
      <measurement ref="HitCount"/>
      <measurement ref="SuccessCount"/>
      <measurement ref="FaultCount"/>
      <measurement ref="AvgResponseTime"/>
      <measurement ref="TP5ResponseTime"/>
      <measurement ref="TP95ResponseTime"/>
    </measurement-refs>
  </hierarchy>
</cube>
```

```

    </measurement-refs>
  </hierarchy>
</hierarchies>
</cube>

```

The table shows attributes and their equivalent API for the cube element:

API for the Cube Element

Attributes	API Usage
cube	Cube RTASchema.getCube(String name)
name	String getName()
display-name	String getDisplayName()
description	String getDescription()

Measurement

A *measurement* represents the actual metric that is to be computed. The measurement model object binds the parameters of the function specified in metric-function to the attributes of the schema. This is a schema level object. After defining a measurement in the schema, it can be used in a DimensionHierarchy.

```

<measurements>
  <measurement name="HitCount" unit="hit">
    <metric-function ref="System.SUM">
      <function-params>
        <function-param name="PARAM1" attribute-ref="hit"/>
      </function-params>
    </metric-function>
  </measurement>
</measurements>

```

The table shows attributes and their equivalent API for the measurement element:

API for the Measurement Element

Attributes	API Usage
measurement	Measurement RTASchema.getMeasurement(String name)
name	String getName()
display-name	String getDisplayName()
description	String getDescription()
unit	String getUnitOfMeasurement()
depends	Collection <Measurement> getDependencies()
datatype	DataType getDataType()

Retention Policy

The retention policy can be defined in the schema.

For example:

```
<retention-policies>
<retention-policy type="fact" period="5" unit="WEEK" purge-time-of-day="1200" purge-
frequency-period="3600000"/>
<retention-policy type="DevNodeCube/DemoServiceHitCount" period="5" unit="WEEK"
purge-time-of-day="1200" purge-frequency-period="86400000"/>
</retention-policies>
```

where:

- `type="fact"` indicates that the purge policy is applied on the facts.
- `period="5" unit="WEEK"` indicates that data older than 5 weeks is purged.
- `purge-time-of-day="1200"` starts the timer for the purge policy at 12.00 P.M. The first two digits denote the hour and last two digits denote the minutes. Time is represented in a 24-hour notation, in the "hhmm" format. To start the timer on startup, set `purge-time-of-day` to "-1".
- `purge-frequency-period="3600000"` indicates the frequency period after which you the purge policy is triggered. This value is in milliseconds. In this case the purge policy is triggered at 12:00 P.M. and after that at a time interval of "3600000" milliseconds.

The table shows attributes and their equivalent API for the retention policy in the schema:

API for the Retention Policy Element

Attributes	API Usage
retention-policy	Collection <code><RetentionPolicy></code> <code>RTASession.getRetentionPolicies()</code>
type	Qualifier <code>getQualifier()</code> . If the qualifier is <code>HIERARCHY</code> , then you can get the type by using <code>String getHierarchyName();</code>
unit	<code>TimeUnits.Unit getRetentionUnit()</code>
period	<code>long getRetentionPeriod()</code>
purge-time-of-day	<code>String getPurgeTimeOfDay()</code>
purge-frequency-period	<code>long getPurgeFrequencyPeriod()</code>

Additional Properties in the Schema

Additional properties can be set as name/value pairs on some of the model artifacts.

In XML, they translate as:

```
<properties>
  <property name="property-name" value="value"/>
  ...
</properties>
```

The corresponding APIs are `MetadataElement.getProperty()` and `MutableMetadataElement.setProperty()`. These properties are used by the runtime. The table shows the properties and their meanings:

Properties

Property Name	Default	Possible Values	Applicable to	Description
storage-datatype	no default	INTEGER LONG DOUBLE STRING BOOLEAN, CLOB,	attributes, measurements	The storage-datatype property defines the datatype used for the column in the underlying database table. For some attributes and measurements, the datatype may not be the same as the underlying datatype (of the attribute) or the metric function (for measurements). For example, the storage datatypes of some large strings, have to be represented as CLOBs. In this case, you can set the storage-datatype as "CLOB"
storage-schema	no default		dimension hierarchies	Represents the table name used for the underlying database. Since the underlying database schema is the same for all Service Performance Manager schemas, this has to be a unique name across the various Service Performance Manager schemas.

Property Name	Default	Possible Values	Applicable to	Description
asset-hierarchy	false	true, false		<p>Represents the hierarchy of assets. Some hierarchies represent the assets in the system. Assets are those entities that have life-cycle events associated with them. Asset are usually created, started, deleted, and so on. Asset hierarchies are different from other hierarchies that are simply aggregations of some measurements over different dimensions. For example, in the TIBCO ActiveMatrix schema, nodes, environments, applications are all assets.</p> <p>Asset hierarchies have a special meaning in the Server Performance Manager system. A change to the asset status can trigger a change to other non-asset hierarchies. For example, if the underlying asset is deleted, all measurements that include this asset in other computation hierarchies are also deleted.</p>
asset-name				Represents the name of the underlying asset. This is only applicable if asset-hierarchy is set to true.

Publishing the Facts to the Server

The Service Performance Manager provides a set of Client APIs to publish facts to the server.

Procedure

1. First, the client applications connect to the server for defining the schema model and for submitting facts. The server then performs aggregations based on user-defined schema or cubes.

To acquire a connection to the server from the factory, the client applications use the following well-defined properties:

```
RtaConnectionFactory connectionFac = new
RtaConnectionFactory();
Map<ConfigProperty, PropertyAtom<?>> configurationMap = new
HashMap<ConfigProperty, PropertyAtom<?>>(); // put all the client properties in
this map
RtaConnection connection = connectionFac.getConnection("server_url", "user_name",
"password", configurationMap);
```

2. RtaSession is a logical layer for interacting with the server. It provides Query service and FACT submission service

To get a session, set the well-defined properties using the ConfigProperty, and create a session from the connection.

```
RtaSession session = connection.createSession("session_name", configurationMap);
session.init();
```



If a session operation is called before `init()` is successfully completed, the caller thread is blocked.

3. Create an instance for this measurement as follows:

```
RtaSchema schema = session.getSchema("AMX-Schema");
Fact fact = schema.createFact();
```

4. Set various attribute values that map to the defined dimensions as follows:

```
fact.setAttribute("Environment", "ENV1");
fact.setAttribute("Application", "BookingService");
fact.setAttribute("ServiceName", "SeatAvialibility");
fact.setAttribute("HitCount", 1);
```

5. Publish the fact:

```
session.publishFact(fact);
```

Executing the Query

The server computes the metric results based on the facts published. You can query the server about the computed metric results using the Query API. The steps help you create and execute a query and browse through the results.

Procedure

1. Define a Query Object.

The code snippets defines query object is:

```
final Query query = session.createQuery();
QueryByFilterDef queryDef = query.newQueryByFilterDef(
    SCHEMA_NAME, CUBE_NAME,
    DIM_HIERARCHY_NAME,
    MEASUREMENT_NAME);
queryDef.setName("SnapshotQueryDef-Gt-filter");
queryDef.setBatchSize(5);
```

2. Set the type of query you are using. The valid types are Snapshot and Streaming. The *snapshot* queries help you evaluate the metric results at a given point-in-time, whereas the *streaming* queries are executed continuously.

- `queryDef.setQueryType(QueryType.SNAPSHOT);`
- `queryDef.setQueryType(QueryType.STREAMING);`

3. Define a filter.

The filter is similar to the WHERE clause in a standard SQL query. You can add various filter conditions using the filters that are available.

```
Filter eqFilter = QueryFactory.INSTANCE.newEqFilter(
    MetricQualifier.DIMENSION_LEVEL,
    DIM_LEVEL_SERVICE);
Filter gtFilter = QueryFactory.INSTANCE.newGtFilter(
    FilterKeyQualifier.MEASUREMENT_NAME,
    MEASUREMENT_HITCOUNT, 5.0);
AndFilter andFilter = QueryFactory.INSTANCE.newAndFilter();
andFilter.addFilter(eqFilter, gtFilter);
queryDef.setFilter(andFilter);
```

4. After setting the filter, you can execute the query. In case of streaming queries, it is recommended to execute the query in a separate thread.

```
Browser<MetricResultTuple> browser = query.execute();
```

You can get two types of query results: Streaming query results and snapshot query results.

5. To get Streaming Query results, you must implement QueryResultHandler and register it with the query.

a) Implementing Query ResultHandler

```
private class MyQueryResultHandler implements QueryResultHandler {
    @Override
    public void onData(QueryResultTuple queryResultTuple) {
        MetricResultTuple rs =
            queryResultTuple.getMetricResultTuple();
        if (rs != null) {
            for (String metricName : rs.getMetricNames()) {
                SingleValueMetric metric =
                    (SingleValueMetric) rs.getMetric(metricName);
                if (metric != null) {
                    String measurementName =
                        metric.getDescriptor().getMeasurementName();
                    MetricKey key = (MetricKey) metric.getKey();
                    System.out.println(String.format("%s: Level = %s",
                        queryResultTuple.getQueryName(),
                        key.getDimensionLevelName()));
                }
            }
        }
    }
}
```

```

        for (String dimName : key.getDimensionNames()) {
            System.out.println(String.format("%s: Dimension name =
            %s, value = %s", pad, dimName,
            key.getDimensionValue(dimName)));
        }
        System.out.println(String.format("%s: Metric = %s,
        value = %s", pad,
        metric.getDescriptor().getMeasurementName(),
        metric.getValue()));
        System.out.println();
    }
}
}
}
@Override
public void onError(Object errorContext) {
    System.out.println("OnError...");
}
}

```

b) Registering QueryResultHandler with the Query

```
query.setResultHandler(new MyQueryResultHandler());
```

6. To get snapshot query results, browse the resultset from the Browser object returned after executing the query.

```

while (browser.hasNext()) {
    MetricResultTuple rs = browser.next();
    SingleValueMetric<Long> metric =
    (SingleValueMetric<Long>) rs.getMetric("HitCount");
    MetricKey key = (MetricKey) metric.getKey();
    System.out.println(String.format(" Level = %s",
    key.getDimensionLevelName()));
    for (String dimName : key.getDimensionNames()) {
        System.out.println(String.format("Dimension name = %s,
        value = %s", dimName, key.getDimensionValue(dimName)));
    }
    System.out.println(String.format(" Metric = %s,value = %s",
    metric.getDescriptor().getMeasurementName(),
    metric.getValue()));
    System.out.println();
}
}

```

Setting up the Rules

The Service Performance Manager provides you with another set of API to define a set of rules that, once registered with the server, gets automatically triggered when the threshold conditions are met.

Defining a Rule Definition Object and Set its Properties

Procedure

1. The following code snippet helps you create a MutableRuleDef object and set the properties of the instance.

```
RuleFactory factory = new RuleFactory();
String ruleName = "APIDemoRule";
MutableRuleDef ruleDef = factory.newRuleDef(ruleName +
Math.round(Math.random()));
ruleDef.setScheduleName("schedule1");
ruleDef.setUserName("userName1");
```

2. To set the threshold conditions at which a rule gets evaluated, define a query object, and set its query type to STREAMING. Rules are nothing but continuously evaluating queries.

```
QueryFactory qfac = QueryFactory.INSTANCE;
//*****Set Condition*****
QueryByFilterDef setCondition =
qfac.newQueryByFilterDef(SCHEMA_NAME, CUBE_DEV,
DIM_HIERARCHY_DEMO, MEASUREMENT_HITCOUNT);
setCondition.setName("RuleService");
setCondition.setQueryType(QueryType.STREAMING);
setCondition.setBatchSize(6);
Filter eqFilter = QueryFactory.INSTANCE.newEqFilter(
MetricQualifier.DIMENSION_LEVEL,
DIM_LEVEL_SERVICE);
Filter gtFilter = QueryFactory.INSTANCE.newGtFilter(
FilterKeyQualifier.MEASUREMENT_NAME,
MEASUREMENT_HITCOUNT, 41.0);
Filter ltFilter = QueryFactory.INSTANCE.newLtFilter(
FilterKeyQualifier.MEASUREMENT_NAME,
MEASUREMENT_HITCOUNT, 45.0);
AndFilter andFilter = QueryFactory.INSTANCE.newAndFilter();
andFilter.addFilter(eqFilter, gtFilter, ltFilter);
setCondition.setFilter(andFilter);
//*****Clear Condition*****
QueryByFilterDef clearCondition=
qfac.newQueryByFilterDef(SCHEMA_NAME, CUBE_DEV,
DIM_HIERARCHY_DEMO, MEASUREMENT_HITCOUNT);
clearCondition.setName("ClearCondition");
clearCondition.setQueryType(QueryType.STREAMING);
clearCondition.setBatchSize(6);
Filter eqFilter1 = QueryFactory.INSTANCE.newEqFilter(
MetricQualifier.DIMENSION_LEVEL,
DIM_LEVEL_SERVICE);
Filter gtFilter1 = QueryFactory.INSTANCE.newGtFilter(
FilterKeyQualifier.MEASUREMENT_NAME,
MEASUREMENT_HITCOUNT, 47.0);
Filter ltFilter1 = QueryFactory.INSTANCE.newLtFilter(
FilterKeyQualifier.MEASUREMENT_NAME,
MEASUREMENT_HITCOUNT, 50.0);
AndFilter andFilter2 = QueryFactory.INSTANCE.newAndFilter();
andFilter2.addFilter(eqFilter1, gtFilter1, ltFilter1);
clearCondition.setFilter(andFilter2);
```



Rules across measurements and hierarchies in the conditions are not supported. For example, you cannot create a rule with a condition, such as:

```
application.HitCount > 200 AND node.AverageUsedMemory > 600 MB
```

OR

```
application.SuccessCount > application.HitCount
```

3. Set and clear conditions can be thought of upper and lower bounds for the threshold values. Once these conditions are defined, you need to register them with the rule.

```
ruleDef.setSetCondition(setCondition);
ruleDef.setClearCondition(clearCondition);
```



You cannot set a rule on alerts hierarchy.

4. After setting the threshold conditions for set and clear, define actions to be taken when these conditions are met. Actions have to be first setup on the Service Performance Manager server. Some actions such as "Send-Email" are already set up on the server. Refer to [Creating Custom Actions](#) for details on how to set up custom actions on the server. First, initialize the action definitions using `session.getAllActionFunctionDescriptors()`.

```
//Get a handle to a pre-defined action function descriptor.
session.getAllActionFunctionDescriptors();
ActionFunctionDescriptor sendToSessionActionFn =
ActionFunctionsRepository.INSTANCE.getFunctionDescriptor
("SendToNamedSession");
//Get a handle to the action's function parameters and provide values to its
parameters.
FunctionParam param = sendToSessionActionFn.getFunctionParam("session-name");
ActionFunctionDescriptorImpl.FunctionParamValueImpl
paramValue = new
FunctionDescriptorImpl.FunctionParamValueImpl();
paramValue.setName(param.getName());
paramValue.setDataType(param.getDataType());
paramValue.setIndex(param.getIndex());
paramValue.setDescription(param.getDescription());
paramValue.setValue(sessionName);
//Bind the function parameter value to the action function.
sendToSessionActionFn.addFunctionParamValue(paramValue);
```

5. Add the time-based constraints to control how many times and how frequently the actions must get triggered.

```
InvokeConstraint invokeConstraint1 =
factory.newInvokeConstraint(Constraint.TIMED);
MutableTimeBasedConstraint tbc = (MutableTimeBasedConstraint) invokeConstraint1;
tbc.setInvocationFrequency(1000);
tbc.setMaxInvocationCount(5);
tbc.setTimeConstraint(TimeBasedConstraint.Constraint.TILL_CONDITION_CLEARS);
```

6. After defining the actions, register the actions with a rule either to fire when the set condition is met or to fire when the clear condition is met.

```
ActionDef setSendSessionAction =
factory.newSetActionDef(ruleDef, sendToSessionActionFn, invokeConstraint1);
//A convenience method that clones the passed in action definition and sets it as
an action for the clear condition.
ActionDef clearSessionAction =
factory.newClearActionDef(ruleDef, setSendSessionAction);
```

The Runtime Model API

Service Performance Manager also exposes certain runtime model elements. This provides the ability to introduce user-defined or custom metric functions and custom actions.

MetricNode

`MetricNode` is a node that holds metric values.

RtaNodeContext

`RtaNodeContext` is a container for name/value tuples. It stores stateful data for the metric calculation. It is required to optimize metric calculations so that it does not iterate over child metric or Facts each time there is a new Fact submitted.

MetricFunction

This interface must be implemented to define a custom metric aggregator function.

ActionHandlerContext

This interface must be implemented for one-time context initialization while developing custom actions.

Creating Custom Metric Functions

You can create custom metric functions to add your own aggregation functions. For example, in a TIBCO ActiveMatrix environment, you might want to set the value of a metric function based on the input you get from the fact published by the probe.

- Determine whether your function is a single-valued function or a multivalued function. Based on that, create a custom class that extends one of the following: `SingleValueMetricFunction` or `MultiValueMetricFunction`.
- Create a catalog for the metric function.
- Bind the metric function parameters to the fact attributes in the schema file.
- Bundle the classes related to the metric function and the catalog files together in a JAR file and place it in the server classpath.

Creating a Class with a Metric Function

Procedure

1. Create a class that extends either `SingleValueMetricFunction` or `MultiValueMetricFunction`.

```
public class MyMetric extends SingleValueMetricFunction{...}
```

2. Override the `init()` method. The `init()` method belongs to the abstract class, `AbstractMetricFunction`.

```
public void init(Fact fact, Measurement measurement, MetricNode startNode,
DimensionHierarchy dh) throws java.lang.Exception
```

For more information about the API, refer to the *TIBCO Service Performance Manager Java API* reference pages.

The `init()` method has the one-time initialization of the metric function for a series of metric nodes starting at the leaf node and moving up to the parent node in the rollup hierarchy.

For example, here you can store the current value for a sum computation. Then you can use this current value in the `compute` function at each level and add this current value to the existing sum at that level (Stored in the "context").

3. Override the `compute()` method. This is where you load the previous state from `RtaContext`. Use the current value set in `init()` to compute the new value.

```
public abstract compute(MetricNode metricNode, SingleValueMetric<N> metric,
RtaNodeContext context)
```

For more information about the API, refer to the *TIBCO Service Performance Manager Java API* reference pages.

Creating a Function Catalog

Procedure

1. Create a function catalog file. For example, `my.function.catalog`. Create the function descriptors as shown in the following sample:

```
<function-descriptors name="my-functions">
  <function-descriptor name="My.AVG"
    category="my category" multivalued="false"
    implclass="com.tibco.rta.runtime.metric.functions.MyMetric"
    datatype="DOUBLE" description="My metric function">
  </function-descriptor>
```


Function Descriptors Properties

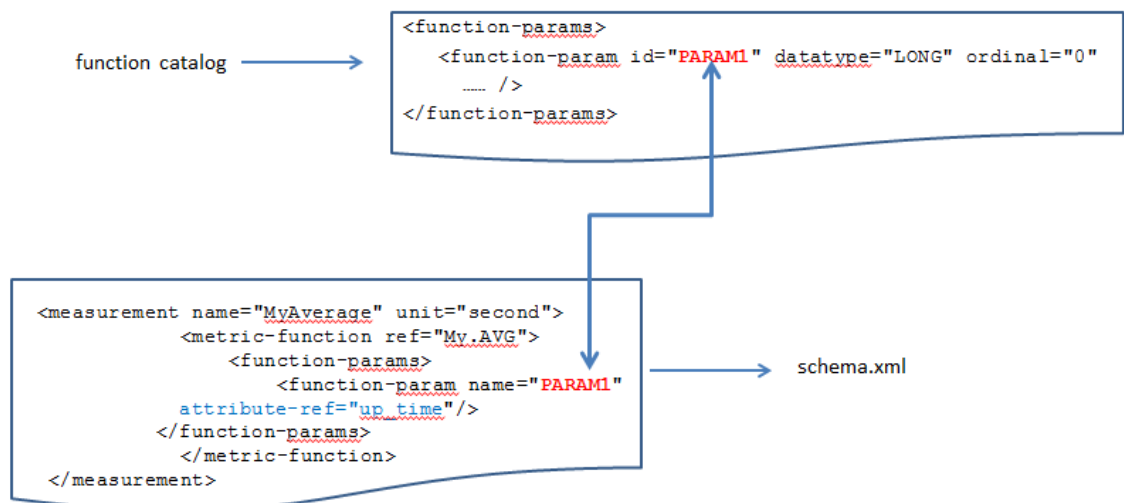
Property Name	Description
name	Name of the function
category	The category name
multivalued	Indicates whether or not the function is single-valued or multivalued. The default value is <code>false</code> . Remember that for a multivalued function, all values must be of the same datatype.
implclass	To create custom metric function, you should extend either <code>SingleValueMetricFunction</code> or <code>MultiValueMetricFunction</code> . For more information about the API, refer to the <i>TIBCO Service Performance Manager Java API</i> reference pages.
datatype	Datatype of the metric. For example, the datatype of AVERAGE is generally be a double.
description	Description of the metric function

2. Create the function parameters as shown in the following example:

```
<function-params>
  <function-param id="PARAM1" datatype="LONG" ordinal="0"
    description="Measuable Quantity to be averaged" />
</function-params>
```

Specify the function parameters, data types, and the ordinal position. The parameters specified here are bound to the fact attributes in the `<product_name>_schema.xml`. For example, the ID specified for the parameter is used in the schema to bind the parameter to the fact.

Binding Function Parameters to the Schema



3. Each metric can store some contextual data for future reuse. The context definition is also used by the database schema generator to generate columns in the metric tables. For each metric, the associated context is stored in these columns in the database. Create the function context as shown in the following example:

```
<function-context>
  <function-param id="count" datatype="LONG" ordinal="0"
    description="" />
</function-context>
```

```

        <function-param id="sum" datatype="DOUBLE" ordinal="1"
        description="" />
    </function-context>
</function-descriptor>
</function-descriptors>

```

Binding the Metric Parameters to the Fact Attributes in the Schema

Procedure

1. Bind the metric function parameters to the fact attributes in the schema file as shown in the following snippet:

```

    <measurement name="MyAverage" unit="second">
        <metric-function ref="My.AVG">
            <function-params>
                <function-param name="PARAM1"                attribute-
ref="up_time"/>
            </function-params>
        </metric-function>
    </measurement>

```

2. Use this measurement in one or more dimension hierarchies as follows:

```

<hierarchy name="NodeTrends">
    <properties>
        <property name="storage-schema" value="NodeTrends"/>
    </properties>
    <dimensions>
        <dimension ref="environment" compute="false"/>
        <dimension ref="host" compute="false"/>
        <dimension ref="node" compute="false"/>
        <dimension ref="weeks"/>
        <dimension ref="days"/>
        <dimension ref="hours"/>
        <dimension ref="minutes"/>
    </dimensions>
    <measurement-refs>
        <measurement ref="MyAverage"/>
    </measurement-refs>
</hierarchy>

```

Make the Metric Function Available to the Product

Procedure

1. Create a JAR file containing the classes related to the metric function and the function catalog file, located at the root of the jar file.
2. The metric function can be made available in one of the following two ways.
 - Update the classpath of the associated server TRA file, which is `tibspm.tra` by default.
 - Copy the JAR files into a location such as `SPM_HOME/lib/ext`.
3. When the Service Performance Manager server starts, it loads all the custom metric function definitions from the function catalog files and makes them available for use in the client API using `session.getAllFunctionDescriptors()`

Creating Custom Actions

You can create custom actions to implement your own actions. For example, instead of sending e-mail notifications, you can send text messages on cell phones.

- Create a class that extends `AbstractActionHandlerContext`
- Create a class that extends `AbstractActionImpl`
- Create a catalog for the custom action
- Bundle the classes related to the custom action and the catalog file together in a JAR file and place it in the server classpath.

Creating a Class that Extends `AbstractActionHandlerContext`

Procedure

1. Create a class that extends `AbstractActionHandlerContext` as shown in the following code snippet:


```
class MyActionHandlerContextImpl extends AbstractActionHandlerContext {
    //override init, stop, getAction here.
    //Action returned by getAction is as shown in Step 3. It should extend
    AbstractActionImpl and provide the performAction method
}
```
2. Override the `init()` method. The `init()` method is called by the Service Performance Manager engine on startup. Perform one-time initialization such as resource allocations, and so on here.


```
void init(java.util.Properties configuration)
```
3. Override the `getAction()` method. This method returns an object of type `Action`. The `Action` object that is returned must implement `AbstractActionImpl`.


```
Action getAction(Rule rule, ActionDef actionDef)
```

For more information about the API, refer to the *TIBCO Service Performance Manager Java API* reference pages.
4. Override the `stop()` method. The `stop()` method is called by the engine during the shutdown process. Perform resource deallocation here.

Creating a Class that Extends `AbstractActionImpl`

Procedure

1. Create a class that extends `AbstractActionImpl`

```
class MyActionImpl extends AbstractActionImpl {
    public MyActionImpl(Rule rule, ActionDef actionDef) {
        super(rule, actionDef);
    }
}
```
2. Override the `getActionHandlerContext()` method. This method returns the associated context handler. The method definition is as follows:


```
ActionHandlerContext getActionHandlerContext()
```
3. Override the `performAction()` method. This method is called by the engine when the filters associated to a rule meet the set criteria.


```
void performAction(Rule rule, MetricNodeEvent node) throws java.lang.Exception
```
4. Override the `getAlertText()` method. This method should return the text associated with this alert. This alert text returned by this class is stored in the corresponding field of the alerts in the system.

5. Override the `getAlertDetails()` method. This method should return the alert details, such as e-mail ID. This string is specific to an action. The alert text returned by this method is stored in the corresponding field of the alerts in the system.

Actions may use values set in the action definition in their function parameters. These values may contain tokens recognized by the system. These tokens are substituted with their corresponding runtime values.

Creating an Action Catalog

Procedure

1. Create an action catalog called `my.action.catalog`. Create the action descriptors as shown in the following sample:

```
<action-descriptor name="My-Action" category="my category"
datatype="BOOLEAN"
implclass="com.tibco.spm.rule.action.MyActionHandlercontextImpl"
description="My custom action function">
```

Function Descriptors Properties

Property Name	Description
name	Name of the function.
category	The category name.
implclass	To create custom metric function, you should extend <code>com.tibco.spm.rule.action.MyActionHandlercontextImpl</code> . This is an implementation class that implements <code>ActionHandlerContext</code> . For more information about the API, refer to the <i>TIBCO Service Performance Manager Java API</i> reference pages.
datatype	Datatype of the action. This is always set to <code>true</code> .
description	Description of the custom action.

2. The action parameters list arguments required by the action function with their ordinal position and data type. The following example for "e-mail action" takes five action parameters:

```
</action-params>
  <action-param id="To" datatype="STRING" ordinal="0" value=""
description="TO Recipient List(Comma Separated)"/>
  <action-param id="Cc" datatype="STRING" ordinal="1" value=""
description="CC Recipient List(Comma Separated)"/>
  <action-param id="Bcc" datatype="STRING" ordinal="2" value=""
description="BCC Recipient(Comma Separated)"/>
  <action-param id="Subject" datatype="STRING" ordinal="3" value=""
description="Subject" />
  <action-param id="Body" datatype="STRING" ordinal="4" value=""
description="e-mail Content" />
</action-params>
</action-descriptor>
</action-descriptors>
```

3. These are made available to the TIBCO ActiveMatrix Dashboard when you use the action. The dashboard user has to provide values to these parameters at the time of configuring an action in a rule.

Making the Custom Action Available to the Product

Procedure

1. Create a JAR file containing the classes related to the custom action and the action catalog.
2. The metric function can be made available in one of the following two ways.
 - Update the classpath of the associated server TRA file, which is `tibspm.tra` by default.
 - Copy the JAR files into a location such as `SPM_HOME/lib/ext`.
3. When the Service Performance Manager server starts, it loads all the actions from the actions catalogs and makes their descriptors available for use in the client API using the `session.getAllActionFunctionDescriptors()` method.

Java API Reference Pages

Detailed descriptions of each class and method in the TIBCO Service Performance Manager Java API reference are provided.

Click the API link to access the TIBCO Service Performance Manager Java API reference page.

[Java API Reference](#)