HP OpenView SD/SLM

for the UNIX and Windows operating system

Software Version: 5.0

Open Metric Adapter Developer Guide

Manufacturing Part Number: None

March 2006

Legal Notices

Warranty

Hewlett-Packard makes no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Copyright Notices

Trademark Notices

Java™ is a US trademark of Sun Microsystems, Inc.

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

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

UNIX® is a registered trademark of The Open Group.

openadaptor™ is a trademark of the Dresdner Kleinwort Wasserstein

Support

Please visit the HP OpenView web site at:

http://www.managementsoftware.hp.com/

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

You can also go directly to the support web site at:

http://support.openview.hp.com/

HP OpenView online 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 valuable support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit and track progress on support cases
- Manage a support contract
- Look up HP support contacts
- Review information about available services
- Enter discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and log in. Many also require a support contract.

To find more information about access levels, go to:

http://support.openview.hp.com/access_level.jsp

To register for an HP Passport ID, go to:

https://passport2.hp.com/hpp/newuser.do

Contents

1 Preface

This guide describes how to use Open Metric Adapter (Open MA) to develop new Service Level Management (SLM) metric adapters.

Audience

This guide is intended for the person who is responsible for developing new SLM metric adapters using Open MA.

Prerequisites

Before reading this guide, it is helpful to read the *Service Level Management User Manual*.

In particular, make sure you understand the following concepts:

- Metric
- Metric definition
- Metric adapter
- Metric discovery
- Metric definition discovery
- Metric data collection
- Heartbeat polling
- Polled metric data values
- Event-based metric data values

Adapter versus Adaptor

In openadaptor, the spelling of the term “adaptor” (with an “o”) is similar to the spelling of the term “adapter” (with an “e”) in SLM. The two terms are *not* interchangeable. The term “adaptor” always refers the openadaptor concept. The term “adapter” always refers to the SLM concept.

License Notice

Open MA depends on the open source software called openadaptor, which in turn depends on other third-party packages. Make sure that you fully understand the licenses of those packages before you start to develop new metric adapters.

The licenses for openadaptor, and for those open source packages on which openadaptor depends, can be found in the following directory:

`${OV_HOME}/license-agreements/openadaptor`

In this path, `${OV_HOME}` indicates the HP OpenView installation directory.

Some packages included in openadaptor are not included in the Open MA package because of license issues or because they are not used in Open MA. If you need these packages, it is very easy to get them by installing the openadaptor package.

You can download openadaptor from the following website:

http://www.openadaptor.org/downloads.html

All packages of openadaptor can be found in the following directory:

`${OA_HOME}/classes`

In this path, `${OA_HOME}` indicates the openadaptor installation.

2 Open Metric Adapter

This chapter provides a brief introduction to Open Metric Adapter (Open MA).

Developing Metric Adapters

Open MA is a toolkit used to develop new metric adapters (MAs).

Some standard metric adapters are delivered with Open MA. These metric adapters can be used to collect metric data values from corresponding monitoring applications. For example, the OVIS metric adapter collects metric data values from HP OpenView Internet Services (OVIS). It is possible that Service Level Management (SLM) users want to collect metric data values from a monitoring application that is not supported by an existing metric adapter. For this reason, it is wise to give SLM users the power to develop specific metric adapters by themselves. Open MA is used for this purpose.

Monitoring Applications, Data Sources, and Metric Data

Monitoring applications are applications used to monitor business-critical applications and services (for example, web applications, mail services, and so on). HP OpenView Internet Services (OVIS) is a typical monitoring application.

The values of monitored metrics are available to metric adapters through data sources, such as databases, files, sockets, and so on. For example, OVIS stores the values of monitored metrics in the database. Metric adapters then collect metric data values from the database.

The data source varies from one monitoring application to another. For example, the data source of OVIS is a database, but the data source of HP OpenView Performance Manager (OVPM) is an HTTP service.

The metric data provided by monitor applications through data sources vary as well. Even if the data sources are similar (for example, if all data sources are Oracle Databases), the metric data of different monitor applications may have different data formats, data value types, data meanings, and so on.

Metric Adapter Developers

Metric adapter developers (MA developers) work with Open MA to develop new metric adapters. MA developers should have IT engineering knowledge (for example, knowledge of SQL, data types, XML, and so on). It is even better if MA developers have Java programming knowledge.

Open Metric Adapter and openadaptor

openadaptor is the key component that gives Open MA the power of a toolkit. When MA developers build new metric adapters, they work primarily with openadaptor.

For example, imagine you are going to develop a fancy metric adapter, called “FancyMA.” FancyMA collects metric data values from a flat file. The metric data values are the sum of two specific fields in each record.

Metric adapters do more than collect metric data. For example, they perform metric discovery, metric definition discovery, and so on. To keep things simple, we discuss only metric data collection in this section.

FancyMA would probably end up containing the following main code segments:

- **Code to read data from the input file**

  To write this code, you would need to know how to open the flat file, and then read each record. You would then need to know the format of each record of data. For example, you could read the data record as a comma-separated list of values or in XML format.

- **Code to calculate metric data values**

  You would need to know how to extract the two fields from each record, and how to calculate the sum of the two fields.

- **Code to send metric data values to SLM**

  You would need to know how to connect and send the metric data values to SLM.

FancyMA would look something like Figure 1.

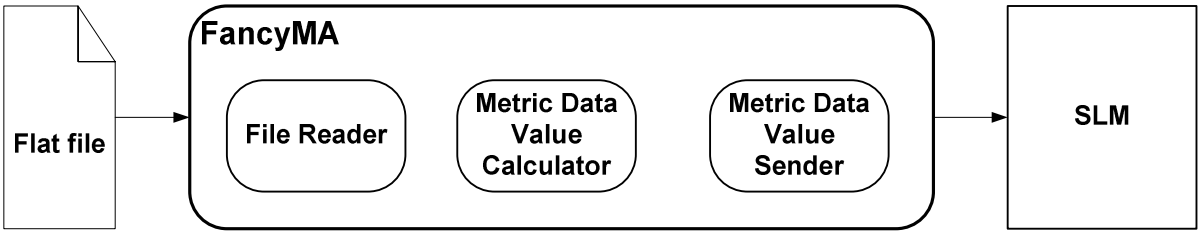

Figure 1: FancyMA

As you can see, the file reader reads records of the file. Records are then passed to the metric data value calculator. Finally, calculated metric data values are sent to SLM by the metric data value sender.

Two weeks later, you decide to develop another metric adapter, called “AnotherFancyMA”. This time, you want to read from an Oracle Database. The metric data values are stored in three columns for each row.

Intuitively, you realize that you can probably reuse much of FancyMA. You might need to alter the file reader to a database reader, and replace the metric data value calculator with the appropriate code to extract metric data values from each record.

AnotherFancyMA would look something like Figure 2.

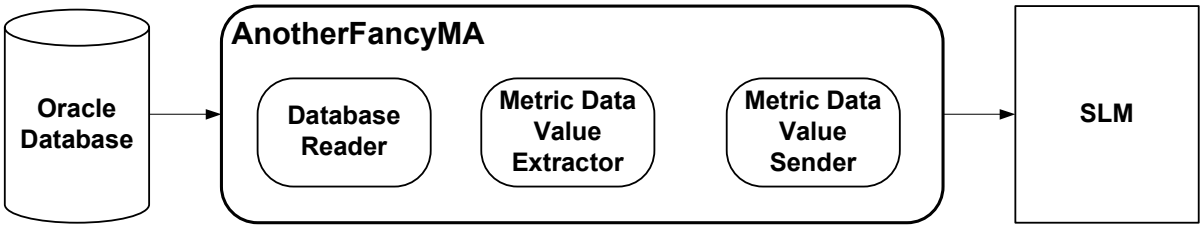

Figure 2: AnotherFancyMA

From FancyMA to AnotherFancyMA, you can see two changes:

- FancyMA file reader changes to the AnotherFancyMA database reader.
- FancyMA metric data value calculator changes to the AnotherFancyMA metric data value extractor.

Probably the most variable jobs performed by different metric adapters are the following, listed in sequential order:

1 Retrieve records of data from the external data source.

2 Analyze and process the retrieved records from which metric data values are to be obtained.

As you will see in Chapter 3, “openadaptor,” you can use openadaptor to perform the two jobs easily by using simple configuration files, rather than by writing actual program code.

After you integrate openadaptor, FancyMA looks like Figure 3.

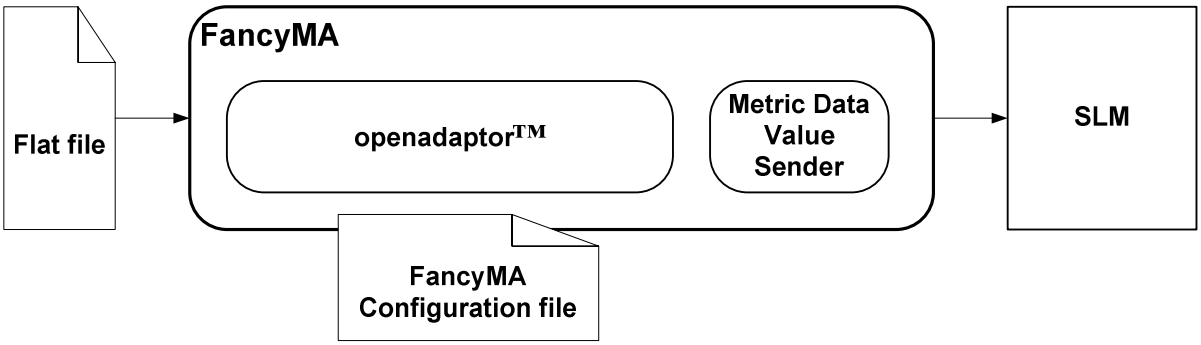

Figure 3: FancyMA Powered by openadaptor

After you integrate openadaptor, AnotherFancyMA looks like Figure 4.

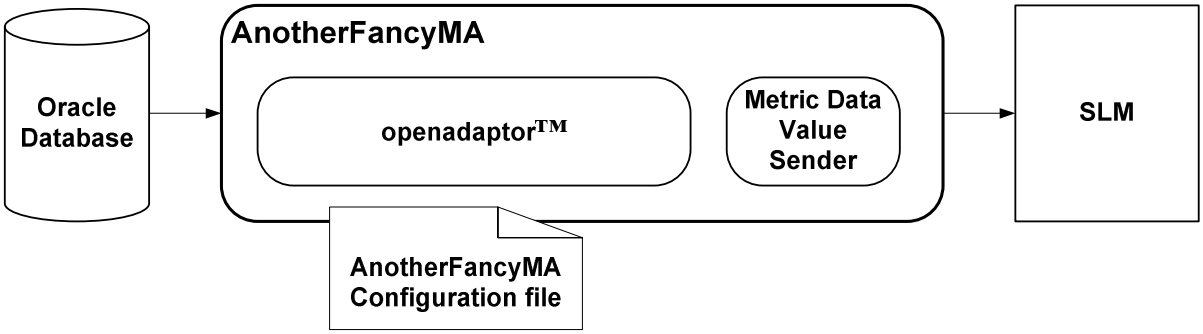

Figure 4: AnotherFancyMA powered by openadaptor

As you can see, after openadaptor is integrated, FancyMA and AnotherFancyMA have identical code segments. The only difference is their openadaptor configurations files. That is, you are able to develop a new metric adapter by writing a configuration file only, without a single line of Java code. This approach simplifies the development of new metric adapters.

As you will learn in Chapter 3, “openadaptor,” you occasionally have to do some programming to develop some openadaptor components. But that is rare.

Open Metric Adapter Configuration

For information about the configuration of the Open Metric Adapter, refer to the *Service Level Management User Manual*.

3 openadaptor

This chapter provides a brief introduction to openadaptor. It also explains where to get more information about openadaptor.

If you are already an openadaptor expert, you can skip this chapter.

Software Platform

According to the openadaptor website, “openadaptor is a Java/XML-based software platform which allows for rapid business system integration with little or no custom programming.”

This section describes two key characteristics of this software platform.

Toolkit for Rapid Business System Integration

openadaptor is a toolkit for rapid business system integration. openadaptor users generally develop applications that send messages between systems. For example, if you want to develop an application that reads records from a flat file, and then writes the records to a database, you may use openadaptor to implement it very rapidly.

Software Framework

openadaptor provides ready-built components, which you can assemble quickly to implement your specific application. Here “assemble” means doing simple configurations, rather than complex programming.

As a software framework, openadaptor is highly extensible. openadaptor users can create custom components to implement custom behavior as needed. For example, if you want to read data from a data source that is not supported by any available source components, you can develop a custom source component to read data from the data source.

In fact, openadaptor is extensible in many ways, including (but not limited to) sources, pipes, sinks, string readers, string writers, and so on.

Basic Concepts

openadaptor includes the following basic concepts:

- Adaptor
- Source
- String Reader
- Pipe
- Sink

- String Writer
- Controller
- DataObject

To learn about these concepts, refer to the *openadaptor Programmer's Guide*.

Source Components in Separate Threads

Each openadaptor source component runs as a separate thread of execution.

Proactive and Reactive Mode

When retrieving data, each openadaptor source component runs in one of the two distinct modes:

- **Proactive mode**

  In this mode, source components retrieve data from data sources proactively. All source components of the "Polling" type (for example, PollingSQLSource) run in this mode. A source component that runs in proactive mode is called a "proactive source."

- **Reactive mode**

  In this mode, source components wait reactively for the data sources to send data. All source components of the "Listening" type (for example, SocketSource) run in this mode. In addition, some source components of the "Callback" type (for example, JMSSource) ultimately run in this mode as well. A source component that runs in reactive mode is called a "reactive source."

For detailed information about "Polling," "Listening," and "Callback," refer to the *openadaptor Programmer's Guide*.

Utilities

This section describes the openadaptor Adaptor Framework Editor and display components.

Adaptor Framework Editor

The Adaptor Framework Editor is a graphical user interface (GUI) editor for the openadaptor framework. It simplifies the process of constructing a properties file and running an adaptor.

The Adaptor Framework Editor is shown in Figure 5.

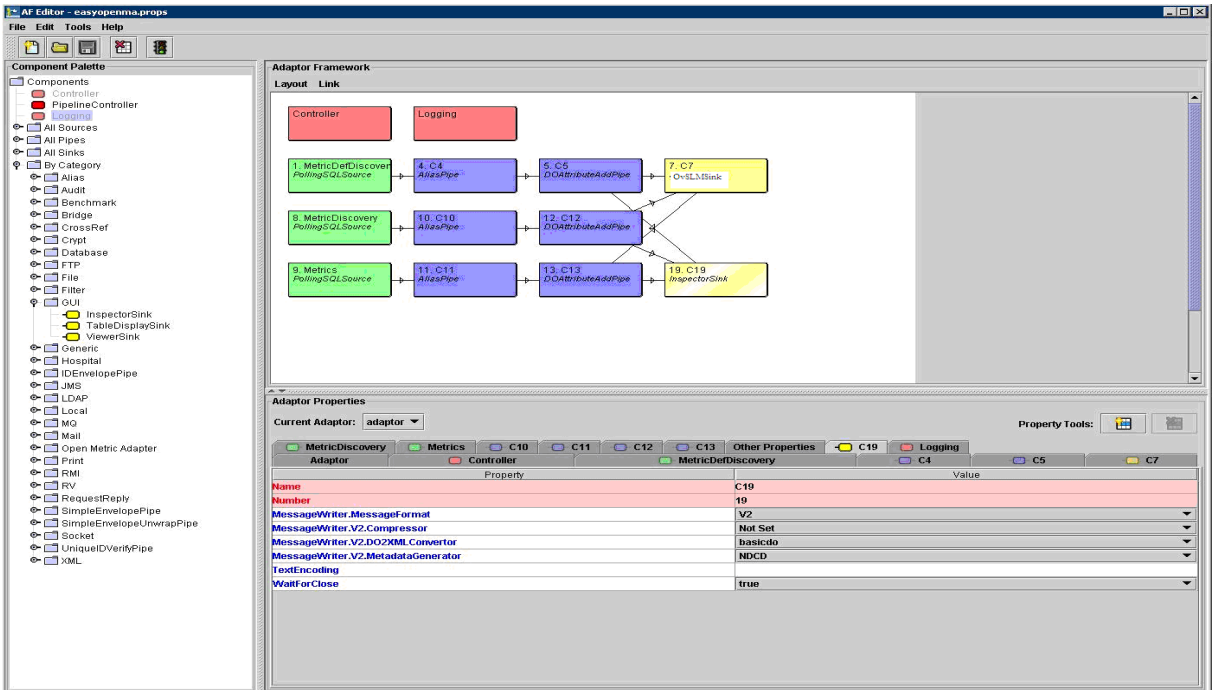

Figure 5: Adaptor Framework Editor

For more information about the Adaptor Framework Editor, refer to the *openadaptor Programmer's Guide*. openadaptor provide the class AFEditor, which allows you to run the Adaptor Framework Editor.

For your convenience, HP provides a front-end script to this AFEditor class that calls the AFEditor for you. In addition, an Open Metric Adapter-specific sink component, called OvSLMSink (see "OvSLMSink Properties") is added to the component palette.

The script is called `afeditor` (`.bat` or `.sh`), and is located in the following directory:

`$InstallDir``/nonOV/slm/openadaptor/`

In this path, *`$InstallDir`* indicates the HP OpenView installation directory.

Display Components

Display components make your work easier. They provide a GUI to show DataObjects as well as the attributes of the selected DataObject. Developers can use the display components to find out whether the DataObjects are constructed correctly.

InspectorSink is one of the display components. InspectorSink is available in the component palette of the Adaptor Framework Editor (in the category "GUI").

The InspectorSink GUI is shown in Figure 6.

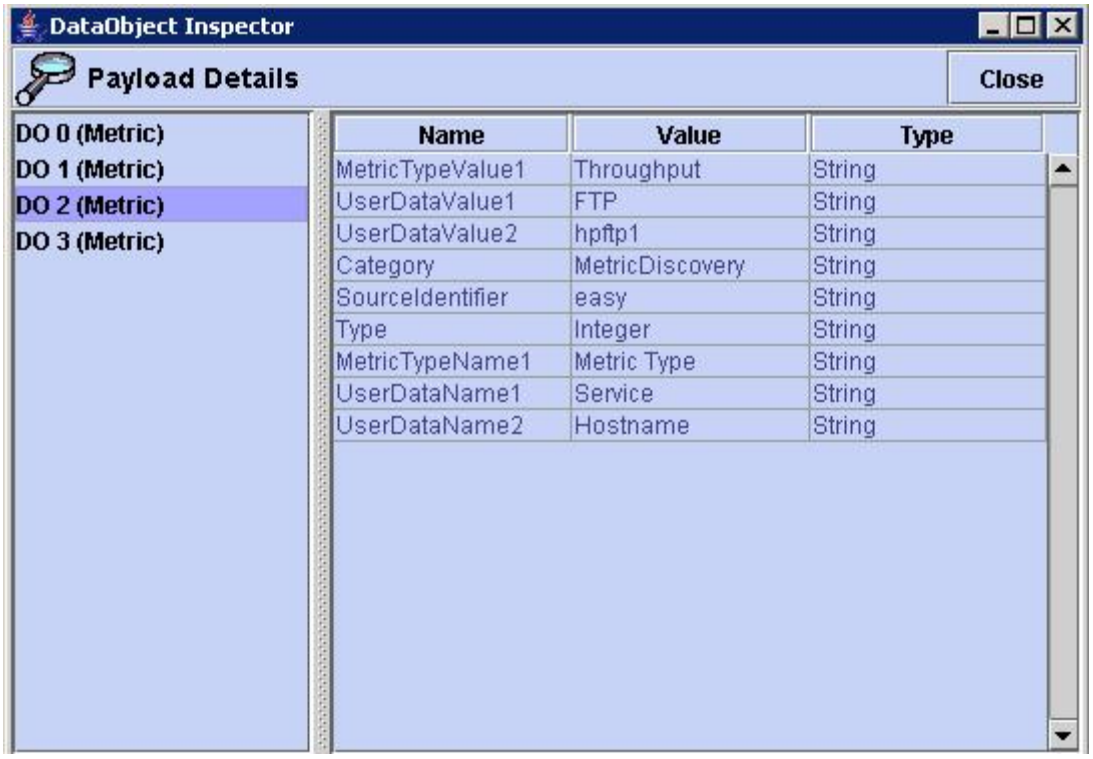

Figure 6: InspectorSink

ViewerSink is another display component. Unfortunately, ViewerSink is not available in the component palette of the Adaptor Framework Editor. So, in the Adaptor Framework Editor, InspectorSink is the only choice.

The ViewerSink GUI is shown in Figure 7.

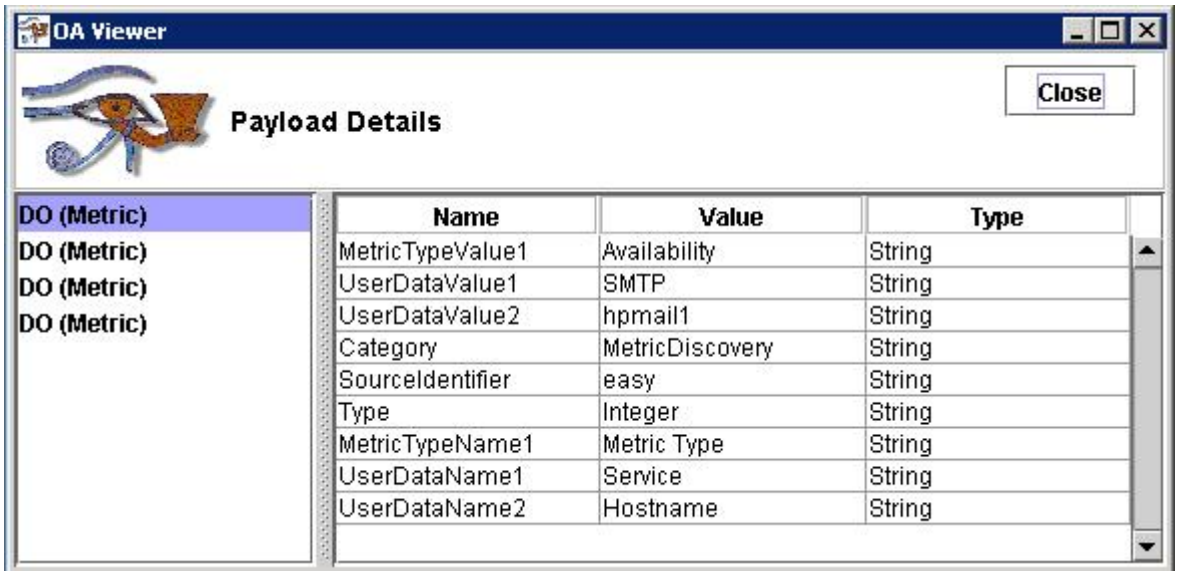

Figure 7: ViewerSink

For more information about ViewerSink, refer to the *openadaptor Programmer's Guide*.

The use of display components is as simple as making them the destination of the adaptor.

If your systems have no GUI environment, you may use other sink components (for example, FileSink).

Website

You will find more information about openadaptor on its web site:

www.openadaptor.org

This site provides full documentation, tutorials, mailing lists, discussion forums, access to source code (under CVS source control), and more.

License

The openadaptor license is a permissive, copyrighted variant of the MIT license.

You will find details about the openadaptor license on the following page:

www.openadaptor.org/licence.html

Software

After the Open MA toolkit is installed, files critical to the run time of openadaptor are installed. You are encouraged to install the openadaptor™ software. In the installation package, you will find more information about openadaptor (for example, openadaptor examples, cookbook, source code, javadocs, and so on).

Stable openadaptor builds are available on the following web page:

www.openadaptor.org/downloads.html

openadaptor 1.7.0 (the latest version at the time of this writing) is bundled in Service Level Management (SLM) 5.0.

Help

You will find openadaptor documentation on the following web page:

www.openadaptor.org/docs/DomainDocs.html

This documentation includes the following:

- *openadaptor Openadaptor FAQ*
- *openadaptor White Paper*
- *openadaptor Programmer's Guide*
- *openadaptor DataObjects Tutorial*
- openadaptor javadocs

If you are new to openadaptor, the *openadaptor Openadaptor FAQ* and the *openadaptor White Paper* are a good place to start. After you understand the basics, you can get more information from the *openadaptor Programmer's Guide* and the *openadaptor DataObjects Tutorial*. The openadaptor javadocs provide you with detailed information about the usage of each component. They are extremely useful even if you are already very familiar with openadaptor.

You will find the openadaptor discussion forum at the following location:

openadaptor.openadaptor.org/servlets/ForumMessageList?forumID=6

This is a good place to post questions and get answers from openadaptor experts.

After you install the openadaptor software, you will find many examples in the openadaptor subdirectories (for example, see `/cookbook` and `/examples`). These examples are good resources for learning how to use the ready-built components.

4 How Open MA Works

In the previous chapter, you learned basic information about Open Metric Adapter (Open MA). In this chapter, you will learn the internal structure of Open MA. More important, you will learn how openadaptor works within Open MA.

Architecture

The overall architecture of a metric adapter developed using Open MA is shown in Figure 8. The arrows indicate the direction of data flow.

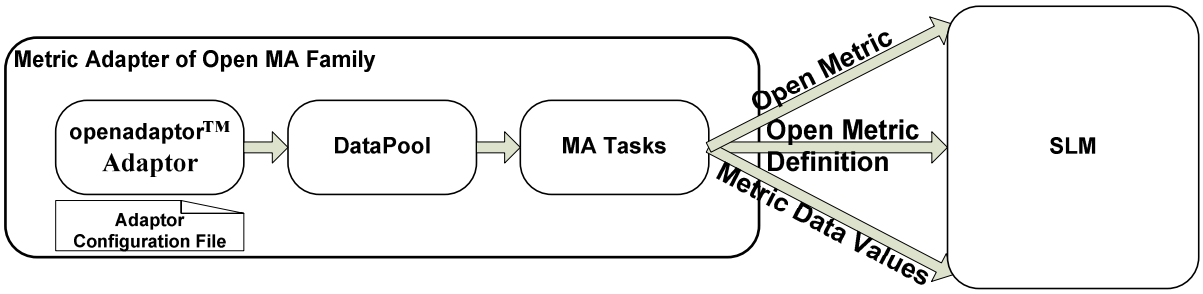

Figure 8: Open MA Overall Architecture

This architecture includes the following:

- **Open MA family**

  Any metric adapter developed using Open MA is called a metric adapter of the Open MA family.

- **Open Metric and Open Metric Definition**

  Any metric object whose metric data values are collected by the metric adapter of the Open MA family is called an Open Metric object.

  Open Metric is different from specific metrics, such as OVIS Metric. OVIS Metric is specific to the OVIS metric adapter and, ultimately, to OVIS itself. In contrast, Open Metric is not specific to any particular metric adapter. Because the OVIS metric adapter collects metric data from OVIS only, OVIS Metric represents only the model of metrics monitored by OVIS. But Open MA works with *any* monitoring application. Open Metric represents the model of *any* metric monitored by *any* monitoring application.

  The model of Open Metric is fixed. Developers are not allowed to change it. In "Open Metric and Open Metric Definition," you will learn how the fixed model of Open Metric is able to fit into the varying metric models.

  Each Open Metric Definition object specifies a type of Open Metric.

- **openadaptor adaptor**

  In each metric adapter developed using Open MA, there is one (and only one) openadaptor adaptor. The adaptor retrieves records of data from monitoring applications (openadaptor sources can be used for this job). Generally, the retrieved data cannot be understood by other components of Open MA. So the openadaptor adaptor must analyze the retrieved data, and convert them into standard formats that can be interpreted by other components of Open MA (openadaptor pipes can be used for this job). After reformatting, the data flow out of openadaptor, and are stored in DataPool.

- **Adaptor configuration file**

  The openadaptor configuration file defines the openadaptor adaptor.

- **DataPool**

  DataPool act as a “warehouse” in which the openadaptor output data is stored temporarily. Other parts of Open MA get data from DataPool to finish their jobs.

- **MA tasks**

  MA tasks are tasks common to all metric adapters (for example, metric discovery, metric definition discovery, metric data collection, and so on). MA tasks use the data stored in DataPool to do their jobs. For detailed information about metric discovery, metric definition discovery, and metric data collection, refer to the *Service Level Management User Manual*.

Open Metric and Open Metric Definition

This section explains how the fixed models of Open Metric and Open Metric Definition are able to fit into varying metric models.

Open Metric

Metrics have general attributes and attributes that are specific to a particular monitoring application. Those specific attributes vary when the metric adapter varies. For example, OVIS Metric has the specific attribute “Customer”, “Hostname”, “Service Name”, and so on. OVSN Metric has the specific attribute “Host”, “Label”, and so on. But the data model used for representing these varying monitoring-application-specific attributes must be fixed. The solution is to represent each monitoring-application-specific metric attribute with one pair of special attributes. One attribute specifies the value of the monitoring-application-specific metric attribute. The other attribute specifies the name of the monitoring-application-specific metric attribute. For example, the attribute “Label” in the OVSN metric is represented by one pair of attributes: “User Data Name” and “User Data Value”. The value of the attribute “User Data Value” is the value of the attribute “Label”. The value of the attribute “User Data Name” is the name of the attribute “Label”, which is the string value “Label”. These paired attributes are called **user-defined metric attributes**.

There are a total of six pairs of user-defined metric attributes. The names of each pair of user-defined metric attributes are “User Data Name *N*” and “User Data Value *N*”, where “*N*” may be 1, 2, 3, 4, 5, or 6. The choice of six pairs is a tradeoff between sufficiency and simplicity. If you need more attributes, you still have the possibility to merge the multiple attributes into a single user-defined metric attribute by using string utilities.

Example:

```
User Data Name 6 = "Country / Region / Town"; User Data Value 6 = "France /
Alpes Maritimes / Nice"
```

Open Metric attributes include general metric attributes and user-defined metric attributes.

Open Metric Definition

Like Open Metric, the fixed model of Open Metric Definition is able to fit into varying metric models. There are four pairs of user-defined metric definition attributes that are used to represent those monitoring-application-specific metric definition attributes. The names of each pair of the user-defined metric definition attributes are "Metric Type Name *N*" and "Metric Type Value *N*", where "*N*" may be 1, 2, 3, or 4.

Open Metric Definition attributes include general metric definition attributes, user-defined metric definition attributes, and the attribute "Source Identifier".

For details about "Source Identifier", refer to the *Service Level Management User Manual*.

How the Adaptor Works

The internal structure of an openadaptor adaptor is shown in Figure 9. The arrows indicate the direction of data flow.

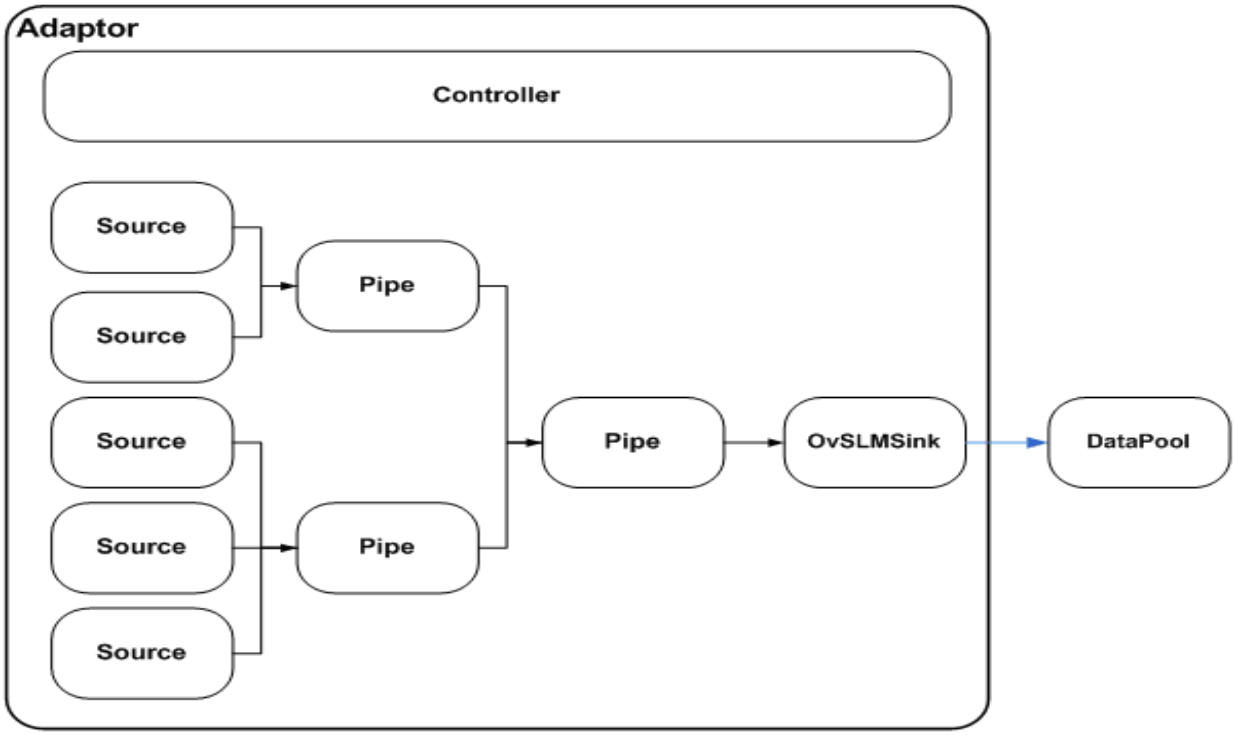

Figure 9: Adaptor Internal Structure

In general, the adaptor performs the following jobs:

1. Sources retrieve the data from monitoring applications.
2. Pipes transform the retrieved data into the standard data structures.
3. OvSLMSink puts the transformed data into DataPool.

During the process, the controller handles the flow of data, from sources to pipes, and ends in OvSLMSink.

In Open MA, the default controller, org.openadaptor.adaptor.SimpleController, is always used. You do not need to change the default configuration of the default controller.

The following sections explain the three jobs in details.

Retrieving Data

openadaptor sources are responsible for retrieving data from monitoring applications (for example, flat files, TCP/IP sockets, databases, and so on).

Each metric adapter collects five categories of data. Each category of data is used for a specific MA task.

There are five categories of data:

- **Metric Data Collection**

  Used for collecting the metric data values from the corresponding monitoring applications.

- **Metric Discovery**

  Used for identifying metrics monitored by the corresponding monitoring applications.

- **Metric Definition Discovery**

  Used for identifying metric definitions within the corresponding monitoring applications.

- **Location Discovery**

  Used for identifying locations within the corresponding monitoring applications. Because location discovery is not supported by the SLM server, this category of information is optional. It is alright if this category of information is not retrieved.

- **Status Collection**

  Used for collecting the status of corresponding monitoring applications. This category of information is optional. It is okay if this category of information is not retrieved.

It is a good practice to define five openadaptor sources in the openadaptor adaptor. Each openadaptor source collects one of the five categories of data. The reason is explained in the next section.

Sometimes, it is impossible to use five distinct sources to retrieve different types of data. For example, if the monitoring application provides all data through a single socket connection, only one source is needed.

Benefit of Using Five openadaptor Sources

It is possible to use only one source to collect all five categories of data. But using five sources has the benefit of separately controlling the data collection of each category. The benefit is important because, as a general rule, different sources retrieve data in different frequencies. For example, a metric adapter runs metric data collection (once every five minutes) more frequently than metric discovery (once an hour). As a result, the “metrics” source collects the metrics data more frequently than the “metric discovery” source collects the metric discovery data. If you use only one openadaptor source to collect the two categories of data, the openadaptor source has to retrieve the two categories of data at the same frequency (that is, the frequency with which the “metrics” source collects the metrics data). Obviously, such an approach is inefficient, and may degrade the performance of the metric adapter significantly.

Synchronizing openadaptor Sources and MA Tasks

Each openadaptor source runs in its own separate thread. So it is important to make sure the openadaptor adaptor and other Open MA components are synchronized. For example, if Open MA is configured to perform metric discovery every hour, and the openadaptor adaptor is configured to collect metric discovery data every two hours, then, in the first hour, no metric discovery data is collected, and Open MA cannot discover metrics. The openadaptor adaptor must collect data more frequently than Open MA. (The tasks of Open MA include location discovery, metric definition discovery, metric discovery, Metric Data Collection and status collection.)

It is not wise to let the openadaptor adaptor run so frequently that it degrades performance significantly. As a rule, the frequency of the openadaptor adaptor data collection should be

twice the frequency of the MA tasks. For example, if Open MA is configured to perform metric discovery every 60 minutes, the openadaptor adaptor should be configured to collect metric discovery data every 30 minutes.

This behavior of Open MA is subject to change. In a future version of Open MA, openadaptor and Open MA will be synchronized automatically, so MA users will not have to configure how frequently the openadaptor adaptor collects data.

Transforming Data

Generally, the data collected from monitoring applications are “raw.” They cannot be interpreted by MA tasks and SLM. To make the data “easy to digest,” you must “cook” it. openadaptor pipe components (for example, DOAttributeAddPipe, AliasPipe, and so on) are especially useful in performing this job.

In the previous section, you learned that each metric adapter requires five categories of data for “food.” For each category, there is a standard data structure. Each standard data structure is like a recipe. It tells the developer how to “cook” the data. Data that fail to follow the standard data structure are ignored.

Every data structure contains a set of attributes. For example, the data structure for the category “Location Discovery” has two attributes: “Category” and “Location”. Each attribute is a pair, with an attribute name and an attribute value. Each attribute name is associated with an openadaptor DataObject attribute type. The attribute names are case-sensitive. (For more information about the openadaptor DataObject attribute type, refer to the *openadaptor Programmer’s Guide*.) Except for one attribute, whose attribute type is “DateTime” or “Date”, the attribute type of all attributes is “String”.

In each data structure, some attributes are mandatory and some are optional. Each mandatory attribute must be assigned an attribute value. Optional attributes have no such constraint. The attribute values of optional attributes can be left unassigned. For example, in the data structure for the category “Metric Discovery”, the attribute “Type” is a mandatory attribute, and the attribute “UserDataName1” is an optional attribute.

Each of the five standard data structures has the attribute “Category”. This attribute specifies the name of the category to which the data belong.

The attribute value of the attribute “Category” must be one of the following five values:

- **Metrics**

  Indicates the category “Metric Data Collection”.

- **MetricDiscovery**

  Indicates the category “Metric Discovery”.

- **MrpDefDiscovery**

  Indicates the category “Metric Definition Discovery”.

- **LocationDiscovery**

  Indicates the category “Location Discovery”.

- **Status**

  Indicates the category “Status Collection”.

There are no spaces between the words in the attribute values.

The five standard data structures are explained in detail in the following sections.

Metric Definition Discovery

The “Metric Definition Discovery” data structure represents Open Metric Definition. (For details information about Open Metric Definition, see “Open Metric Definition.”) Each attribute (except for the attribute “Category”) of the data structure represents either one general metric definition attribute or one Open Metric Definition-specific attribute. For example, the data structure attribute “SourceIdentifier” represents the general metric definition attribute “Source Identifier”. The data structure attribute “MetricTypeValue1” represents the Open Metric Definition-specific attribute “Metric Type Value 1”.

All of the attributes in this data structure are listed in Table 1.

Table 1: Attributes of Metric Definition Discovery

Attribute Name	Attribute Type	Attribute Value	Optional
Category	String	“MrpDefDiscovery”	No
Type	String	See general metric definition attribute “Value Type”	No
Unit	String	See general metric definition attribute “Unit”	Yes
SourceIdentifier	String	See general metric definition attribute “Source Identifier”	Yes
MetricTypeValue1	String	See Open Metric Definition attribute “Metric Type Value 1”	Yes
MetricTypeName1	String	See Open Metric Definition attribute “Metric Type Name 1”	Yes
MetricTypeValue2	String	See Open Metric Definition attribute “Metric Type Value 2”	Yes
MetricTypeName2	String	See Open Metric Definition attribute “Metric Type Name 2”	Yes
MetricTypeValue3	String	See Open Metric Definition attribute “Metric Type Value 3”	Yes
MetricTypeName3	String	See Open Metric Definition attribute “Metric Type Name 3”	Yes
MetricTypeValue4	String	See Open Metric Definition attribute “Metric Type Value 4”	Yes

MetricTypeName4	String	See Open Metric Definition attribute “Metric Type Name 4”	Yes

An example of “Metric Definition Discovery” data is shown in Table 2.

Table 2: Example of Metric Definition Discovery Attributes

Attribute Name	Attribute Value
Category	MrpDefDiscovery
SourceIdentifier	OVIS
Type	Double
Unit	Percent
MetricTypeName2	Metric Type
MetricTypeValue2	AVAILABILITY
MetricTypeName4	Probe Type
MetricTypeValue4	HTTP

Metric Discovery

The “Metric Discovery” data structure includes all attributes of the data structure “Metric Definition Discovery”. All other attributes, including general metric attributes and Open Metric-specific attributes, are derived from Open Metric attributes. (For details information about Open Metric, see “Open Metric.”)

All of the attributes in this data structure are listed in Table 3.

Table 3: Attributes of Metric Discovery

Attribute Name	Attribute Type	Attribute Value	Optional
Category	String	“MetricDiscovery”	No
Type	String	See general metric definition attribute “Value Type”	No
SourceIdentifier	String	See general metric definition attribute “Source Identifier”	Yes
Unit	String	See general metric definition attribute “Unit”	Yes
MetricTypeValue1	String	See Open Metric Definition attribute “Metric Type Value 1”	Yes
MetricTypeName1	String	See Open Metric Definition attribute “Metric Type Name 1”	Yes
MetricTypeValue2	String	See Open Metric Definition attribute “Metric Type Value 2”	Yes

Attribute Name	Attribute Type	Attribute Value	Optional
MetricTypeName2	String	See Open Metric Definition attribute “Metric Type Name 2”	Yes
MetricTypeValue3	String	See Open Metric Definition attribute “Metric Type Value 3”	Yes
MetricTypeName3	String	See Open Metric Definition attribute “Metric Type Name 3”	Yes
MetricTypeValue4	String	See Open Metric Definition attribute “Metric Type Value 4”	Yes
MetricTypeName4	String	See Open Metric Definition attribute “Metric Type Name 4”	Yes
Location	String	See general metric attribute “Location”	Yes
MrpName	String	See general metric attribute “Name”	Yes
UserDataValue1	String	See Open Metric attribute “User Data Value 1”	Yes
UserDataName1	String	See Open Metric attribute “User Data Name 1”	Yes
UserDataValue2	String	See Open Metric attribute “User Data Value 2”	Yes
UserDataName2	String	See Open Metric attribute “User Data Name 2”	Yes
UserDataValue3	String	See Open Metric attribute “User Data Value 3”	Yes
UserDataName3	String	See Open Metric attribute “User Data Name 3”	Yes
UserDataValue4	String	See Open Metric attribute “User Data Value 4”	Yes
UserDataName4	String	See Open Metric attribute “User Data Name 4”	Yes

Attribute Name	Attribute Type	Attribute Value	Optional
UserDataValue5	String	See Open Metric attribute “User Data Value 5”	Yes
UserDataName5	String	See Open Metric attribute “User Data Name 5”	Yes
UserDataValue6	String	See Open Metric attribute “User Data Value 6”	Yes
UserDataName6	String	See Open Metric attribute “User Data Name 6”	Yes

If you want to use the location filter (not yet supported in Service Level Management 5.0) to reduce the amount of metrics identified during metric discovery, you may not leave the attribute “Location” empty. For details about the location filter, refer to the *Service Level Management User Manual*.

In the data structure “Metric Discovery”, the attribute “MrpName” is derived from the general metric attribute “Name”. The general metric attribute “Name” specifies the name of the metric.

If the attribute “MrpName” is not assigned a value (that is, if the metric is not assigned a name), the metric adapter generates a name for the metric automatically, according to the values of certain attributes.

To generate metric names, the following rule is used:

```
<SourceIdentifier> [ "_" + <MetricTypeValue1> ]
[ + "_" + <MetricTypeValue2> ] [ + "_" + <MetricTypeValue3> ]
[ + "_" + <MetricTypeValue4> ] [ + "_" + <UserDataValue1> ]
[ + "_" + <UserDataValue2> ] [ + "_" + <UserDataValue3> ]
[ + "_" + <UserDataValue4> ] [ + "_" + <UserDataValue5> ]
[ + "_" + <UserDataValue6> ]
```

In this rule, the following is true:

- <*SourceIdentifier*> is the value of the attribute “SourceIdentifier”.
- <MetricTypeValue*N*> is the value of the attribute “MetricTypeValue*N*” (*N* = 1, 2, 3, or 4). If the value of the attribute <MetricTypeValue*N*> is empty, the underscore (_) that immediately precedes it is omitted.
- <*UserDataValueN*> is the value of attribute “UserDataValue*N*” (*N* = 1, 2, 3, 4, 5, or 6). If the value of the attribute <UserDataValue*N*> is empty, the underscore (_) that immediately precedes it is omitted.

If the attribute “MrpName” is assigned a non-empty value, the value is the name of the metric. In this case, developers are free to determining how to name the metric.

An example of “Metric Discovery” data that do not use the attribute “MrpName” to specify the metric name is shown in Table 4.

Table 4: Example of Metric Discovery Data without MrpName

Attribute Name	Attribute Value
Category	MetricDiscovery
SourceIdentifier	OVIS
Type	Double
Unit	Percent
MetricTypeName1	Metric Type
MetricTypeValue1	AVAILABILITY
MetricTypeName2	Probe Type
MetricTypeValue2	HTTP
Location	*www.hp.com*
UserDataName1	System
UserDataValue1	*www.hp.com*
UserDataName2	Host
UserDataValue2	webserver
UserDataName3	Target
UserDataValue3	webserver

In this example of metric discovery data, the metric name is to be generated by the metric adapter as follows:

```
OVIS_AVAILABILITY_HTTP_www.hp.com_webserver_webserver
```

An example of “Metric Discovery” data that explicitly specify the metric name using the attribute “MrpName” is shown in Table 5.

Table 5: Example of Metric Discovery Data with MrpName

Attribute Name	Attribute Value
Category	MetricDiscovery
SourceIdentifier	OVIS
Type	Double
Unit	Percent
MetricTypeName1	Metric Type
MetricTypeValue1	AVAILABILITY
MetricTypeName2	Probe Type
MetricTypeValue2	FTP
Location	*ftp.hp.com*
UserDataName1	System
UserDataValue1	*ftp.hp.com*
UserDataName2	Host
UserDataValue2	ftpserver
UserDataName3	Target
UserDataValue3	Ftpserver
MrpName	Metric011

In this example of metric discovery data, the name of the metric is “Metric011”, which is the value of the attribute “MrpName”.

Metric Data Collection

The “Metric Data Collection” data structure includes all attributes of the data structure “Metric Discovery” plus two attributes, “Value” and “Timestamp”, both of which are derived from the general metric attributes.

In contrast with other attributes, the attribute type of the attribute “Timestamp” may be “DateTime” or “Date”.

All of the attributes in this data structure are listed in Table 6.

Table 6: Attributes of Metric Data Collection

Attribute Name	Attribute Type	Attribute Value	Optional
Category	String	“Metrics”	No
Value	String	See general metric attribute “Value”	No
Timestamp	DateTime or Date	See general metric attribute “Timestamp”	Yes
Type	String	See general metric definition attribute “Value Type”	Yes
Unit	String	See general metric definition attribute “Unit”	Yes
SourceIdentifier	String	See general metric definition attribute “Source Identifier”	Yes
MetricTypeValue1	String	See Open Metric Definition attribute “Metric Type Value 1”	Yes
MetricTypeName1	String	See Open Metric Definition attribute “Metric Type Name 1”	Yes
MetricTypeValue2	String	See Open Metric Definition attribute “Metric Type Value 2”	Yes
MetricTypeName2	String	See Open Metric Definition attribute “Metric Type Name 2”	Yes
MetricTypeValue3	String	See Open Metric Definition attribute “Metric Type Value 3”	Yes
MetricTypeName3	String	See Open Metric Definition attribute “Metric Type Name 3”	Yes

Attribute Name	Attribute Type	Attribute Value	Optional
MetricTypeValue4	String	See Open Metric Definition attribute “Metric Type Value 4”	Yes
MetricTypeName4	String	See Open Metric Definition attribute “Metric Type Name 4”	Yes
Location	String	See general metric attribute “Location”	Yes
MrpName	String	See general metric attribute “Name”	Yes
UserDataValue1	String	See Open Metric attribute “User Data Value 1”	Yes
UserDataName1	String	See Open Metric attribute “User Data Name 1”	Yes
UserDataValue2	String	See Open Metric attribute “User Data Value 2”	Yes
UserDataName2	String	See Open Metric attribute “User Data Name 2”	Yes
UserDataValue3	String	See Open Metric attribute “User Data Value 3”	Yes
UserDataName3	String	See Open Metric attribute “User Data Name 3”	Yes
UserDataValue4	String	See Open Metric attribute “User Data Value 4”	Yes
UserDataName4	String	See Open Metric attribute “User Data Name 4”	Yes
UserDataValue5	String	See Open Metric attribute “User Data Value 5”	Yes
UserDataName5	String	See Open Metric attribute “User Data Name 5”	Yes
UserDataValue6	String	See Open Metric attribute “User Data Value 6”	Yes

Attribute Name	Attribute Type	Attribute Value	Optional
UserDataName6	String	See Open Metric attribute “User Data Name 6”	Yes

Each piece of "Metric Data Collection" data must contain the following types of information:

- **Information that uniquely identifies the metric**

  In metric adapters, each metric is uniquely identified by its metric name. There are two ways to determine the metric name. One way is to explicitly provide the metric name, using the attribute "MrpName". The other way is to let Open MA automatically generate the metric name, according to the value of certain attributes. All attributes required for computing the metric name must be given so that the metric name can be computed correctly.

- **Information about the value of the metric**

  Two attributes are involved: "Value" and "Timestamp". The attribute "Value" indicates the most recent value of the metric. The attribute "Timestamp" indicates when the most recent value is collected. If attribute "Value" has no value assigned, or if the assigned value is invalid, based on the value type of the metric, the metric adapter ignores the value. If the attribute "Timestamp" has no value assigned, or if the assigned value is invalid, the metric adapter assigns the current system time to it.

An example of "Metric Data Collection" data that do not use the attribute "MrpName" to specify the metric name is shown in Table 7.

Table 7: Example of Metric Data Collection Data without MrpName

Attribute Name	Attribute Value
Category	Metrics
SourceIdentifier	OVIS
MetricTypeValue1	AVAILABILITY
MetricTypeValue2	HTTP
UserDataValue1	*www.hp.com*
UserDataValue2	webserver
UserDataValue3	webserver
Value	0.99
Timestamp	2005-1-1 00:00:00 000

The value type of the attribute "Timestamp" is "DateTime".

An example of "Metric Data Collection" data that explicitly specify the metric name using the attribute "MrpName" is shown in Table 8.

Table 8: Example of Metric Data Collection Data with MrpName

Attribute Name	Attribute Value
Category	Metrics
MrpName	Metric011
Value	0.99

Location Discovery

The “Location Discovery” data structure includes only two attributes: “Category” and “Location”. Of the two attributes, “Location” is derived from the Open Metric attribute “Location”. All of the attributes in this data structure are listed in Table 9.

Table 9: Attributes of Location Discovery

Attribute Name	Attribute Type	Attribute Value	Optional
Category	String	“LocationDiscovery”	No
Location	String	See general metric attribute “Location”	No

An example of “Location Discovery” data is shown in Table 10.

Table 10: Example of Location Discovery Data

Attribute Name	Attribute Value
Category	LocationDiscovery
Location	*www.hp.com*

Status

The “Status” data structure includes only two attributes: “Category” and “Status”. The attribute “Status” specifies information about the status of the monitoring application. All of the attributes in this data structure are listed in Table 11.

Table 11: Attributes of Status

Attribute Name	Attribute Type	Attribute Value	Optional
Category	String	“Status”	No
Status	String	The information about the status of the monitoring application	No

An example of “Status” data is shown in Table 12.

Table 12: Example of Status Data

Attribute Name	Attribute Value
Category	Status
Status	available

Putting Data into DataPool

DataPool is the destination of all data flowing in the openadaptor adaptor. An openadaptor sink component, named “com.hp.ov.sd.slm.sa.openma.OvSLMSink”, is responsible for putting data into DataPool. In the openadaptor adaptor, OvSLMSink is always the destination of all the pipelines.

OvSLMSink Properties

The sink component OvSLMSink has two properties:

- **DataPoolTimeLimit**

  This property indicates the period, in minutes, within which DataPool keeps the data it receives. For example, if the value of "DataPoolTimeLimit" is 60, DataPool keeps the data it receives within the last 60 minutes. A value of zero ("0") means that DataPool keeps all the data, regardless of when DataPool received it. If this property is not defined, its default value is "0", which means no limit.

- **DataPoolIsEventBased**

  If the metric adapter collects event-based metric data values, the value of this property must be "true". Otherwise, the value must be "false". If this property is not defined, its default value is "false".

The larger the value of "DataPoolTimeLimit", the more memory is to be retained by DataPool. It is not recommended to set a large value for "DataPoolTimeLimit" because only recent data are useful to the metric adapter. The memory retained by non-recent data is, in fact, wasted. As a rule, the value of "DataPoolTimeLimit" should be identical to the largest interval among the five MA tasks (see "Five Important Tasks"). For example, if the largest interval among the five MA tasks is 60 minutes, the value of "DataPoolTimeLimit" is recommended to be 60 minutes.

How MA Tasks Work

Metric discovery and metric data values collection are two important tasks.

In the following two sections, you will learn the following:

- What the five important tasks are
- How metric discovery detects metrics currently being monitored by the monitoring application
- Values of polled metric data and event-based metric data

Five Important Tasks

The five important tasks are the following:

- **Metric Data Collection**

  Metric adapters run this task to collect metric data values.

- **Metric Discovery**

  Metric adapters run this task to identify discovered metrics.

- **Metric Definition Discovery**

  Metric adapters run this task to identify discovered metric definitions.

- **Location Discovery**

  Metric adapters run this task to identify discovered locations.

- **Status Collection**

  Metric adapters run this task to collect their own status and, optionally, the status of the corresponding monitoring applications.

As a general rule, the five tasks run in periodic-polling mode. The exception is the Metric Data Collection task, which may run in event-based mode. (For details about event-based mode, see "Polled Metric Data Values and Event-based Metric Data Values.") The metric adapter runs each task interval after interval. The interval is defined as the time elapsed between the startups of two consecutive procedures. The time spent on the task is counted into the interval.

Detecting Metrics Currently Being Monitored

Metric discovery is used to keep track of the metrics currently being monitored by the monitoring application.

To detect metrics being monitored by the monitoring application, there are two methods for metric discovery:

- **Get the status of the metrics directly**

  This method is applicable if the metric adapter can directly monitor whether the metrics are currently monitored by the monitoring application. This is the simplest method, but it is not applicable to all monitoring applications because some monitoring applications do not explicitly indicate whether the metric is currently monitored. The following method is applicable to such monitoring applications.

- **Use "DiscoveryMaxHistory" to filter out metrics not being monitored**

  Service Level Management (SLM) 5.0, Beta 2, is marked as "for OvisMA only" in the Metric Adapter form, but it is used for Open MA as well. "DiscoveryMaxHistory" is a general metric adapter configuration setting. This configuration setting defines a time filter with which metric adapters are able to distinguish monitored metrics from unmonitored metrics. For detailed information about "DiscoveryMaxHistory", refer to the *Service Level Management User Manual*. This method is applicable only if the metric adapter *cannot* explicitly get the status of metrics in the monitoring application.

If you get the status of the metrics directly, the value of "DiscoveryMaxHistory" must be set to 0.

The parameter "DiscoveryMaxHistory" is also applicable to metric definition discovery and location discovery.

Polled Metric Data Values and Event-based Metric Data Values

Metric adapters can collect polled metric data values and event-based metric data values. For polled values, the configured polling interval determines the collection frequency. For event-based values, a metric adapter collects event-based metric data values as soon as the monitoring application recognizes that an event (of new value) has occurred. For details about polled values and event-based values, refer to the *Service Level Management User Manual*.

During the development of new metric adapters using Open MA, apply the following rules:

- **Polled metric data values**

  If the metric adapter is required to collect polled metric data values, apply these rules:

  — Both proactive sources and reactive sources can be used for collecting metric data values. For detailed information about proactive and reactive sources, see “Proactive and Reactive Mode.”

  — The value of the metric adapter parameter “isEventBased” must be “0”. For details about the parameter “isEventBased”, refer to the *Service Level Management User Manual*.

  — The value of the OvSLMSink property “DataPoolIsEventBased” must be “false”. For details about the property “DataPoolIsEventBased”, see “OvSLMSink Properties.”

- **Event-based metric data values**

  If the metric adapter is required to collect event-based metric data values, follow these rules:

  — Only the reactive source can be used for collecting metric data values. For details about the proactive and reactive sources, see “Proactive and Reactive Mode.”

  — The metric adapter parameter “isEventBased” must be set to “1”. For details about the parameter “isEventBased”, refer to the *Service Level Management User Manual*.

  — The value of the OvSLMSink property “DataPoolIsEventBased” must be “true”. For details about the property “DataPoolIsEventBased”, see “OvSLMSink Properties.”

How DataPool Works

Data output from openadaptor are stored in DataPool. Other components in Open MA retrieve data from DataPool, and use the data to finish tasks (for example, metric discovery and so on).

As a general rule, Open MA is used only for recent metric data values. So it is wise to let DataPool keep recent metric data only. Developers are able to define which data are “recent data,” using the DataPool configuration setting. For details about the DataPool configuration setting, see “Display Components.”

Open Metric Adapter Configuration

Different metric adapters developed using Open Metric Adapter can be deployed and run on the same host. But it is not possible to start multiple instances for an individual metric adapter.

5 How to Create New Metric Adapters Using Open MA

This chapter explains how to create new metric adapters using Open Metric Adapter (Open MA). It guides you step by step through the whole development process.

Before you start this chapter, make sure you have read Chapters 3 and 4, understand how Open MA works, and understand how to construct openadaptor adaptors using openadaptor components (sources, pipes, and sinks).

Creating Metric Adapters

When creating a new metric adapter, you work primarily with openadaptor to construct an adaptor. That adaptor is able to retrieve data from the corresponding monitoring application, and transform the data into standard data structures that can be understood by Service Level Management (SLM). The construction of the adaptor is embodied in an openadaptor configuration file.

After you finish the openadaptor configuration file, most of your work is done. The remaining work consists of tasks, such as creating the metric adapter configuration files, creating scripts for installing and configuring the new metric adapter in HP OpenView Control (OVC), and so on. Templates of configuration files and scripts are provided, and the tasks are kept as simple as possible.

The life cycle of creating a new metric adapter can be divided into five steps. In each step, you complete several tasks.

The life cycle consists of the following steps:

1. **Gathering requirements**

   In this step, you develop an image of what the new metric adapter should look like. You gather information about behaviors of the new metric adapter (for example, from which monitoring application the metric data values are supposed to be collected, which metrics are supposed to be collected, and so on).

2. **Designing the adapter**

   In this step, you draw a blueprint for the new metric adapter. You determine how to organize the data collected from the monitoring application, which (source and pipe) components to use, how to connect the components, and so on.

3. **Producing the adapter**

   In this step, you implement the blueprint. After you finish this step, you will have produced an openadaptor configuration file. The configuration file defines the openadaptor adaptor that collects data from the monitoring application, and transforms the data into the standard data structures.

   The adaptor framework editor is a GUI editor that makes your work easier. It simplifies the process of creating the configuration file by freeing you from labor-intensive text editing.

Occasionally, you may even develop your own openadaptor source components or pipe components in this step.

4 **Testing the adapter**

In this step, you verify whether the adaptor implemented in the previous step works as expected. Check whether data can be retrieved from the data sources, and whether the data are correctly transformed according to the standard data structures. In addition, pay attention to other aspects of the implementation (for example, performance, portability, and so on). The openadaptor ViewerSink is very useful in this step.

5 **Releasing the adapter**

The final step is never minor. In this step, you package the new metric adapter for distribution. With the package, end users must be able to easily install and configure the new metric adapter in their run-time environments.

The first four steps focus on constructing the adaptor. The last step focuses on other supporting tasks. The following sections explain these five steps in detail.

Step 1 – Gathering Requirements

In this step, you gather critical information about the new metric adapter.

Monitoring Application

First of all, you must have knowledge about the monitoring application from which metric data values are collected. The monitoring application may be an HP service-monitoring application (for example, OVIS) or a non-HP service-monitoring application.

Metric Types

You must do some research on the metrics monitored by the monitoring application, and understand which pieces of information are critical for identifying the metrics. For example, with OVIS, the critical information includes “Metric Type”, “Probe Type”, “Customer”, “Hostname”, “Service Name”, “System” and “Target”. Some pieces of information are regarded as monitoring-application-specific metric attributes. Other pieces of information are regarded as monitoring-application-specific metric definition attributes.

To distinguish between metric attributes and metric definition attributes, follow this general rule:

- **Metric attribute**

  If the piece of information specifies the source of metric data values (for example, “Hostname”), it is a metric attribute.

- **Metric definition attribute**

  If the piece of information specifies the type of metric (for example, “Probe Type”), it is a metric definition attribute.

Each monitoring-application-specific metric attribute is represented by one pair of user-defined metric attributes. Likewise, each monitoring-application-specific metric definition attribute is represented by one pair of user-defined metric definition attributes. For details about user-defined metric attributes, see “Open Metric.” For details about user-defined metric definition attributes, see “Open Metric Definition.”

Value Types

The value type of the metric data values is fixed.

Possible value types are the following:

- Integer
- Double (double-precision real values)
- Timespan (duration)
- Percentage (double-precision real values no less than 0 and no greater than 100)

For more information about those value types, refer to the *Service Level Management User Manual*.

Data Sources

After studying the monitoring application, you can figure out which data sources to access for the metric data values. Frequently used data sources are database, file, socket, and so on. For example, the data source of OVIS is database. After that, it is a good practice to gather further details about the data source. The details are useful in subsequent steps. For example, if you find that the data source is a database server, you should probe deeper, and find out in which tables the required information is stored, what the structure of each table is, and other details.

Sometimes, the same monitoring applications have different data sources. OVIS is a good example. Its data source may be an Oracle Database or a Microsoft SQL Server database. In this case, you must be careful with the SQL statements used for accessing the database during implementation of the metric adapter. The SQL statements are quite different in Oracle Database and in Microsoft SQL Server. The SQL statements that work well with Microsoft SQL Server will probably fail with Oracle Database. If only standard SQL statements are used, the metric adapter can work with both Oracle Database and Microsoft SQL Server. Otherwise, you have to create two versions of metric adapters: one for Oracle Database and the other for Microsoft SQL Server.

It is also possible that the monitoring application has multiple data sources. For example, HP OpenView Operations for Windows (OVOW) has a database and Windows Management Instrumentation (WMI).

In this case, you have to do further research on each data source to answer the following questions:

- What information can be collected from the data source?

  It is possible that the data source does not provide all of the required information.

- Is there a standard openadaptor source component that can work with the data source?

  It is possible that no standard openadaptor source component can work with the data source. That is, you have to develop your own openadaptor source component to work with the data source.

- Is there any restriction when accessing the data source?

  For example, accessing the data source may not be permitted by the security policy.

Answers to these questions should help you make your decision. Decisions should be made case by case.

Step 2 – Designing the Adapter

In this step, you create the blueprint of the new metric adapter.

Naming the New Metric Adapter

The name of the new metric adapter must be unique. It is a good practice to define a descriptive name, so users can recognize the metric adapter easily. The name of the monitoring application should appear in the adapter name. If the metric adapter is specific to one of the possible data sources, the name of the data source must appear in the adapter name too.

When you name the new metric adapter, it is recommended that you follow this rule:

<name of monitoring application> + *<(optional) name of data source>* + "OpenMA"

When running, metric adapters generate some files with underscores (_) in their names. It is a good practice to avoid underscores in the metric adapter name.

Example 1: If the name of the monitoring application is "MP1", and the adapter is not specific to any data source, the name of the new metric adapter is "MP1OpenMA".

Example 2: If the name of the monitoring application is "MP2", and the adapter is specific to Oracle Database, the name of the new metric adapter is "MP2OracleOpenMA".

The metric adapter name is case-sensitive.

Organizing the Attributes

In Step 1, you found information specific to metrics monitored by the monitoring application. In the current step, you define which user-defined attributes (metric or metric definition) to represent which piece of monitoring-application-specific information (metric or metric definition). The result is a mapping between the monitoring-application-specific attribute and the user-defined attribute representing it.

For example, in Step 1, you found the OVIS-specific metric definition attributes "Metric Type" and "Probe Type", as well as the OVIS-specific metric attributes "Customer", "Hostname", "Service Name", "System", and "Target".

The mapping is shown in Table 13.

Table 13: First Example of OVIS-specific Attribute Mappings

Use	To Represent
MetricTypeName1 MetricTypeValue1	Metric Type
MetricTypeName2 MetricTypeValue2	Probe Type
UserDataName1 UserDataValue1	Customer
UserDataName2 UserDataValue2	Hostname
UserDataName3 UserDataValue3	Service Name
UserDataName4 UserDataValue4	System
UserDataName5 UserDataValue5	Target
Unit	Unit

The mappings in Table 14 would work as well.

Table 14: Second Example of OVIS-specific Attribute Mappings

Use	To Represent
MetricTypeName2 MetricTypeValue2	Metric Type
MetricTypeName4 MetricTypeValue4	Probe Type
UserDataName6 UserDataValue6	Customer
UserDataName1 UserDataValue1	Hostname
UserDataName5 UserDataValue5	Service Name
UserDataName3 UserDataValue3	System
UserDataName2 UserDataValue2	Target
Unit	Unit

There is no rule for choosing which pair of user-defined attributes should represent which monitoring-application-specific attributes. Developers are free to make their own choices. In

the first example, the attribute pair “MetricTypeName2” and “MetricTypeValue2” are used to represent “Probe Type”. In the second example, they are used to represent “Metric Type”.

Selecting Sources

Based on the data sources from which you decided to collect data in Step 1, you now select the openadaptor source components to be used. Table 15 lists the frequently used openadaptor source components for different data sources. Although these openadaptor source components are used frequently, they are not the only possible source components.

Table 15: Frequently Used Source Components

Data Source	Frequently Used openadaptor Source
Database	PollingSQLSource
File	AdvancedFileSource FileSource
Socket	SocketSource
JMS Service	JMSSource
FTP Service	FTPSource
Mail Service	MailSource
LDAP Service	LDAPSource
Web Service	WebServiceSource

If there is no openadaptor source that can work with the data source, you should consider writing your own source components in the next step. Think twice before writing your own source components. There may already be some source components that meet your requirement. If you are not sure, consult openadaptor experts (for example, submit your questions to the openadaptor forum).

Drawing a Blueprint

This task is the core task of Step 2. In this task, you define which components are used, and how they are connected to construct the adaptor.

The process of designing the openadaptor adaptor involves many variables. The data sources, data formats, and openadaptor components are variable. For the sake of simplicity, this section discusses the process of drawing a blueprint of the adaptor in a general manner. In the real world, you will most likely encounter issues that are specific to your organization. If you encounter specific issues, you may find examples (see Chapter 6, “Examples”) and technical tips (see Chapter 7, “Best Practices”) helpful.

In Step 1, you determined which source components to use. In “Retrieving Data,” you learned that, as a general rule, there should be five sources, each collecting one category of data. In “OvSLMSink Properties,” you learned that OvSLMSink is always the destination of all the pipelines.

The initial version of the blueprint would look like Figure 10.

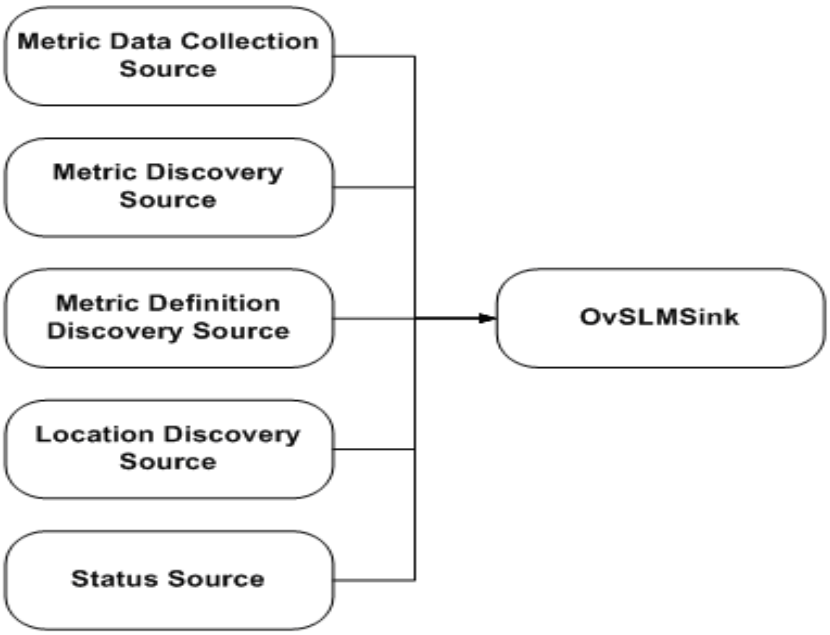

Figure 10: Initial Version of the Blueprint

- Of the five sources, the location discovery source and the status source are optional.
- In this step, you should focus on choosing the correct components, and creating the correct connection between the components. Other tasks, such as configuring the components, are performed in Step 3.

If data that flows out of the five sources fits into the standard data structures, as explained in "Putting Data into DataPool," the task is done. But, most of the time, you are not so lucky. The data from sources are "raw" data, so you have to do some "cooking." At this point, you try to transform the "raw" data using openadaptor pipe components.

In each standard data structure, the attribute "Category" specifies the category to which the data belong. Generally, the "raw" data does not have this attribute. So you have to add it. A standard openadaptor pipe component, DOAttributeAddPipe, can help. This pipe adds new attributes to the data passing through it. As a result, five DOAttributeAddPipes are added to the blueprint. Each DOAttributeAddPipe adds the attribute "Category" with different values. The attribute value is the category name of the data collected by the source preceding the pipe.

After you add the five DOAttributeAddPipes, the blueprint looks like Figure 11.

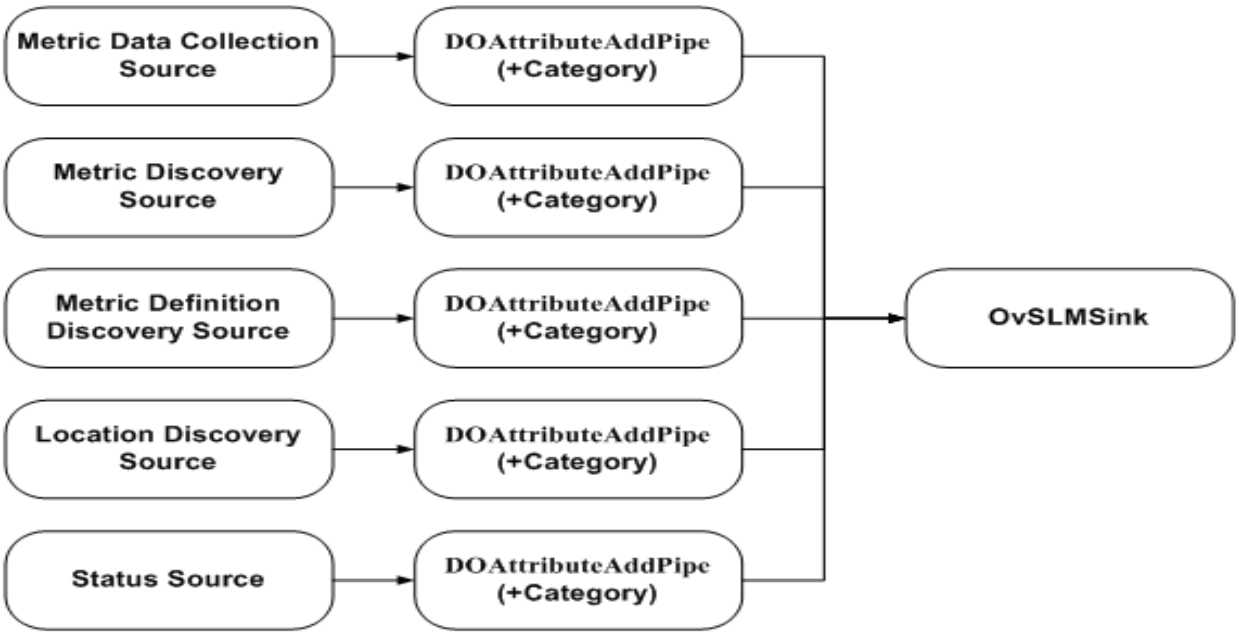

Figure 11: Second Version of the Blueprint

In addition to the attribute “Category”, other attributes (for example, “Type” and “SourceIdentifier”) generally need to be added as well. Because each DOAttributeAddPipe is able to add multiple attributes, you simply change the configuration of previously added DOAttributeAddPipes to add more attributes.

After you change the configuration of DOAttributeAddPipe, the blueprint looks like Figure 12.

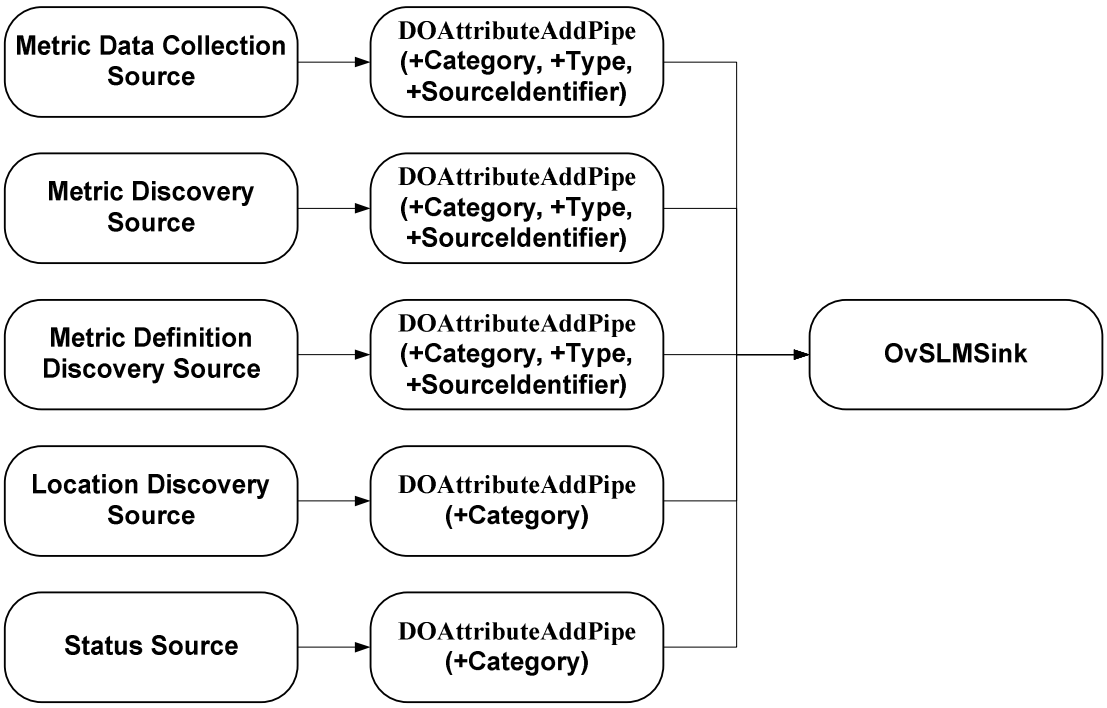

Figure 12: Third Version of the Blueprint

As you learned in "Open Metric and Open Metric Definition" and "Putting Data into DataPool," those monitoring-application-specific attributes (metric or metric definition) are represented by the user-defined attributes (metric or metric definition). The representing was defined in the previous task, as described in "Metric Types." You conduct the process of representing by following the table, which describes the mapping between the monitoring-application-specific attributes (metric or metric definition) and the user-defined attributes (metric or metric definition).

The process of representing includes two jobs. In the first job, you rename each monitoring-application-specific attribute to the name of the mapped user-defined attribute. There are two user-defined attributes mapped to each monitoring-application-specific attribute: one represents the attribute value (for example, UserDataValue1), and the other represents the attribute name (for example, UserDataName1). You must rename the monitoring-application-specific attribute to the user-defined attribute representing the value of the monitoring-application-specific attribute (for example, UserDataValue1). The standard openadaptor component AliasPipe can be used for renaming. You can find detailed information about AliasPipe in openadaptor documentation.

After you rename the attribute, the blueprint looks like Figure 13.

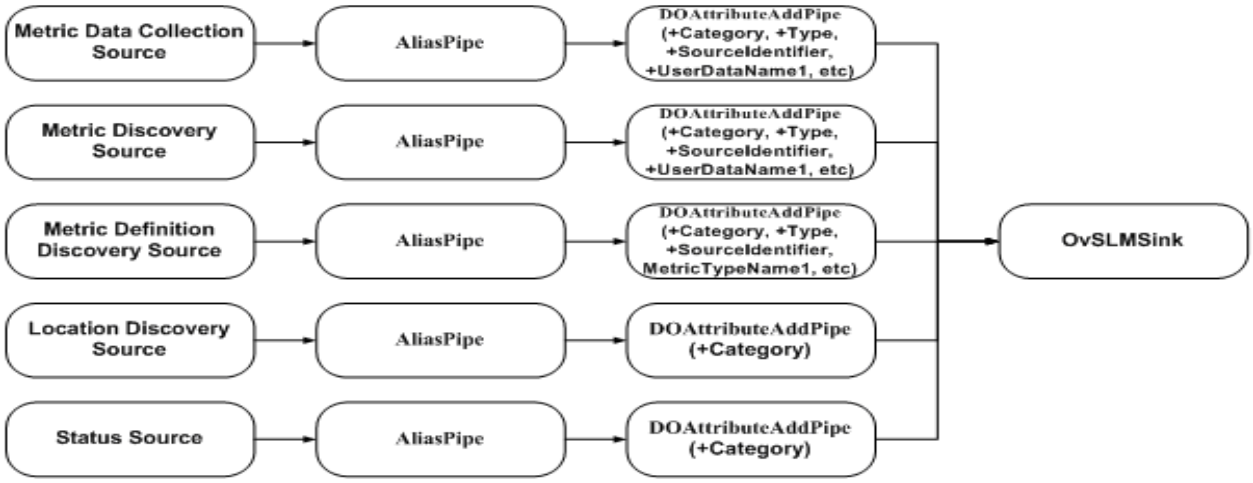

Figure 13: Final Version of the Blueprint

In Figure 13, you can see that the user-defined attributes (for example, UserDataName1) that represent the attribute names of the monitoring-application-specific attributes are added by the DOAttributeAddPipes.

The AliasPipes are added before the DOAttributeAddPipes to simplify the configuration of AliasPipe. For details, see Chapter 7, “Best Practices.”

As a second job in the process of representing, you must give each component (source, pipe, and sink) a unique name, which is used to identify the component. The name of the adaptor must be identical to the name of the new metric adapter. Otherwise, the new metric adapter is not able to work, even if the openadaptor adaptor works well.

At this point in the generic process, you have completed the blueprint. But, in the real world, you may transform the data and pipe components in a more sophisticated fashion. Components other than DOAttributeAddPipe and AliasPipe may be involved. For details, see Chapter 7, “Best Practices,” and Chapter 8, “Feedback and Troubleshooting.”

It is possible that the standard openadaptor pipe components are not able to perform the data transformation as you expect. In such cases, you should consider writing your own pipes. To find out how to write your own pipes, refer to the *openadaptor Programmer’s Guide*. Even though the special pipe component is not currently available, you are able to continue with the blueprint. You may use a different symbol to indicate that custom pipe component.

Step 3 – Producing the Adapter

In this step, you implement the blueprint, and produce a working metric adapter. When you finish this step, you will have created an openadaptor configuration file and some metric adapter configuration files.

Implementing the Adapter

In this step, the biggest challenge is to create a correct configuration file, based on the blueprint you created in the previous step.

You must correctly configure each component. You must correctly configure the relationship between the components. And you must correctly perform other configurations (for example, the configurations of DataPool, the openadaptor Controller, and so on).

In Open MA, the default controller, org.openadaptor.adaptor.SimpleController, is always used. You do not need to change the default configuration of the default controller. In AFEditor, you do not have to pay any attention to the configuration of the controller.

AFEditor

Adaptor Framework Editor (AFEditor) is introduced in "Adaptor Framework Editor." As a rule, this tool is more productive than text editors. AFEditor is especially useful for creating the links between components. Using text editors to create the links between components is more labor-intensive and more error-prone than using AFEditor.

For more tips about using AFEditor, see Chapter 7, "Best Practices."

Variable Substitution

Using variable substitution is a good practice for the parameters that will probably be changed after development (for example, the hostname and port number of the database service, the username and password of the database account, and so on).

For details about variable substitution, refer to the *openadaptor Programmer's Guide*.

Display Components

After you finish the configuration file, you can run the adaptor to see if it is correctly configured. If you are using AFEditor, you can run the adaptor by clicking a button.

The data stored in DataPool is not visible from the outside. Display components (see "Display Components") are useful for making the execution result of the adaptor visible. Using a display component is as simple as making it another destination of the adaptor.

Figure 14 illustrates a common use case of InspectorSink.

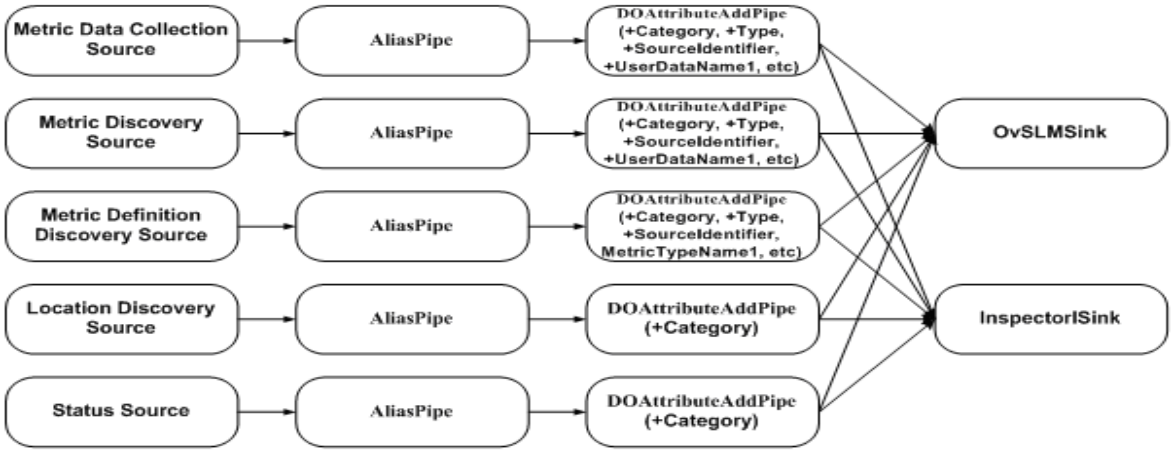

Figure 14: Common Use Case of InspectorSink

Configuration File Name

The openadaptor configuration file must be named according to following schema:

<name of the metric adapter> + ".props"

The file name *<name of the metric adapter>* is defined in Step 2. For example, if the adapter name is "FooOpenMA", the file name of the adaptor configuration file is "FooOpenMA.props".

If you fail to follow the schema, the new metric adapter will not work, even if the openadaptor adaptor itself works well. In this case, the new metric adapter will not work because that the name of the metric adapter is used for creating the file name of the adaptor configuration file.

Creating the Metric Adapter Configuration File

The configuration settings of each metric adapter are stored in the metric adapter configuration file.

The configuration file can be created easily by using the template configuration file, which is located in the following directory:

$DataDir/conf/slm/OpenMATemplate.xml

In this path, *$DataDir* indicates the OpenView data directory.

First, make a copy of the template file, and name the new file according to the following schema:

<name of the metric adapter> + ".xml"

The file name *<name of the metric adapter>* is defined in Step 2. For example, if the adapter name is "FooOpenMA", the configuration file name is "FooOpenMA.xml".

If you fail to follow the schema, the new metric adapter will not work well. It will not work well because the name of the metric adapter is used for creating the file name of the metric adapter configuration file.

The content of the template configuration file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config>
    <MA name="$METRIC_ADAPTER_NAME$">
        <ServerHost>$SLM_HOSTNAME$</ServerHost>
        <DefaultTaskPollingPeriod>60</DefaultTaskPollingPeriod>
        <DiscoveryInterval>600</DiscoveryInterval>
        <MrpDefinitionDiscoveryInterval>600</MrpDefinitionDiscoveryInterval>
        <LocationDiscoveryInterval>600</LocationDiscoveryInterval>
        <HeartBeatsInterval>300</HeartBeatsInterval>
        <DiscoveryMaxHistory>60</DiscoveryMaxHistory>
        <DataPointVersionByte>1</DataPointVersionByte>
        <TypeByte>1</TypeByte>
        <isEventBased>0</isEventBased>
        <SequenceNumber>0</SequenceNumber>
        <DefaultTaskExpirePeriod>600</DefaultTaskExpirePeriod>
        <DataPointSynchronizationDelay>10</DataPointSynchronizationDelay>
        <Publisher.RESPONSE_TIMEOUT>60</Publisher.RESPONSE_TIMEOUT>
    </MA>
    <Connector name="OpenConnector">
        <Class>com.hp.ov.sd.slm.sa.openma.OpenConnector</Class>
        <MaxLatency>3</MaxLatency>
    </Connector>
    <DiscoveryLocationFilter>
        <All/>
    </DiscoveryLocationFilter>
</Config>
```

You are required to modify the metric adapter configuration file. Modify the “name” attribute of the element `/Config/MA` to be the name of the metric adapter. The part that needs to be modified is marked as **`$METRIC_ADAPTER_NAME$`**.

`$SLM_HOSTNAME$` indicates the host where the corresponding SLM server is running. End users are required to replace it with the name of the host where the SLM server is running. It is recommended that you use the Metric Adapter Configuration GUI to set the parameter.

If the name of the metric adapter is “FooOpenMA”, after the modification, the content of the configuration file becomes the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config>
    <MA name="FooOpenMA">
        <ServerHost>$SLM_HOSTNAME$</ServerHost>
        <DefaultTaskPollingPeriod>60</DefaultTaskPollingPeriod>
        <DiscoveryInterval>600</DiscoveryInterval>
        <MrpDefinitionDiscoveryInterval>600</MrpDefinitionDiscoveryInterval>
        <LocationDiscoveryInterval>600</LocationDiscoveryInterval>
        <HeartBeatsInterval>300</HeartBeatsInterval>
        <DiscoveryMaxHistory>60</DiscoveryMaxHistory>
        <DataPointVersionByte>1</DataPointVersionByte>
        <TypeByte>1</TypeByte>
        <isEventBased>0</isEventBased>
        <SequenceNumber>0</SequenceNumber>
        <DefaultTaskExpirePeriod>600</DefaultTaskExpirePeriod>
        <DataPointSynchronizationDelay>10</DataPointSynchronizationDelay>
        <Publisher.RESPONSE_TIMEOUT>60</Publisher.RESPONSE_TIMEOUT>
    </MA>
    <Connector name="OpenConnector">
        <Class>com.hp.ov.sd.slm.sa.openma.OpenConnector</Class>
        <MaxLatency>3</MaxLatency>
    </Connector>
    <DiscoveryLocationFilter>
        <All/>
    </DiscoveryLocationFilter>
</Config>
```

The modified value is marked in **BLACK**.

Creating the Metric Adapter Control File

The metric adapter control file is used for configuring the metric adapter in HP OpenView Control (OVC). After the configuration, end users are able to use the command “ovc” to start and stop the metric adapter.

To create the metric adapter control file, perform these tasks:

1 Define the metric adapter.

2 Make a copy of the template file.

3 Modify the metric adapter configuration file.

Task 1 – Define the Metric Adapter

First, you must define the following:

1 **OVC component name of the metric adapter**

In OVC, each component has a unique name that is used to address it. The name is an ASCII identifier. Normally, the component name is lowercase. In OVC, the component name is case-sensitive.

2 **OVC component label of the metric adapter**

In OVC, each component has a label that is used when printing the status of a component. For example the component “opcle” would have a label “Log File Encapsulator”.

The control file can be created easily using the template control file, which is located in the following directory:

$DataDir/conf/slm/OpenMACtrlTemplate.xml

In this path, *$DataDir* indicates the HP OpenView data directory.

Task 2 – Make a Copy of the Template File

Second, you must make a copy of the template file, and name the new file according to following schema:

<name of the metric adapter> + “Ctrl.xml”

In this schema, *<name of the metric adapter>* is defined in Step 2. For example, if the adapter name is “FooOpenMA”, the configuration file name is “FooOpenMACtrl.xml”.

The content of the template control file is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ovc:OvCtrl xmlns:ovc="http://openview.hp.com/xmlns/ctrl/registration/1.5">
  <ovc:Component>
    <ovc:Name>$COMPONENT_NAME$</ovc:Name>
    <ovc:Label>
      <ovc:String>$COMPONENT_LABEL$</ovc:String>
    </ovc:Label>
    <ovc:Category>SLM</ovc:Category>
    <ovc:Options>
      <ovc:AllowAttach>true</ovc:AllowAttach>
      <ovc:AutoRestart>true</ovc:AutoRestart>
      <ovc:AutoRestartLimit>5</ovc:AutoRestartLimit>
      <ovc:AutoRestartMinRuntime>60</ovc:AutoRestartMinRuntime>
      <ovc:AutoRestartDelay>5</ovc:AutoRestartDelay>
      <ovc:AutoShutdown>false</ovc:AutoShutdown>
      <ovc:AutoShutdownTimer>5</ovc:AutoShutdownTimer>
      <ovc:StartAtBootTime>false</ovc:StartAtBootTime>
      <ovc:MentionInStatus>true</ovc:MentionInStatus>
      <ovc:Monitored>true</ovc:Monitored>
      <ovc:IsContainer>false</ovc:IsContainer>
      <ovc:PollingInterval>30</ovc:PollingInterval>
      <ovc:WorkingDirectory>$OvDataDir</ovc:WorkingDirectory>
    </ovc:Options>
    <ovc:ProcessDescription>java</ovc:ProcessDescription>
    <ovc:CommandLine>-D$COMPONENT_NAME$</ovc:CommandLine>
    <ovc:OnHook>
      <ovc:Name>START</ovc:Name>
      <ovc:Actions>
        <ovc:Start>
          <ovc:CommandLine>"$OvJreDir\java" -D$component -Xrs -cp "$InstallDir\java\OvSlmMaOpen.jar"
com.hp.ov.sd.slm.sa.openma.OpenMain $METRIC_ADAPTER_NAME$</ovc:CommandLine>
        </ovc:Start>
      </ovc:Actions>
    </ovc:OnHook>
    <ovc:OnHook>
      <ovc:Name>STOP</ovc:Name>
      <ovc:Actions>
        <ovc:Execute>
          <ovc:CommandLine>"$OvJreDir\java" -D$component -Xrs -cp "$InstallDir\java\OvSlmMaOpen.jar"
com.hp.ov.sd.slm.sa.openma.OpenMain $METRIC_ADAPTER_NAME$ -stop</ovc:CommandLine>
        </ovc:Execute>
      </ovc:Actions>
    </ovc:OnHook>
  </ovc:Component>
</ovc:OvCtrl>
```

Task 3 – Modify the Metric Adapter Configuration File

Third, you are required to modify the metric adapter configuration file as follows:

1. In the value of the following element, replace **$COMPONENT_NAME$** with the component name of the new metric adapter:

   ```
   /ovc:OvCtrl/ovc:Component/ovc:Name
   ```

2. In the value of the following element, replace **$COMPONENT_LABEL$** with the component label of the new metric adapter:

   ```
   /ovc:OvCtrl/ovc:Component/ovc:Label/ovc:String
   ```

3. In the value of the following element, replace **$COMPONENT_NAME$** with the component name of the new metric adapter:

   ```
   /ovc:OvCtrl/ovc:Component/ovc:CommandLine
   ```

   For example, if the component name is “fooopenma”, the value is `-D`**`fooopenma`**.

   There is no white space between “-D” and “fooopenma”.

4. In the value of the following element, replace **$METRIC_ADAPTER_NAME$** with the name of the new metric adapter:

   ```
   /ovc:OvCtrl/ovc:Component/ovc:OnHook/ovc:Actions/ovc:Start/ovc:CommandLine
   ```

   For example, if the metric adapter name is “FooOpenMA”, the value is as follows:

   ```
   "$OvJreDir\java" -D$component -Xrs -cp "$InstallDir\java\OvSlmMaOpen.jar"
   com.hp.ov.sd.slm.sa.openma.OpenMain FooOpenMA
   ```

Do *not* remove the existing system properties definition “-D$component”. If you do, OVC will not be able to correctly manage the metric adapter.

If the name of the metric adapter is “FooOpenMA”, after the modification, the content of the control file becomes:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ovc:OvCtrl xmlns:ovc="http://openview.hp.com/xmlns/ctrl/registration/1.5">
  <ovc:Component>
    <ovc:Name>fooopenma</ovc:Name>
    <ovc:Label>
      <ovc:String>Foo Open Metric Adapter</ovc:String>
    </ovc:Label>
    <ovc:Category>SLM</ovc:Category>
    <ovc:Options>
      <ovc:AllowAttach>true</ovc:AllowAttach>
      <ovc:AutoRestart>true</ovc:AutoRestart>
      <ovc:AutoRestartLimit>5</ovc:AutoRestartLimit>
      <ovc:AutoRestartMinRuntime>60</ovc:AutoRestartMinRuntime>
      <ovc:AutoRestartDelay>5</ovc:AutoRestartDelay>
      <ovc:AutoShutdown>false</ovc:AutoShutdown>
      <ovc:AutoShutdownTimer>5</ovc:AutoShutdownTimer>
      <ovc:StartAtBootTime>false</ovc:StartAtBootTime>
      <ovc:MentionInStatus>true</ovc:MentionInStatus>
      <ovc:Monitored>true</ovc:Monitored>
      <ovc:IsContainer>false</ovc:IsContainer>
      <ovc:PollingInterval>30</ovc:PollingInterval>
      <ovc:WorkingDirectory>$OvDataDir</ovc:WorkingDirectory>
    </ovc:Options>
    <ovc:ProcessDescription>java</ovc:ProcessDescription>
    <ovc:CommandLine>-Dfooopenma</ovc:CommandLine>
    <ovc:OnHook>
      <ovc:Name>START</ovc:Name>
      <ovc:Actions>
        <ovc:Start>
          <ovc:CommandLine>”$OvJreDir\java” -D$component -Xrs -cp
“$InstallDir\java\OvSlmMaOpen.jar” com.hp.ov.sd.slm.sa.openma.OpenMain
FooOpenMA</ovc:CommandLine>
        </ovc:Start>
      </ovc:Actions>
    </ovc:OnHook>
  </ovc:Component>
</ovc:OvCtrl>
```

The modified values are marked in **BLACK**.

Sometimes, you have to add new jar files to the Java Virtual Machine (JVM) classpath. For example, the metric adapter uses some custom components that are developed by you.

It is recommended that you copy the new jar files into the following directory:

$InstallDir\nonOV\slm\

In this path, *$InstallDir* indicates the HP OpenView installation directory. You have to manually update the value of the element <ovc:CommandLine> to add the required jar files to the classpath.

For example, if two new jar files, "new1.jar" and "new2.jar", are involved, and the two files are located in the directory *$InstallDir*\nonOV\slm\, the command line becomes:

```
<ovc:CommandLine>"$OvJreDir\java" -D$component -Xrs -cp
"$InstallDir\java\OvSlmMaOpen.jar;$InstallDir\nonOV\slm\new1.jar;$InstallDir\nonOV\slm\new2.jar"
com.hp.ov.sd.slm.sa.openma.OpenMain FooOpenMA</ovc:CommandLine>
```

You may need to add some system properties as well. Also, update the value of the element <ovc:CommandLine> to add the required system properties.

For example, if two system properties, "org.omg.CORBA.ORBInitialHost=localhost" and "org.omg.CORBA.ORBInitialPort=3700", are involved, the command line becomes:

```
<ovc:CommandLine>"$OvJreDir\java" -D$component -Dorg.omg.CORBA.ORBInitialHost=localhost -
Dorg.omg.CORBA.ORBInitialPort=3700 -Xrs -cp "$InstallDir\java\OvSlmMaOpen.jar" com.hp.ov.sd.slm.sa.openma.OpenMain
FooOpenMA</ovc:CommandLine>
```

You *must* always add your own system properties after the existing system properties definition "-D$component". If you fail to do so, OVC is not able to correctly manage the metric adapter.

Step 4 – Testing the Adapter

In this step, you verify whether the adaptor implemented in the previous step works as expected.

Running the Metric Adapter

To run the metric adapter, perform these tasks:

1. Copy the configuration and control files.
2. Modify the configuration file.

Task 1 – Copy the Configuration and Control Files

First, you must copy the configuration and control files to the following directory:

$DataDir/conf/slm/

In this path, *$DataDir* indicates the HP OpenView data directory.

You must copy the following files:

- Adaptor configuration file you developed in Step 3
- Configuration file of the metric adapter
- Control file of the metric adapter

Formally, the start and stop of the metric adapter must be managed by HP OpenView Control (OVC).

To configure the metric adapter in OVC, you must run the following command:

ovcreg -add "$DataDir/conf/slm/<file name of the control file>"

For example, if the file name of the control file is "FooOpenMACtrl.xml" and the HP OpenView data directory is `/var/OV/data`, the command is:

ovcreg -add "/var/OV/data/conf/slm/FooOpenMACtrl.xml"

Task 2 – Modify the Configuration File

Second, you modify the configuration file of the metric adapter, as follows:

1. Update the value of parameter `$SLM_HOSTNAME$` to the hostname or IP address of the corresponding SLM server (for example, localhost).
2. Update the values of other parameters you expect.

   For details about metric adapters configuration settings, refer to the *Service Level Management User Manual*.

To start the metric adapter and see if it works well, run the following command:

ovc -start <OVC component name of the metric adapter>

To stop the metric adapter, run the following command:

ovc -stop <OVC component name of the metric adapter>

To restart the metric adapter, run the following command:

ovc -restart <OVC component name of the metric adapter>

To see the status of the metric adapter, run the following command:

ovc -status <OVC component name of the metric adapter>

For testing purpose, you can start the metric adapter using following commands:

- **Windows**

  startOpenMA.bat <name of the metric adapter>

- **UNIX**

  startOpenMA.sh <name of the metric adapter>

The only required argument is the name of the metric adapter.

The command "startOpenMA.bat" or "startOpenMA.sh" is only for testing purposes. "ovc" is the formal tool.

For information about the configuration of the Open MA, refer to the *Service Level Management User Manual*.

Viewing the Log File

The log files of all the metric adapters are located in the following directory:

$InstallDir/data/log

In this path, *$InstallDir* indicates the HP OpenView installation directory.

The file name of the log file is defined as follows:

<name of the metric adapter> + *<instance NO.>* + "." + *<file NO.>* + "." + *<locale name>*

In this file name, *<instance NO.>* indicates the serial number of the metric adapter running instance. Generally, its value is 0. If, by mistake, the metric adapter has multiple running instances, its value might be greater than 0.

In the file name, *<file NO.>* indicates the serial number of the log file.

In the file name, *<locale name>* indicates current locale of the runtime environment (for example, en_US).

openadaptor log records of levels FATAL, ERROR, and WARN are redirected to the log file of the metric adapter.

Step 5 – Releasing the Adapter

In the previous steps, you successfully created a working metric adapter. In this step, you package it so it can be distributed and installed easily.

Packaging the Adapter

The package must contain the following three files:

- openadaptor adaptor configuration file
- Configuration file of the metric adapter
- Control file of the metric adapter

It is also important to provide end users with some guidance about the following:

- Purpose of (the monitoring application and the data source) of the metric adapter
- How to perform configurations specific to the metric adapter (for example, where to set the username and password of the database account)

Creating an Installer and Uninstaller

It is a good practice to create an installer and uninstaller for the metric adapter.

Create an Installer

To create an installer, follow these steps:

1. Copy the required files into the corresponding directories.

   The openadaptor adaptor configuration file, the metric adapter configuration file, and the metric adapter control file must be copied into the following directory:

   `$DataDir/conf/slm/`

   In this path, *`$DataDir`* indicates the HP OpenView data directory.

2. Configure the metric adapter in HP OpenView Control (OVC) using the command **`ovcreg`**.

Create an Uninstaller

To create an uninstaller, follow these steps:

1. Remove files copied during installation.

   These files include the openadaptor adaptor configuration file, the metric adapter configuration file, and the metric adapter control file in the following directory:

   `$DataDir``/conf/slm/`

   In this path, *`$DataDir`* indicates the HP OpenView data directory.

2. Remove the metric adapter component from HP OpenView Control (OVC) using the following command:

   `ovcreg -del <component name of the metric adapter>`

3. Remove other files generated by the metric adapter during run time:

   a. *`<metric adapter name>`* + "`_discovered.props`".

      For example, if the metric adapter name is "FooOpenMA", the file name is "FooOpenMA_discovered.props".

   b. *`<metric adapter name>`* + "`_discovery.xml`"

      For example, if the metric adapter name is "FooOpenMA", the file name is "FooOpenMA_discovery.xml".

   c. *`<metric adapter name>`* + "`_`" + *`<task name>`* + "`.xml`"

      For example, if the metric adapter name is "FooOpenMA" and the task name is "OpenConnector", the file name is "FooOpenMA_OpenConnector.xml". For details about task names, refer to the *Service Level Management User Manual*.

6 Examples

In this chapter, you will find examples of metric adapters developed using the Open Metric Adapter (Open MA) toolkit.

All of the examples are available in the following directory:

$InstallDir/examples/OpenMA/

In this path, *$InstallDir* indicates the HP OpenView installation directory.

The first example, “EasyOpenMA”, introduces you to the development process, and general issues related to that process. As per the second and third example, we only list some tricks you should pay attention to when you try the examples.

EasyOpenMA

The monitoring application is a fancy monitoring application, called “Easy”. It stores metric data values in an Oracle Database. All information about metric data is stored in an “EASY” table. Table 16 illustrates the structure of the “EASY” table.

Table 16: Structure of the “EASY” Table

Name	Type	Description
METRICTYPE	VARCHAR2(20)	Metric Type
SERVICE	VARCHAR2(20)	Service name
HOSTNAME	VARCHAR2(20)	Hostname
VALUE	FLOAT	Metric value for Availability

To create the “EASY” table, you can use the following SQL statement:

```
CREATE TABLE EASY (
        METRICTYPE VARCHAR2(20)
      , SERVICE VARCHAR2(20)
      , HOSTNAME VARCHAR2(20)
      , VALUE FLOAT
);
```

The available records are shown in Table 17.

Table 17: Records in the “EASY” Table

METRICTYPE	SERVICE	HOSTNAME	VALUE
Availability	SMTP	hpmail1	1.00
Throughput	FTP	hpftp1	1100.00
ResponseTime	HTTP	hpweb1	0.01

Throughput	FTP	hpftp2	2200.00

To populate the “EASY” table, you can use the following SQL statements:

```
INSERT INTO EASY VALUES('Availability', 'SMTP', 'hpmail1', 1.00);
INSERT INTO EASY VALUES('Throughput', 'FTP', 'hpftp1', 1100.00);
INSERT INTO EASY VALUES('ResponseTime', 'HTTP', 'hpweb1', 0.01);
INSERT INTO EASY VALUES('Throughput', 'FTP', 'hpftp2', 2200.00);
```

You may ask, “How can I run the monitoring application? Where is it?” The answer is that you yourself function as the monitoring application. You can update the column “VALUE” with SQL statements manually.

Gathering Requirements for EasyOpenMA

When gathering requirements for EasyOpenMA, keep the following in mind:

- The name of the monitoring application is “Easy”.
- There are three pieces of information specific to metrics monitored by the monitoring application:
  - Metric Type
  - Service
  - Hostname
- Because the data type of column “VALUE” is “FLOAT”, the value type of the metric values is “Double”.
- The data source is the Oracle Database. All required information is stored in the table “EASY”.

Designing EasyOpenMA

When designing EasyOpenMA, keep the following in mind:

- The name of the metric adapter is “EasyOpenMA”.

  Because there is no other possible data source, it is not necessary to include the name of the data source in the name of the metric adapter.
- Organize the monitoring-application-specific attributes as shown in Table 18.

Table 18: Monitoring Application-Specific Attributes

Use	To Represent
MetricTypeName1 MetricTypeValue1	Metric Type
UserDataName1 UserDataValue1	Service
UserDataName2 UserDataValue2	Hostname

- PollingSQLSource is a standard openadaptor source component that can fit into this case.
- The blueprint looks like Figure 15.

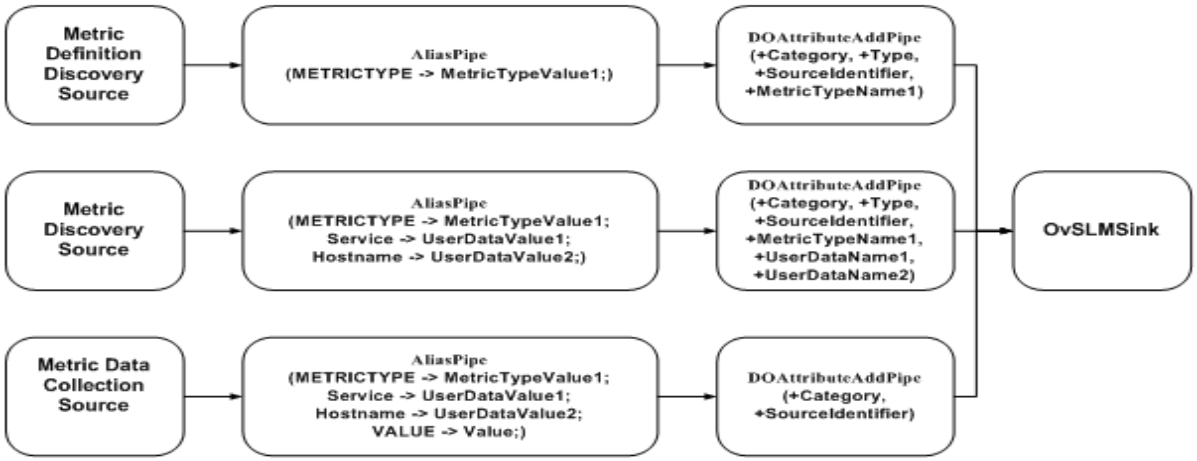

Figure 15: Blueprint of the Adaptor EasyOpenMA"

To keep the example as simple as possible, the location discovery source and the status collection source are not included.

Implementing EasyOpenMA

This section explains how to implement the example adapter EasyOpenMA.

Implementing the Adaptor

The file name of the adaptor configuration file is "EasyOpenMA.props".

You will find the file in the following directory:

$InstallDir/examples/OpenMA/EasyOpenMA

In this path, *$InstallDir* indicates the HP OpenView installation directory.

The name of the adaptor is "EasyOpenMA".

It is easy to use AFEditor to implement links of components within the adaptor according to the blueprint. The most difficult part is to correctly configure each single component.

Each of the three sources has properties related to database connection, including the following:

- Database hostname
- Database port number
- Database instance ID
- Database username
- Database password
- JDBC driver
- JDBC URL

These properties are subject to change in end-user environments. It is a good practice to use variable substitution here.

To represent these database-connection-related properties, you can define the following variables:

```
DB_HOST      =     <DB Hostname>
DB_PORT      =     <DB Port>
DB_SID       =     <DB Instance ID>
DB_USER      =     <DB Username>
DB_PASS      =     <DB Password>
JDBC_DRIVER  =     oracle.jdbc.driver.OracleDriver
JDBC_URL     =     jdbc:oracle:thin:@${DB_HOST}:${DB_PORT}:${DB_SID}
```

Among these properties, the value of `JDBC_DRIVER` is fixed, and the value of `JDBC_URL` is constructed using other properties.

For now, critical information (for example, the database password) is stored in the openadaptor configuration file, in plain text. For this reason, you must be careful when distributing the configuration file. This issue should be resolved in the future. For now, the recommended workaround is to change the permissions of the openadaptor configuration files.

Other properties about the polling periods of the three sources are also subject to change.

You can define the following variables for them:

```
METRICS_PERIOD                  =     300000      # 5 minutes
METRIC_DEF_ DISCOVERY_PERIOD    =     300000      # 5 minutes
METRIC_DISCOVERY_PERIOD         =     300000      # 5 minutes
```

You are free to change the three properties, according to your frequency of updating.

For example, if you change the column “VALUE” in the “EASY” table every 30 minutes, the three properties should be updated as follows:

```
METRICS_PERIOD                  =     1800000     # 30 minutes
METRIC_DEF_ DISCOVERY_PERIOD    =     1800000     # 30 minutes
METRIC_DISCOVERY_PERIOD         =     1800000     # 30 minutes
```

Figure 16 shows the definitions of those properties in AFEditor.

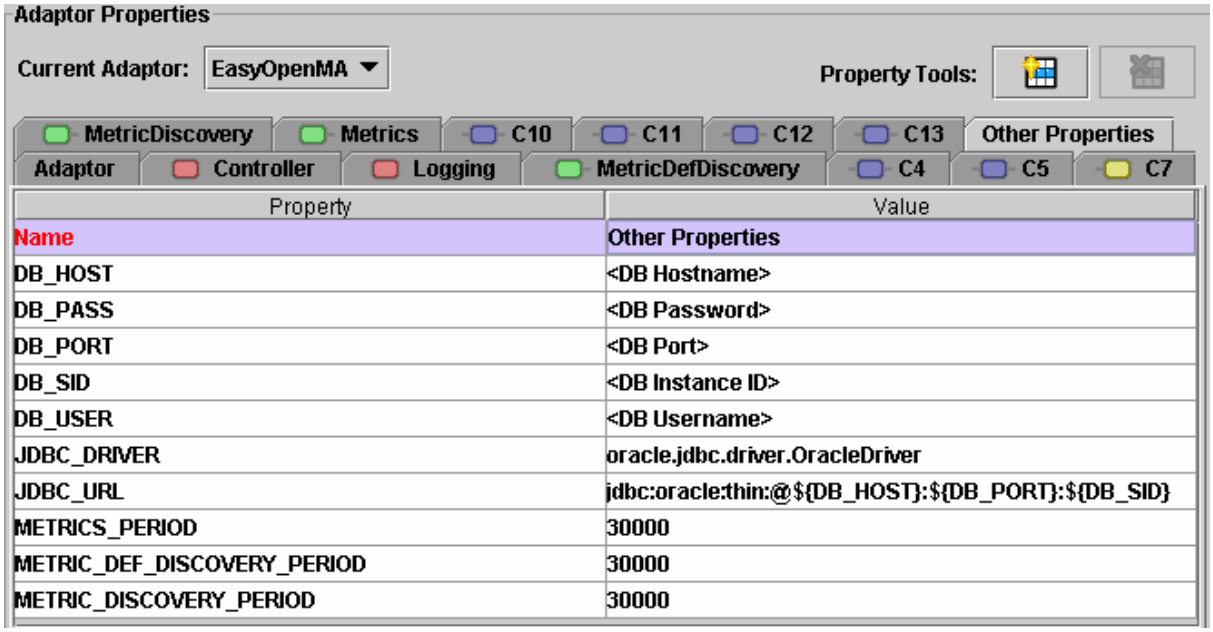

Figure 16: Definition of the Properties

Configuring the Sources

At this point, you configure the three sources, one by one:

- **Metric Definition Discovery Source**

  Figure 17 shows the configuration of the metric definition discovery source named "MetricDefDiscovery" in AFEditor:

  — Variable substitutions are used for defining properties (for example, JdbcDriver, JdbcUrl, PollPeriod, and so on).

  — Although the setting of the property "NextPrimaryKeySQL" is not used, the property is mandatory. It cannot be left empty. So a simple SQL statement is used as a placeholder.

  — In the SelectSQL configuration, "SELECT DISTINCT" must be used. There may be multiple metrics whose metric definitions are identical. "SELECT DISTINCT" helps reduce the number of returned records.

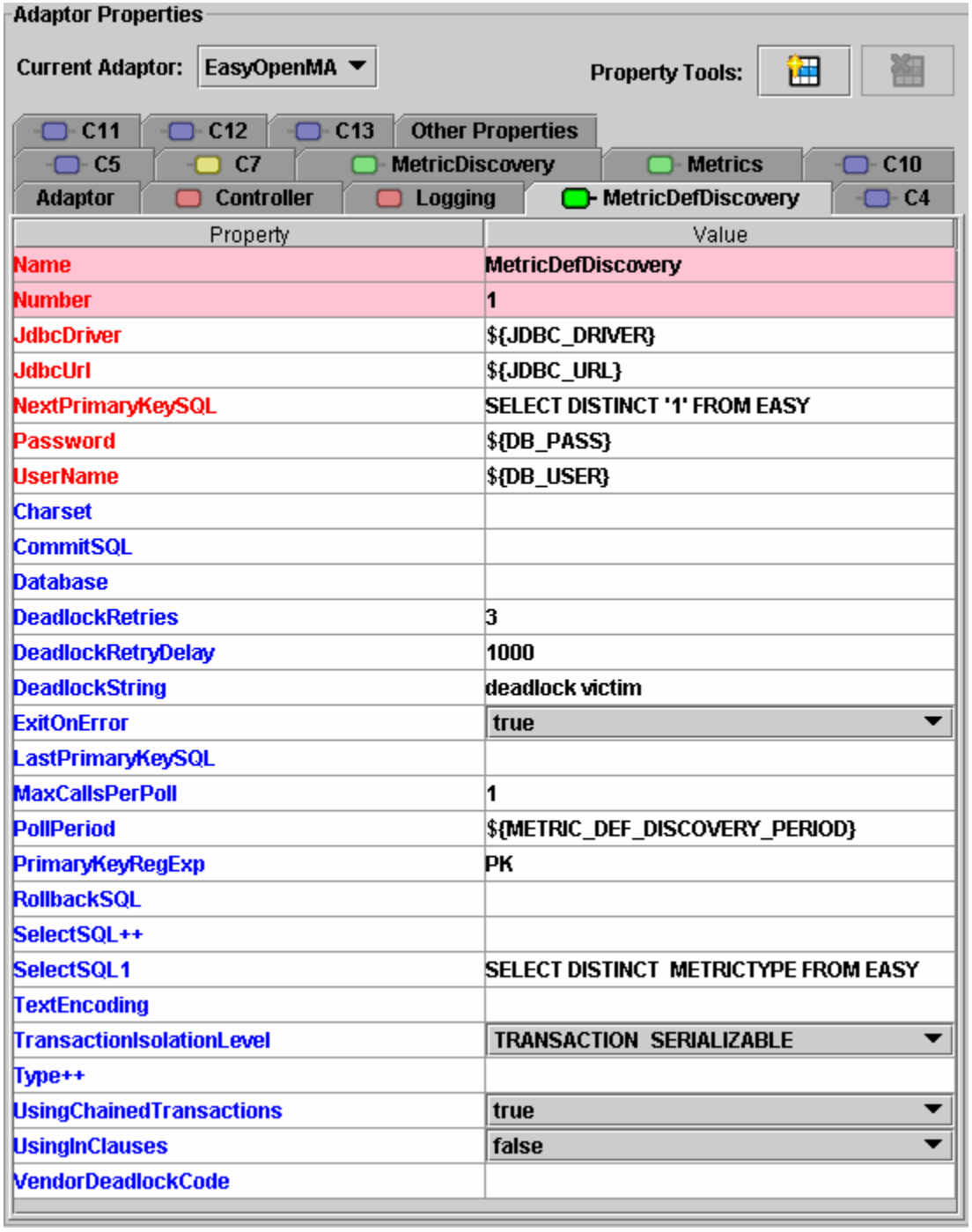

Figure 17: Configuration of the Metric Definition Discovery Source

- **Metric Discovery Source**

  Figure 18 shows the configuration of the metric discovery source named "MetricDiscovery" in AFEditor:

  — Variable substitutions are used to define properties (for example, JdbcDriver, JdbcUrl, PollPeriod, and so on).

  — Although the setting of the property "NextPrimaryKeySQL" is not used, the property is mandatory. It cannot be left empty. So a simple SQL statement is used as a placeholder.

  — In the SelectSQL configuration, "SELECT DISTINCT" must be used. There may be multiple metric data value records for every individual metric. "SELECT DISTINCT" helps reduce the number of returned records.

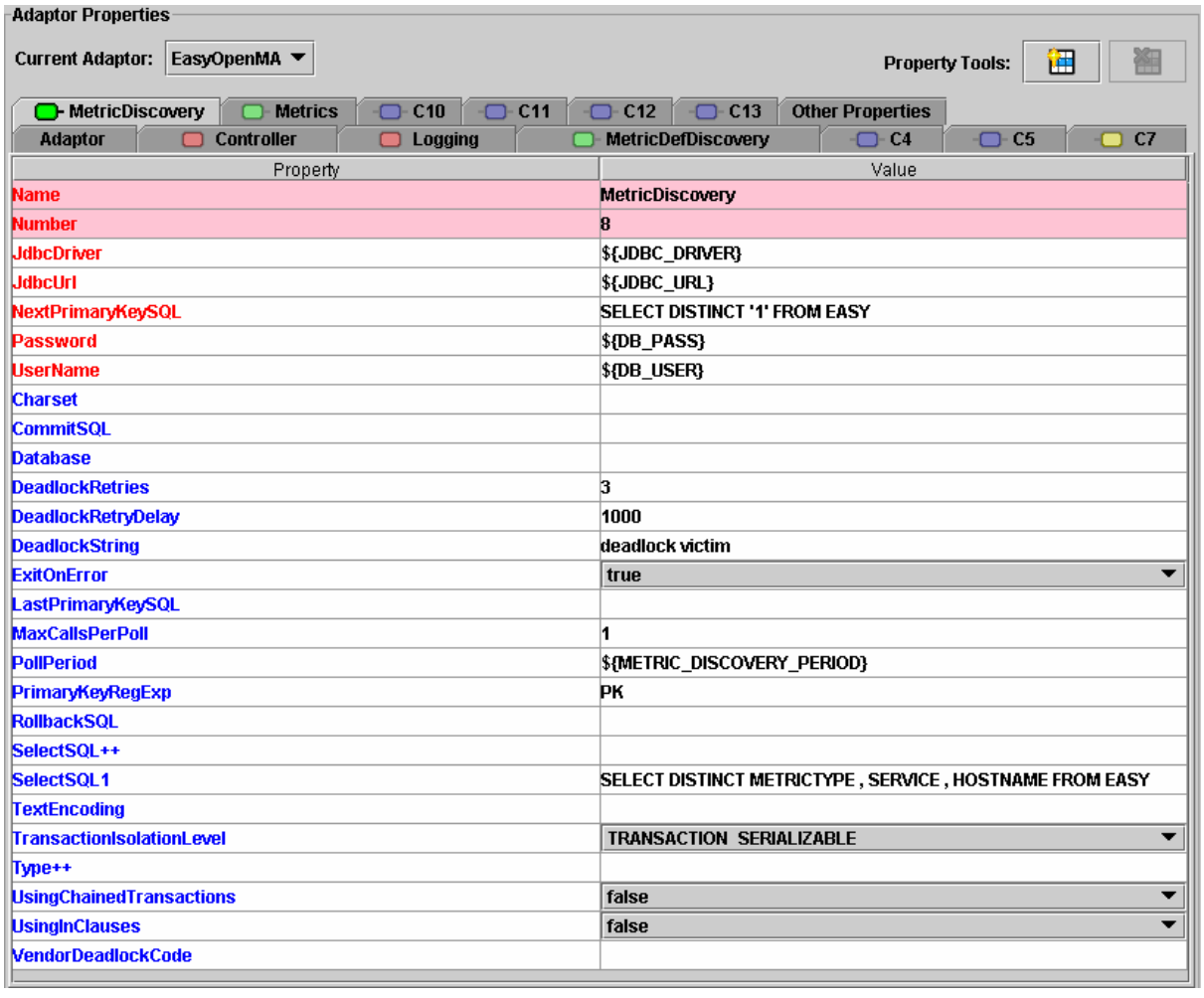

Figure 18: Configuration of the Metric Discovery Source

- **Metric Data Values Collection Source**

  Figure 19 shows the configuration of the metric data values collection source named "Metrics" in AFEditor:

  — Variable substitutions are used for defining properties (for example, JdbcDriver, JdbcUrl, PollPeriod, and so on).

  — Although the setting of the property "NextPrimaryKeySQL" is not used, the property is mandatory. It cannot be left empty. So a simple SQL statement is used as a placeholder.

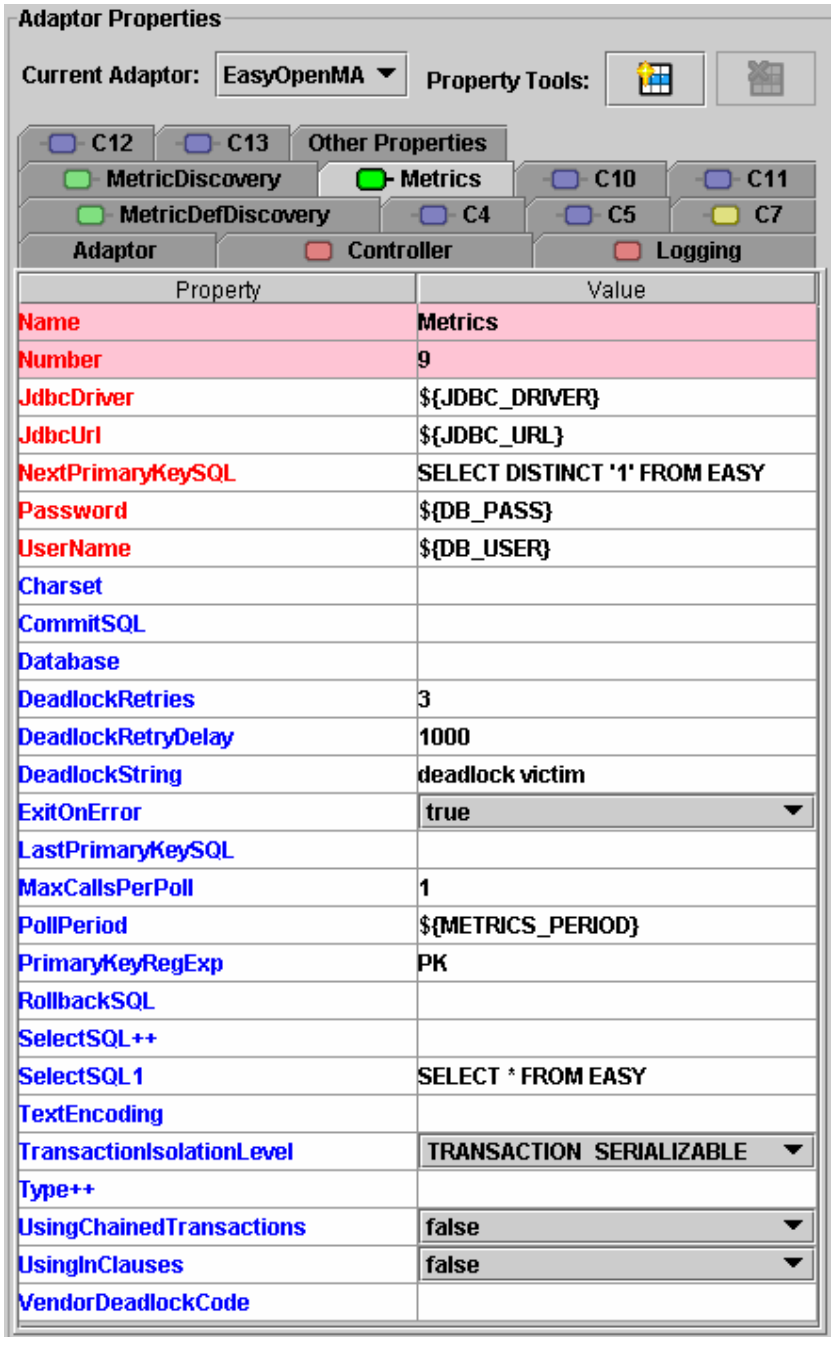

Figure 19: Configuration of the Metric Data Values Collection Source

Configuring Pipe Components

Next, you configure pipe components.

- **AliasPipe**

  There are a total of three AliasPipes. For example, Figure 20 shows the configuration of one AliasPipe in AFEditor.

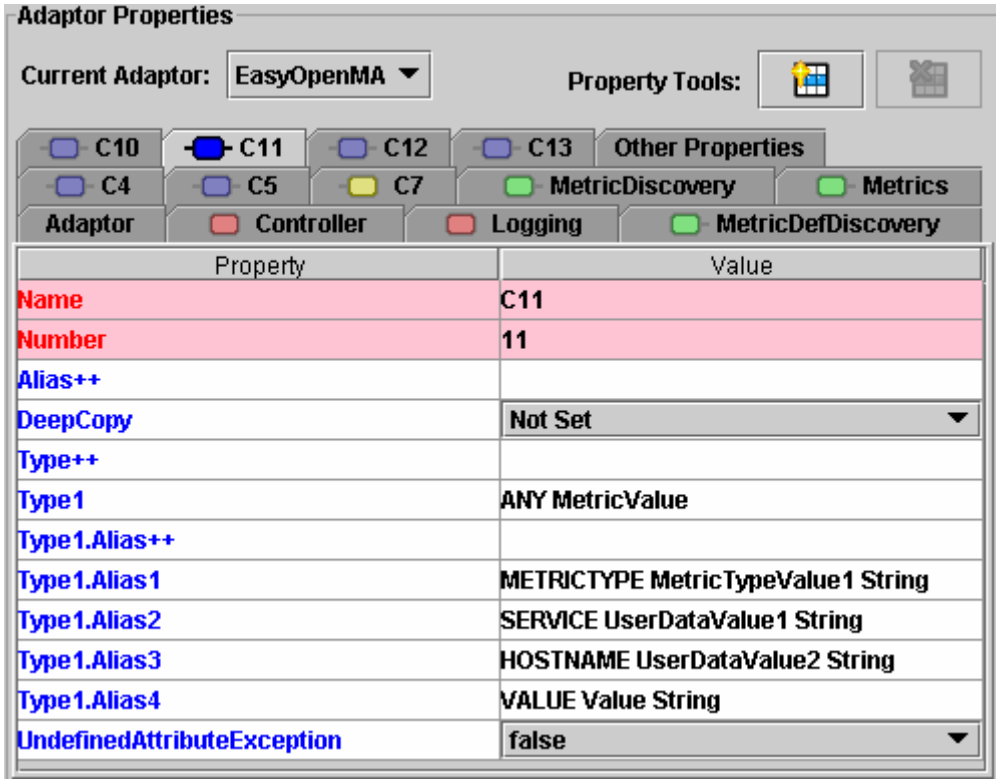

Figure 20: Configuration of One AliasPipe

- **DOAttributeAddPipe**

  There are a total of three DOAttributeAddPipes. For example, Figure 21 shows the configuration of one DOAttributeAddPipe in AFEditor.

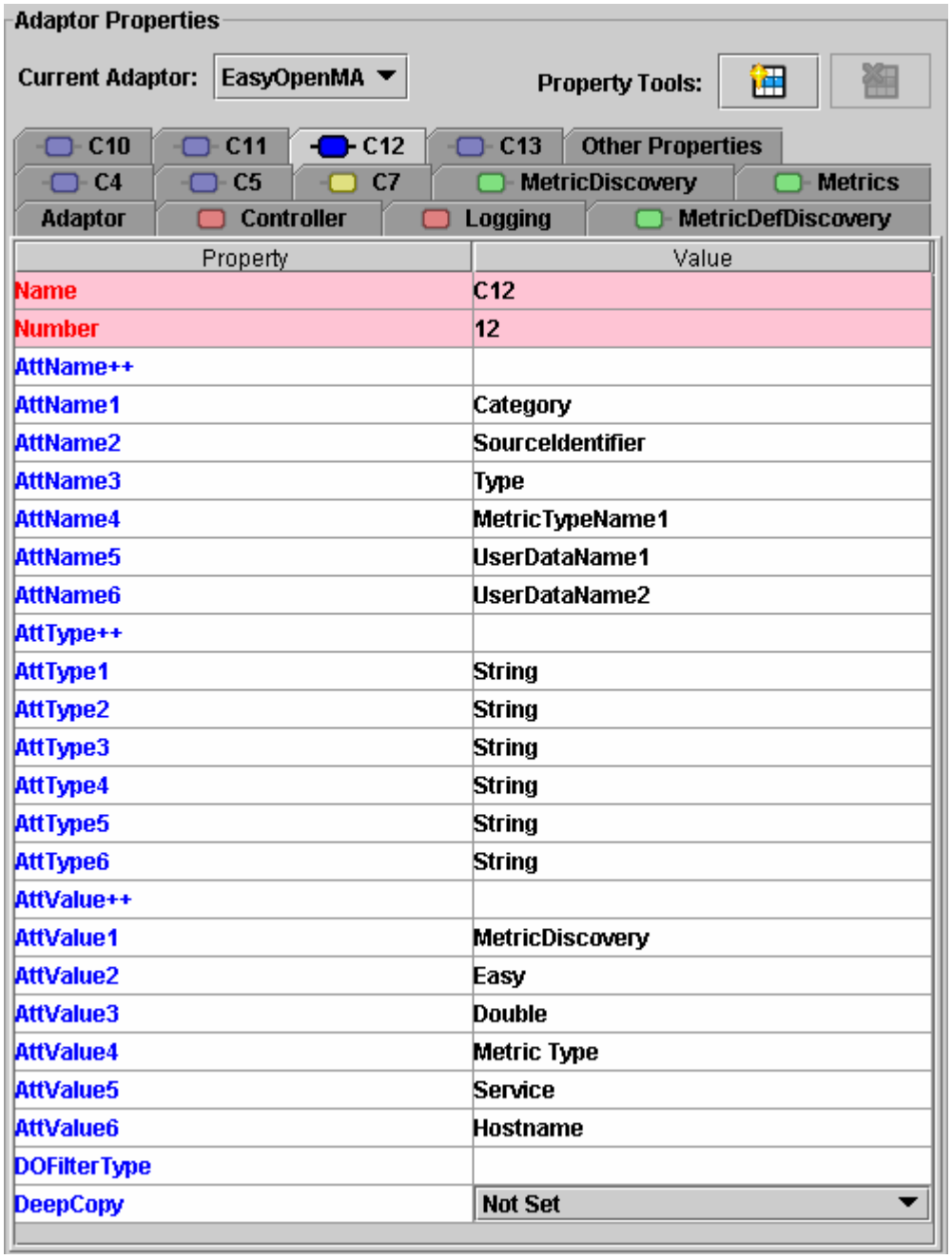

Figure 21: Configuration of One DOAttributeAddPipe

Configuring Sink Components

At this point, you are ready to configure sink components.

- **OvSLMSink**

  Figure 22 shows the configuration of OvSLMSink in AFEditor.

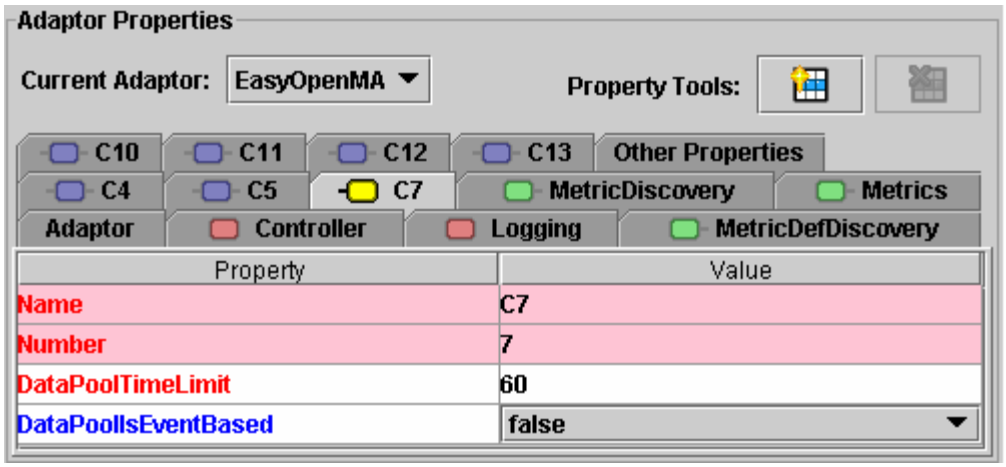

Figure 22: Configuration of OvSLMSink

You have produced an adaptor that has a structure in AFEditor like that shown in Figure 23.

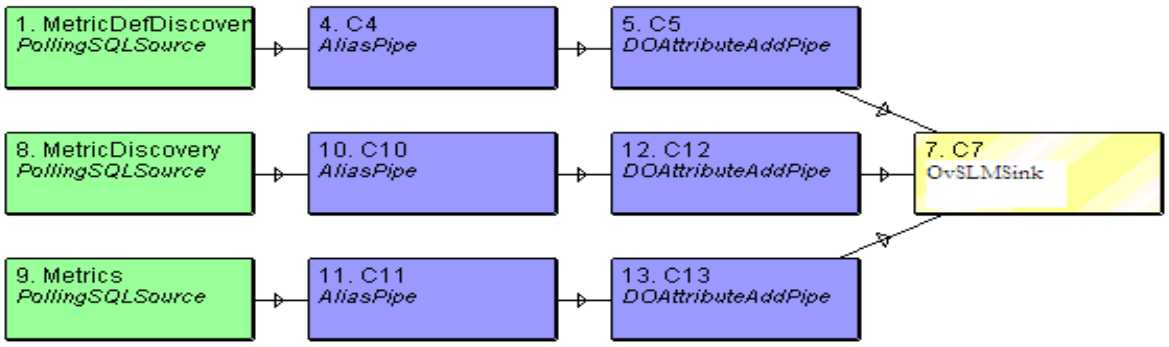

Figure 23: Structure of the EasyOpenMA Adaptor

You can run the EasyOpenMA adaptor in AFEditor. It is a good practice to use InspectorSink to display and verify the execution result.

Figure 24 show the use of InspectorSink in AFEditor.

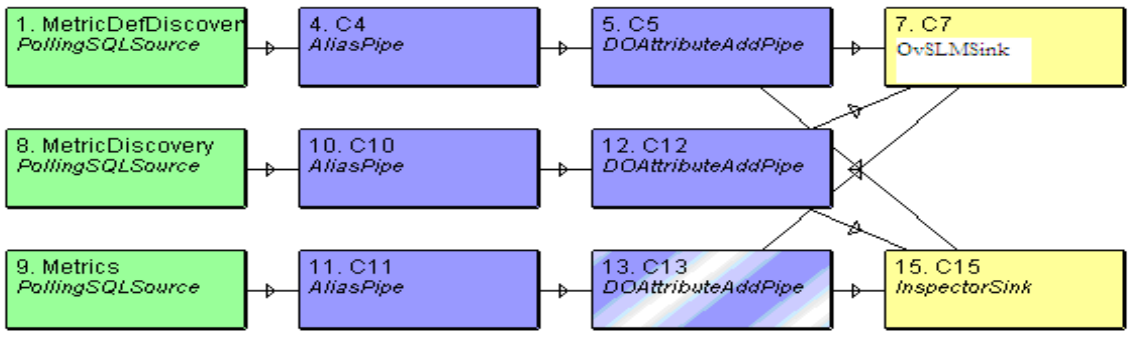

Figure 24: Structure of the EasyOpenMA Adaptor with InspectorSink Configured

After you finish verifying the execution result, you must remember to remove the InspectorSink from the adaptor.

Creating the EasyOpenMA Configuration File

The file name of the EasyOpenMA configuration file is “EasyOpenMA.xml”.

The content of EasyOpenMA configuration file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config>
    <MA name="EasyOpenMA">
        <ServerHost>$SLM_HOSTNAME$</ServerHost>
        <DefaultTaskPollingPeriod>60</DefaultTaskPollingPeriod>
        <DiscoveryInterval>600</DiscoveryInterval>
        <MrpDefinitionDiscoveryInterval>600</MrpDefinitionDiscoveryInterval>
        <LocationDiscoveryInterval>600</LocationDiscoveryInterval>
        <HeartBeatsInterval>300</HeartBeatsInterval>
        <DiscoveryMaxHistory>60</DiscoveryMaxHistory>
        <DataPointVersionByte>1</DataPointVersionByte>
        <TypeByte>1</TypeByte>
        <isEventBased>0</isEventBased>
        <SequenceNumber>0</SequenceNumber>
        <DefaultTaskExpirePeriod>600</DefaultTaskExpirePeriod>
        <DataPointSynchronizationDelay>10</DataPointSynchronizationDelay>
        <Publisher.RESPONSE_TIMEOUT>60</Publisher.RESPONSE_TIMEOUT>
    </MA>
    <Connector name="OpenConnector">
        <Class>com.hp.ov.sd.slm.sa.openma.OpenConnector</Class>
        <MaxLatency>3</MaxLatency>
    </Connector>
    <DiscoveryLocationFilter>
        <All/>
    </DiscoveryLocationFilter>
</Config>
```

$SLM_HOSTNAME$ indicates the host where the corresponding SLM server is running. End users are required to replace it with the name of the host where the SLM server is running. It is recommended that you use the “Metric Adapter Configuration GUI” to set the parameter.

Creating the EasyOpenMA Control File

The file name of the EasyOpenMA control file is “EasyOpenMACtrl.xml”.

The content of the EasyOpenMA control file is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ovc:OvCtrl xmlns:ovc="http://openview.hp.com/xmlns/ctrl/registration/1.5">
  <ovc:Component>
    <ovc:Name>easyopenma</ovc:Name>
    <ovc:Label>
      <ovc:String>Easy Open Metric Adapter</ovc:String>
    </ovc:Label>
    <ovc:Category>SLM</ovc:Category>
    <ovc:Options>
      <ovc:AllowAttach>true</ovc:AllowAttach>
      <ovc:AutoRestart>true</ovc:AutoRestart>
      <ovc:AutoRestartLimit>5</ovc:AutoRestartLimit>
      <ovc:AutoRestartMinRuntime>60</ovc:AutoRestartMinRuntime>
      <ovc:AutoRestartDelay>5</ovc:AutoRestartDelay>
      <ovc:AutoShutdown>false</ovc:AutoShutdown>
      <ovc:AutoShutdownTimer>5</ovc:AutoShutdownTimer>
      <ovc:StartAtBootTime>false</ovc:StartAtBootTime>
      <ovc:MentionInStatus>true</ovc:MentionInStatus>
      <ovc:Monitored>true</ovc:Monitored>
      <ovc:IsContainer>false</ovc:IsContainer>
      <ovc:PollingInterval>30</ovc:PollingInterval>
      <ovc:WorkingDirectory>$OvDataDir</ovc:WorkingDirectory>
    </ovc:Options>
    <ovc:ProcessDescription>java</ovc:ProcessDescription>
    <ovc:CommandLine>-Deasyopenma</ovc:CommandLine>
    <ovc:OnHook>
      <ovc:Name>START</ovc:Name>
      <ovc:Actions>
        <ovc:Start>
          <ovc:CommandLine>“$OvJreDir\java” -D$component -Xrs -cp “$InstallDir\java\OvSlmMaOpen.jar”
com.hp.ov.sd.slm.sa.openma.OpenMain EasyOpenMA </ovc:CommandLine>
        </ovc:Start>
      </ovc:Actions>
    </ovc:OnHook>
    <ovc:OnHook>
      <ovc:Name>STOP</ovc:Name>
      <ovc:Actions>
        <ovc:Execute>
          <ovc:CommandLine>”$OvJreDir\java” -D$component -Xrs -cp “$InstallDir\java\OvSlmMaOpen.jar”
com.hp.ov.sd.slm.sa.openma.OpenMain EasyOpenMA -stop</ovc:CommandLine>
        </ovc:Execute>
      </ovc:Actions>
    </ovc:OnHook>
  </ovc:Component>
</ovc:OvCtrl>
```

Testing EasyOpenMA

Copy the files “EasyOpenMA.props”, “EasyOpenMA.xml”, and “EasyOpenMACtrl.xml” into the following directory:

$DataDir/conf/slm/

In this path, *$DataDir* indicates the HP OpenView data directory.

Then configure EasyOpenMA by updating the file “EasyOpenMA.xml”.

You must provide the hostname of the corresponding SLM server by updating the value of the parameter $SLM_HOSTNAME$.

$

Start EasyOpenMA

It is convenient to start EasyOpenMA using the following commands:

- **Windows**

  startOpenMA.bat EasyOpenMA

- **UNIX**

  startOpenMA.sh EasyOpenMA

The other method is to configure EasyOpenMA in HP OpenView Control (OVC) by executing the following command:

ovcreg -add ”$DataDir/conf/slm/EasyOpenMACtrl.xml”

After that, you can start EasyOpenMA using the following command:

ovc -start easyopenma

Stop EasyOpenMA

To stop EasyOpenMA, use the following command:

ovc -stop easyopenma

Restart EasyOpenMA

To restart EasyOpenMA, use the following command:

ovc -restart easyopenma

View the EasyOpenMA Status

To view the status of EasyOpenMA, use the following command:

ovc -status easyopenma

For information about the configuration of the Open Metric Adapter, refer to the *Service Level Management User Manual.*

Releasing EasyOpenMA

To release EasyOpenMA, you use Windows batch scripts.

Install EasyOpenMA

To install EasyOpenMA, use the following Windows batch script:

```
@echo off

REM this file will install and configure EasyOpenMA

cd %~dp0

echo get OpenView directories...
for /f "tokens=*" %%a in ( '"ovpath.exe" -instdir' ) do set OV_HOME=%%a
for /f "tokens=*" %%a in ( '"ovpath.exe" -datadir' ) do set OV_DATA=%%a
if not exist "%OV_HOME%" (
    echo HP OpenView MUST be installed on this machine first
    exit -1
)

echo copy configuration files for EasyOpenMA...
copy /y EasyOpenMA.props "%OV_DATA%\conf\slm"
copy /y EasyOpenMA.xml "%OV_DATA%\conf\slm"
copy /y EasyOpenMACtrl.xml "%OV_DATA%\conf\slm"

echo configure easyopenma into HP OpenView Control...
"%OV_HOME%\bin\ovcreg" -add "%OV_DATA%\conf\slm\EasyOpenMACtrl.xml"

pause
```

Uninstall EasyOpenMA

To uninstall EasyOpenMA, use the following Windows batch script:

```
@echo off

REM this file will install and configure EasyOpenMA

cd %~dp0

echo get OpenView directories...
for /f "tokens=*" %%a in ( '"ovpath.exe" -instdir' ) do set OV_HOME=%%a
for /f "tokens=*" %%a in ( '"ovpath.exe" -datadir' ) do set OV_DATA=%%a

if not exist "%OV_HOME%" (
    echo HP OpenView MUST be installed on this machine first
    exit -1
)

echo unconfigure easyopenma from HP OpenView Control...
"%OV_HOME%\bin\ovcreg" -del easyopenma

echo delete configuration files for EasyOpenMA...
del /f "%OV_DATA%\conf\slm\EasyOpenMA.props"
del /f "%OV_DATA%\conf\slm\EasyOpenMA.xml"
del /f "%OV_DATA%\conf\slm\EasyOpenMACtrl.xml"

echo delete files generated by EasyOpenMA...
del /f "%OV_DATA%\conf\slm\EasyOpenMA_discovered.props"
del /f "%OV_DATA%\conf\slm\EasyOpenMA_*.xml"

pause
```

OvisMsSqlOpenMA

Please read readme.txt in this example for more details about the issue caused by org.openadaptor.adaptor.jdbc.PollingSQLSource in openadaptor(1.7.0)

WebService

The web service we use in this example, http://live.capescience.com/ccx/AirportWeather, may be unavailable sometimes. So please check if the service is available before you try to run this example. In additional, polling interval setting (see parameter 'PollPeriod' in props file) only works when parameter 'MaxCallsPerPoll' is used and is explicitly set to 1 (MaxCallsPerPoll = 1). This defect happens in openadaptor 1.7.0, which we include in HP OpenView SD/SLM (version 5.0). The issue will be resolved in the future release of openadaptor.

7 Best Practices

Before you decide to write your own openadaptor components, make sure that it is really necessary to do so. There may be already some openadaptor components that meet your requirements. If you are not sure which components meet your requirements, consult with openadaptor experts (for example, submit your questions to the openadaptor forum).

Tips for Sources

Generally, it is wise to define five sources in the adaptor. Each source collects one of the five categories of data. For details, see “Benefit of Using Five openadaptor Sources.”

Tips for AliasPipe

This characteristic is best explained by an example.

A DataObject has three attributes:

- Category
- MetricType
- Availability

To rename the attribute “Availability” to “Value”, the configuration of the AliasPipe is something like this:

```
Adaptor.Component.Alias1 = “Availability” “Value”
Adaptor.Component.Alias2 = “Category” “Category”
Adaptor.Component.Alias3 = “MetricType” “MetricType”
```

The last two lines rename the attribute “Category” to “Category”, and rename the attribute “MetricType” to “MetricType”. There is no difference between the input attribute name and the output name. Nevertheless, the two lines are important because they tell the AliasPipe to put the two attributes, “Category” and “MetricType”, into the outgoing DataObject. If the two lines are not included, the two attributes “Category” and “MetricType” are not available in the outgoing DataObject.

Do *not* forget to include all attributes that must appear in the outgoing DataObject in the configuration of the AliasPipe, including those attributes which are not to be renamed (by configuring them with the identical input name and output name). It is generally a good practice to put AliasPipe before pipe components that add new attributes. In this way, the configuration of the AliasPipe can be simplified by reducing the number of attributes that are not really renamed but that have to be involved in the configuration of the AliasPipe.

Tips for AFEditor

AFEditor does not support undo or redo. You must be careful when you perform delete operations.

In this section, you will learn about typical operations in which it is easy to make mistakes.

Deleting Text

When deleting text in AFEditor, you could make a mistake in the following process:

1 Select a component in the panel named “Adaptor Framework”, as shown in Figure 25.

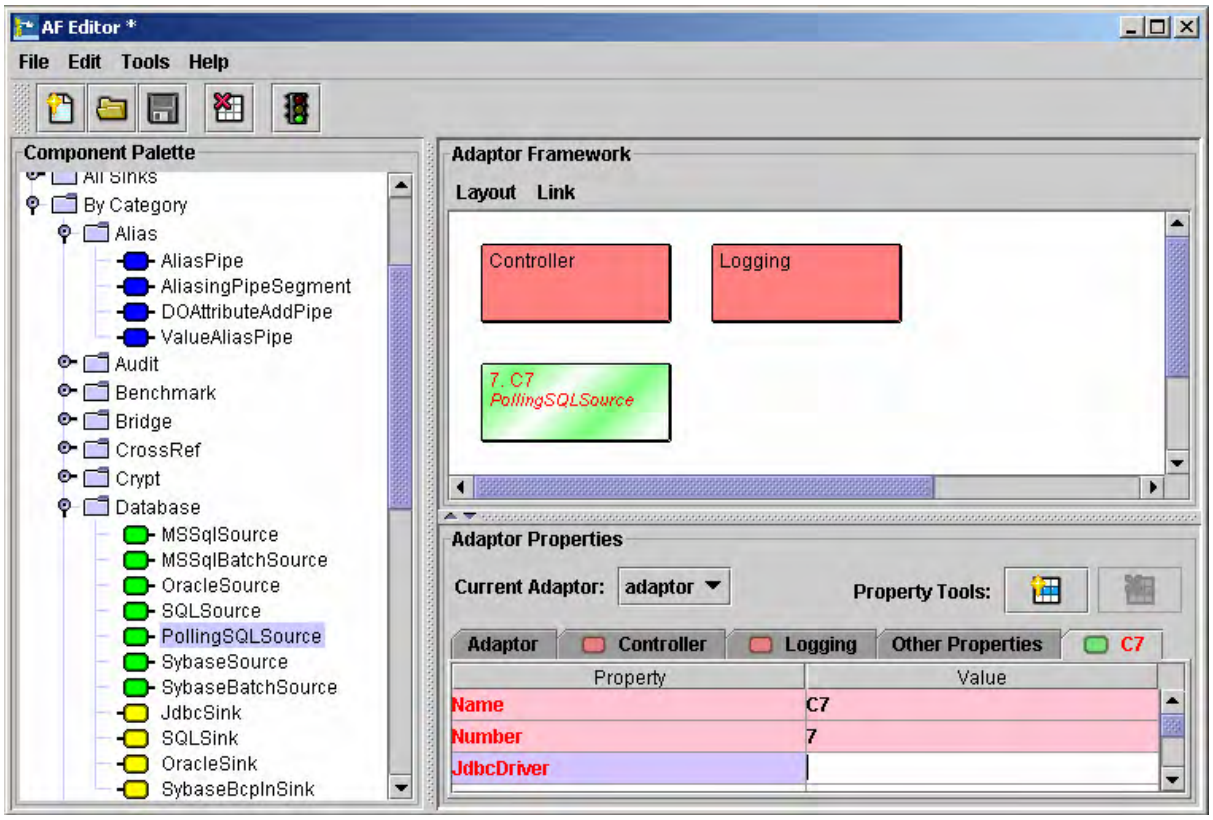

Figure 25: Adding a Component

2 Edit the properties of the selected component in the panel named “Adaptor Properties”, (for example, the property “JdbcDriver”).

When you edit the value of the property “JdbcDriver”, you find that previous input is wrong. So you try to delete the old value of the property by selecting the text in the “Value” field of the property “JdbcDriver” (as shown in Figure 26), and then pressing the **DEL** key.

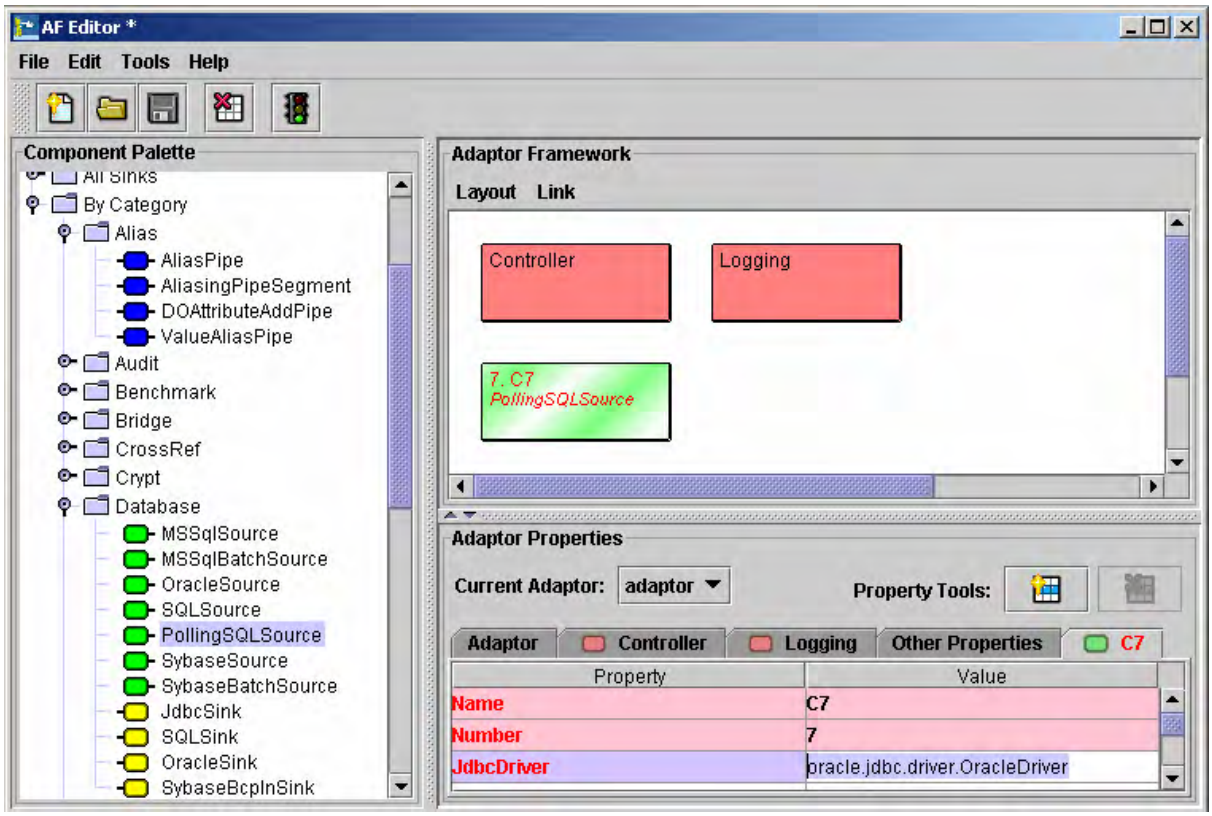

Figure 26: Selecting the Text and Trying to Delete It

3 After you press the **DEL** key, the component is deleted, as shown in Figure 27.

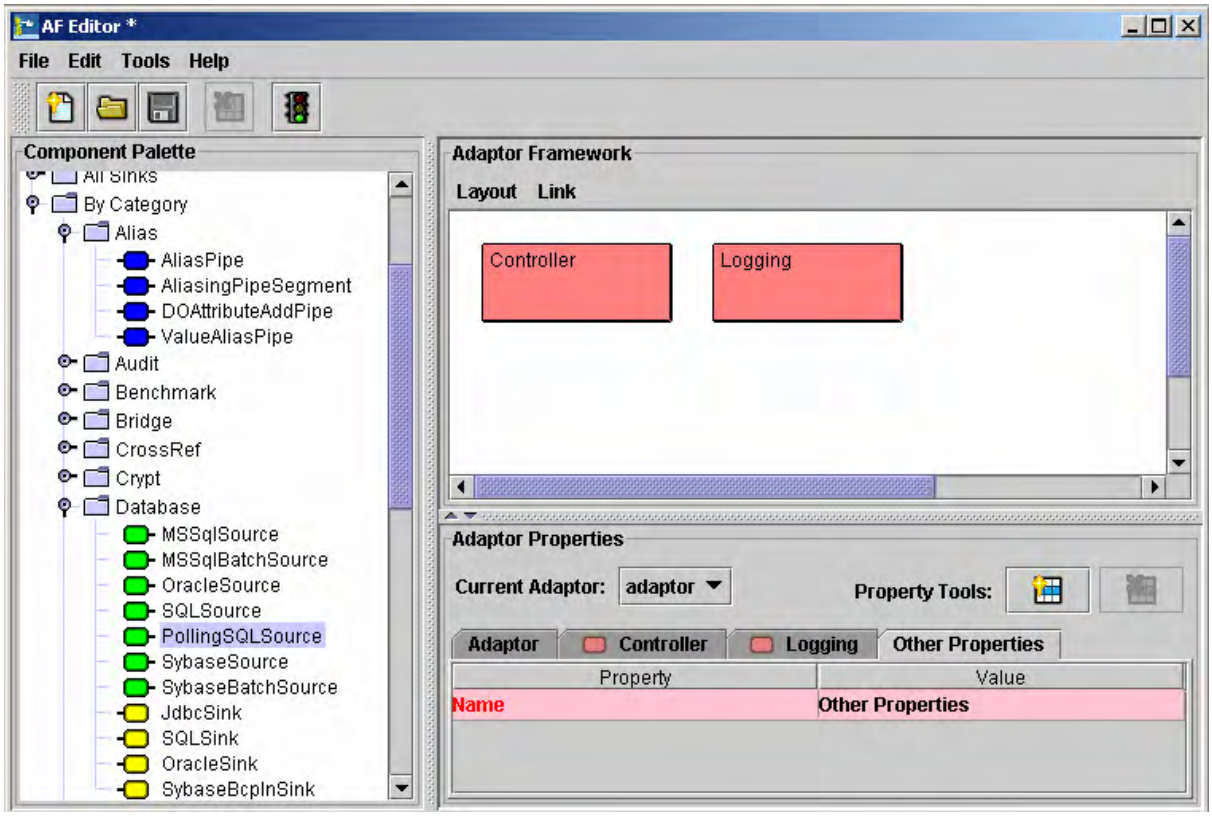

Figure 27: Component is Deleted

This is not the operation you intended to perform. You wanted to delete the text, but the component is deleted instead. This is a defect in AFEditor. As a workaround, deselect the component before you try to edit the property of the component.

8 Feedback and Troubleshooting

Troubleshooting

Timestamp for Data Collected by Open Metric Adapter

There is no issue when you do not set any timestamp for the data collected by Open Metric Adapter. Open Metric Adapter will automatically add timestamp to the collected data using system time.

There maybe exist problems when the user specifies certain timestamp for the data which will be collected by Open Metric Adapter. For example, the time from database is used as the timestamp for the data collected. In such circumstance, the user should pay attention to 2 parameters:

'DataPoolTimeLimit' in props file

'DiscoveryMaxHistory' in xml file

The user should make sure that in certain time, the difference of the timestamp between the oldest data and any other data, which are being collected from 5 sources(Metric Definition Discovery, Metric Discovery, Metric Data Collection, Location Discovery and Status), is always within 'DataPoolTimeLimit'. For example, 3 sources are used and in certain time, Open Metric Adapter collects one Metric Definition Discovery with timestamp 09:00:00 01Jun2000, one Metric Discovery with timestamp 09:00:55 01Jun2000 and one Metric Data Collection with timestamp 09:00:15 01Jun2000. 'DataPoolTimeLimit' is set to 60 seconds. Metric Discovery is 55 seconds later than the oldest data, Metric Definition Discovery. Metric Data Collection is 15 seconds later than Metric Definition Discovery. The difference is both within 'DataPoolTimeLimit'(60 seconds). All data are valid in this case. Then if in any time the timestamp the user sets for Metric Discovery is always 1 hour later than Metric Definition Discovery or Metric Data Collection, the Metric Discovery will be discarded by Open Metric Adapter always.

The user should also make sure the difference between the timestamp of the data collected by Open Metric Adapter and the system time of the machine where Open Metric Adapter is located is within 'DiscoveryMaxHistory'. For example, 'DiscoveryMaxHistory' is set to 60 seconds. Then if the timestamp of the data collected by Open Metric Adapter is 1 hour later than the system time of the machine where Open Metric Adapter is located, the data will be discarded.

JVM Heap Size

In some circumstance, increasing JVM heap size can help to resolve the error, such as 'out of memory', thrown by Open Metric Adapter when it deals with huge data.