

Extending Oracle Enterprise Manager 10g

Alexander Gorbachev

*This article was first published
in the "SELECT Journal"
Quarter 2 2006, Volume 13, No. 2*

Table of Content

INTRODUCTION	3
EXTENDING EXISTING ORACLE TARGETS	3
SQL-based User Defined Metrics	4
OS command-based Metrics	6
UDMs Pros and Cons	9
CREATING NEW TARGET TYPES	9
Target type metadata file	10
Display element	13
InstanceProperties element	13
Metric element	14
QueryDescriptor element	16
Fetchlets	17
Validating Target Type Metadata	20
Validation with ILint	20
Validation with Metric Browser	22
Default Collections	22
Putting It All Together	24
WHERE TO GO NEXT?	26
CONCLUSION	26

Extending Oracle Enterprise Manager 10g

Are you missing your favorite scripts and indicators in Oracle Enterprise Manager Grid Control? Find out how easy it is to incorporate your own metrics into the Grid Control framework. Do you manage a multiple vendors' environment and need to monitor many different components? Learn how to create your own completely new target types in Grid Control with rich set of availability and performance metrics.

Introduction

Many of you have probably used Oracle Enterprise Manager (OEM) to monitor and control your Oracle environment. Now with Oracle Grid Control 10g you may even consider it a tool of choice for enterprise monitoring and management. Thanks to the exciting new feature Extensibility framework in OEM 10g you can now use it to monitor virtually any component of an enterprise including systems from vendors other than Oracle.

There are two ways to add new functionality to OEM – adding user-defined metrics (UDMs) to standard targets provided by Oracle and adding new targets such as network components, application servers, third-party and home grown applications, non-Oracle databases, etc. UDMs have limited functionality but they are much easier to set up. We will cover both ways, starting with simpler UDMs.

This article is organized as a series of examples and provides a good deal of information to get you started with extending OEM. It is assumed that the reader has some knowledge of OEM and its principles in order to follow the practical examples. The material is based on 10g Grid Control Release 1 and Release 2. There are few handy enhancements in the second release in the area of extensibility. Examples are valid for any platform excluding shell scripts that are written for a UNIX environment. Readers can easily adapt them to be used in Windows using Perl, for example. Note that all samples of SQL and scripts are just for demonstration purposes.

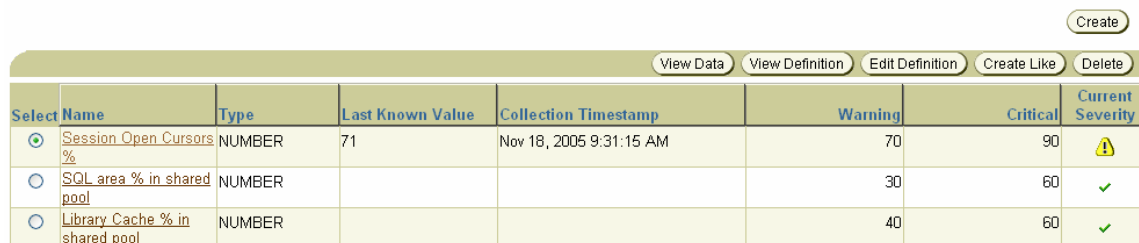
Extending Existing Oracle Targets

Only Oracle Database and Host targets can be extended through the UDM mechanism. Metrics based on SQL queries are used for database targets and those based on OS scripts are for host targets.

UDMs are accessible from the bottom of the target's home page in the Related Links section. From this list you can manage existing and create new metrics. Figure 1 shows a sample UDM screen.

User-Defined Metrics

User-Defined Metrics allow you to extend the monitoring of your environment by defining new metrics to be monitored. New metrics are defined by specifying your own custom scripts.






Select	Name	Type	Last Known Value	Collection Timestamp	Warning	Critical	Current Severity
<input checked="" type="radio"/>	Session Open Cursors %	NUMBER	71	Nov 18, 2005 9:31:15 AM	70	90	
<input type="radio"/>	SQL area % in shared pool	NUMBER			30	60	
<input type="radio"/>	Library Cache % in shared pool	NUMBER			40	60	

Figure 1 User-Defined Metrics screen

OEM provides basically the same functionality for UDM as for standard metrics. You can set warning and critical thresholds, receive notifications, and view metric history. UDMs are fully integrated into 10g Grid Control framework.

SQL-based User Defined Metrics

OEM 10g Release 1 supports only metrics returning a single value so SQL queries should return a single row with one column.

Let's create a sample metric that will show actual number of open cursors per session as a percent of the max open cursors (defined by `open_cursors` `init.ora` parameter). We only need to consider the session with top number of opened cursors. This metric will show how close we are to the `open_cursors` limit. Note that this is sampling data and reflects the state at the moment of metric collection so it might be possible that in between collections the value is higher or lower.

```
SELECT ROUND (c.open_cursors / p.value * 100) open_cursors
FROM (SELECT sid, COUNT (*) open_cursors
      FROM v$open_cursor
      GROUP BY sid
      ORDER BY 2 DESC) c,
     v$parameter p
WHERE p.name = 'open_cursors' AND ROWNUM = 1
```

Listing 1 SQL query to retrieve maximum percent of open cursors

Enter "Max Open Cursors %" as metric name. In the SQL Query field enter the select statement from Listing 1. Select metric type NUMBER and, if you use OEM 10gR2, chose a Single Value option for SQL Query Output (OEM 10gR1 has no such option). In the credentials section, enter the Oracle username and password – these can be the same as your target configuration, for example, DBSNMP user. Now chose the greater than sign (>) as a comparison operator and define warning and critical thresholds. I took values 70 and 90 for warning and critical thresholds respectively, with the number of consecutive occurrences preceding notification set to 1. We define that collection should start immediately and repeat every five minutes. That's it – our first user-defined metric is ready. In 10gR2 we can test metric collection using the "Test" button before actually creating it. In the second release it's also possible to define a custom alert message but we will cover it in the next example.

OEM will collect this new UDM as if it's a normal metric and store the results in the repository. You can view metrics history by clicking on its name in the User-Defined Metrics screen. See the output produced by this metric in my sample environment on Figure 2.

User-Defined Numeric Metric: Script Max Open Cursors %: Last 24 hours

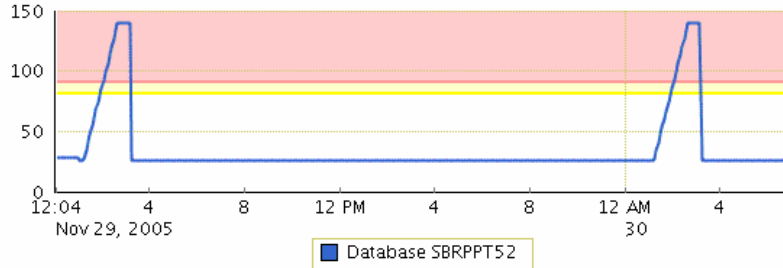
Last Updated Nov 30, 2005 6:54:34 AM GMT
View Data Last 24 hours

Script **Max Open Cursors %**

Statistics

Last Known Value 26
Average Value 32.02
High Value 141
Low Value 26
Warning Threshold 80
Critical Threshold 90
Occurrences Before Alert 1
Corrective Action None

Metric Value



Alert History

Comment for Most Recent Alert

Severity	Timestamp	Message	Last Comment	Details
✓	Nov 30, 2005 3:14:34 AM	CLEARED - One of sessions opened 26% of opened cursors		-
✗	Nov 30, 2005 2:04:34 AM	One of sessions opened 94% of opened cursors		-
⚠	Nov 30, 2005 1:59:34 AM	One of sessions opened 87% of opened cursors		-
✓	Nov 29, 2005 3:14:34 AM	CLEARED - One of sessions opened 26% of opened cursors		-

Figure 2 UDM History screen

Note that you can have metrics that return string values and notification conditions can be set using either CONTAINS or MATCH operators. One example of such metric is the standard metric database state (Mounted, Open, etc). We will use this type later when we create an OS-based metric.

In OEM 10g Release 2 you can define UDMs with two columns – key and value. In this case OEM will consider the key-value pairs separately. A good example of a standard metric is “Tablespace Usage” which returns the percent used for each tablespace. An excellent choice for a two-column SQL-based UDM is queue monitoring. Oracle Queues are part of Oracle’s Advanced Queuing feature – see [4] for more information on AQ.

```
SELECT q.owner || '.' || q.NAME queue_name, s.waiting
FROM v$aq s, dba_queues q
WHERE s.qid = q.qid
ORDER BY 2 DESC
```

Listing 2 SELECT statement for AQ monitoring

This time we choose the “Two columns” option in the “SQL Query Output” field with “Metric Type” once again “NUMBER”. We will fill the SQL Query field with a statement from Listing 2. You may choose to filter queue owners and names so that it takes into account only the queues that you are interested in so modify the WHERE clause to suite your requirements. Use the same credentials as in the previous example. Alert thresholds can be defined as generic and valid for all keys and/or as key specific in format “<key1>:<threshold1>;<key2>:<threshold2>”. I will enter just generic thresholds: 10 and 100 for warning and critical alerts, respectively.

Another nice feature of the second release is that a custom message is displayed whenever threshold conditions are met. A custom notification message should include the placeholders %Value% and %Key%. We define it as “Queue %Key% contains %Value% messages”. Use the “Test” button to verify your statement.

When the first metric values are collected they can be seen in the UDM screen (Figure 3) and historical values for each key are available as a graph similar to the one on Figure 2.

Select	Details	Name ▲	Type	Last Known Value	Collection Timestamp	Comparison Operator	Warning	Critical	Current Severity
		AQ size	NUMBER		Nov 30, 2005 8:44:09 AM GMT	>	10	100	
		Key		Value					Severity
		OAQPNLPRD_P.ADL		0					
		OAQPNLPRD_P.ADL_ERROR		0					
		OAQPNLPRD_P.MSG		0					
		OAQPNLPRD_P.MSG_ERROR		0					
		OAQPNLPRD_P.PFS		0					
		OAQPNLPRD_P.PFS_ERROR		0					
		OAQPNLPRD_P.PNL		0					
		OAQPNLPRD_P.PNL_ERROR		0					
		OAQPNLPRD_P.RQL		0					
		OAQPNLPRD_P.RQL_ERROR		0					
						Total Keys for Metric	32		

Figure 3 UDM screen for two columns metric

Alerts for UDM will be generated and shown on the home page along with others using the message template provided in the metric definition. In our example, a critical alert can be “Queue SCOTT.MY_QUEUE contains 110 messages”.

One small piece of advice about SQL based metrics – avoid returning a NULL from the query and use the NVL function to substitute a value for NULL. I saw OEM behaving strangely when my UDM returned a NULL value in the key column.

OS command-based Metrics

UDM based on an OS script/command can be used for a target of type Host. This can be either a script or a binary program returning collected data to its standard output. Oracle standard targets use Perl because it exists on all platforms and there are no big issues with the portability between Windows and UNIX platforms unless the principles of metric collection vary significantly.

We will create an UDM to monitor a MySQL engine running on the machine. Let’s call our metric MySQL Engine Status and it will return a string either “Up” or “Down”. Defining a numeric metric is practically the same and you can follow the same path. OS command based UDMs with two columns are not supported even in OEM Release 2 so we are limited to one value only.

The screen to create a new OS-based UDM is similar to the one for a SQL-based metric. Instead of a SQL query we should enter an OS command to run the required program. There is an optional field for a space-separated list of environment variables that should be set before calling the specified script. Note on the right side, the area with the list of predefined OEM variables which can be used by the command line in the format of %variable%. These will be substituted by OEM with corresponding values. For example, %perlbin% is the path to the Perl binary directory. Perl is the Oracle standard scripting language, and one of its definite advantages is portability between different platforms. For our example we will use just a simple

UNIX shell script. If you need to use it for Windows, you should use either Perl or DOS-style scripting.

```
#!/usr/bin/sh
# Called from Oracle Grid Control to check MySQL engine status. Returns "Up"
or "Down"
# Author: Alexander Gorbachev, 2005

status=`/usr/bin/mysqladmin1 ping 2>&1`

if [ "$status" = "mysqld is alive" ] ; then
    echo "em_result=Up"
else
    # Count mysqld processes
    mysqld=`ps -e | grep mysqld | grep -v grep | wc -l`
    if [ "$mysqld" = "0" ] ; then
        # No processes - MySQL is down
        echo "em_result=Down"
    else
        # Process is up so that's a metric collection error
        echo "em_error=$status"
        exit 1
    fi
fi
```

Listing 3 Shell script to monitor MySQL engine status

For the purpose of this exercise previous knowledge of MySQL is not really required – it's enough to understand that our program should perform the required checks to find out the status and produce the output line in the form of "em_result=<value>". In our case it will be "em_result=Up" or "em_result=Down". You can find more about MySQL in **[Error! Reference source not found.]**. In case of problems encountered during metric evaluation, the script should print the string in format "em_error=<error message>" and return a non-zero exit code. This is demonstrated in Listing 3.

Let's create our OS-based UDM. We'll name it MySQL Engine Status and it should be of type STRING. Enter the full path to the script from Listing 3, which is /opt/oracle/monitor/mysql/mysql_db_status_udm.sh. In this case, Field "Environment" should be empty. If you need to make your UDM collection script more flexible you might want set here some variables as parameters. The Credentials section should include OS user name and password, for example, mysql. Since our metric is of type STRING we choose the condition operator CONTAINS and the critical threshold value "Down". Let's set it to be collected every five minutes starting immediately.

User Defined String Metric: Script MySQL Engine Status: Last 24 hours

Last Updated **04-Dec-2005 17:10:31 GMT**
View Data **Last 24 hours**

Script **MySQL Engine Status**

Metric Value History

Collection Timestamp	Value
04-Dec-2005 17:10:31	Up
04-Dec-2005 16:30:49	Down
04-Dec-2005 15:20:46	Up

Alert History

Comment for Most Recent Alert

Severity	Timestamp	Message	Last Comment	Details
✓	04-Dec-2005 17:10:31	CLEARED - User Defined Metric MySQL Engine Status returned a value of Up		-
✗	04-Dec-2005 16:30:49	User Defined Metric MySQL Engine Status returned a value of Down		-

Figure 4 MySQL Status metric history

OEM will show changes in the metric value on its history screen as shown in Figure 4. When values match the notification condition, OEM generates alerts that are shown on the target's home page along with other notifications (Figure 5).

Host: **lcbdt201**

Latest Data Collected From Target **04-Dec-2005 16:48:16 GMT**

[Home](#)
[Performance](#)
[Targets](#)
[Configuration](#)

General

Status **Up**

Up Time **3 days**

Logons **4** [View Current Users](#)

Availability **13.24**
(%)
(Last 24 Hours)

Cluster **lcbdt**

Configuration

Operating System [SUSE LINUX Enterprise Server 9 \(x86_64\) 2.6.5 7.191 \(64-bit\)](#)

Hardware Platform [x86_64](#)

IP Address **172.17.44.10**

CPUs **4**

Memory Size (MB) **4797**

Local File Systems (GB) [26.86](#)

Disk Groups Space (GB) [40.7](#)

Alerts

Metric Collection Errors **2**

Metric Name	Severity	Alert Triggered	Value	Last Checked
User Defined String Metric for MySQL Engine Status	✗	04-Dec-2005 16:30:49	Down	04-Dec-2005 16:45:46
Swap Utilization (%)	✗	01-Dec-2005 04:59:09	0	04-Dec-2005 16:47:22

Figure 5 MySQL Engine Status critical alert on home page

Collection errors are displayed in the same way as for standard metrics. I intentionally put a typo in the script, and it produced an error like one shown in the Figure 6.

Error Details

Target	lcbdt201
Type	Host
Metric	User Defined Metrics: MySQL Engine Status
Collection Timestamp	Dec 5, 2005 7:33:02 AM
Error Type	Collection Failure
Message	em_error=/opt/oracle/monitor/mysql/mysql_db_status_udm.sh: line 1: /usr/bin/mysqladmin1: No such file or directory

OK

Figure 6 Collection error for MySQL Engine Status

UDMs Pros and Cons

The main advantage of UDM is that it is a straightforward method which does not require any additional knowledge of the framework and complex manipulations. However, there are number of disadvantages:

- No direct support for user-defined monitoring targets
- Very limited number of collection types (SQL-based and OS script-based)
- No support for collection based on cumulative counters (like V\$SYSSTAT) when the metric's value is calculated based on the difference between previous and current measurements.
- UDM should be defined for each instance of target type, i.e. having 100 database instances will require user to add 100 UDM metrics.
- Credentials are defined separately and different from those provided in target setup. If they are changed for one target than every UDM of that target has to be updated manually.
- Thresholds are defined in a separate screen from other metrics and have no real time status.

All of the above are addressed with user-defined target types.

Creating New Target Types

OEM provides XML-based framework for describing the targets it monitors and manages. XML files are used to store information about target types – what attributes should be defined for every target instance, what metrics are collected, how metrics are calculated, how often, etc.

So how do you create a new target type? Well, the same way in which developers in Oracle Corp. create their targets! All standard targets, such as Oracle database, Listener and Host, are defined using the very same framework. XML files contain target type metadata, which is often called data about data. Recall what you need to define new Oracle database target in OEM – machine name, Oracle home, login credentials, port, and SID. These are attributes required for each instance of an Oracle database target. OEM is also collecting database metrics like performance statistics, tablespace space usage and etc. Information *about* the structure of these attributes and collected statistics is called target type metadata.

Where and how does OEM use target type metadata? OEM 10g Grid Control has a three-tier architecture and includes the following functions:

- One Oracle Agent per host collecting information about monitored targets
- One or more Oracle Management Servers (OMS) receiving information from agents and serving as central access point for end users via browser-based GUI
- An Oracle database hosting a central OEM Repository

Since target monitoring and interactions are done by the Oracle Agent, all target type metadata files are placed on the agent's side. The agent will transfer to OMS all metadata about new targets as well as properties and collected metrics of defined target instances.

Directory \$AGENT_HOME/sysman/admin/dtds contains XML Data Type Definition (DTD) files describing the grammar of XML files used by Grid Control to store metadata – These are : TargetMetadata.dtd, TargetCollection.dtd, Schedule.dtd, TargetInstance.dtd. These files should also serve as reference on possible tags usage in the target type definition. Additional information about XML and DTD can be found in [5].

As an example we will create a new user-defined target type with unique parameters and metrics. It will also show that custom target types are much more flexible and have a wealth of features in comparison to the UDMs we created earlier.

We will take MySQL as example of a new target as it becomes more and more popular even within Oracle shops for performing various less demanding tasks. In addition, it's easy to understand so you can follow examples even if you've never used it.

There are several main steps to create a new target type:

1. Create target type metadata
2. Validate new target type and metric collection
3. Define metrics default collection parameters
4. Distribute metadata and required support files to all hosts
5. Add instances of new target type

Target type metadata file

Target type metadata files are located in \$AGENT_HOME/sysman/admin/metadata. All standard predefined metadata files are located in this directory. You can start a new metadata file from scratch or use one of predefined target types as a starting point and modify it as required. The aforementioned directory is a good place for further education if you need more complex target type examples.

We will call our metadata file "mysql.xml" and it should be placed in \$AGENT_HOME/sysman/admin/metadata. The file content is presented in Listing 4. As any standard XML file, the first line should be the version of XML standard. The second line referencing DOCTYPE is needed to specify the path to DTD so that XML file can be validated for conformance to the rules defined in that DTD. This is a requirement for ILint tool (covered later).

The top level element is named TargetMetadata and it includes unique target type id "mysql" and metadata version "1.0". You will need to change version number if you change the file. When OMS detects that target type version changed, it will reload new metadata. The next level normally includes three elements – Display, Metric and InstanceProperties.

```
<?xml version="1.0" ?>
```

```

<!DOCTYPE TargetMetadata SYSTEM "../dtds/TargetMetadata.dtd">
<TargetMetadata META_VER="1.1" TYPE="mysql">
  <Display>
    <Label NLSID="mysql_name">MySQL</Label>
  </Display>
  <AltSkipCondition METRIC="Response" COLUMN="State"/>
  <Metric NAME="Response" TYPE="TABLE">
    <Display>
      <Label NLSID="resp">Response</Label>
    </Display>
    <TableDescriptor>
      <ColumnDescriptor NAME="Status" TYPE="STRING">
        <Display FOR_SUMMARY_UI="TRUE">
          <Label NLSID="resp_status">Status</Label>
        </Display>
      </ColumnDescriptor>
      <ColumnDescriptor NAME="response_time" TYPE="NUMBER">
        <Display FOR_SUMMARY_UI="TRUE">
          <Label NLSID="resp_time">Response time (ms)</Label>
          <ShortName NLSID="resp_time_short">Response</ShortName>
        </Display>
      </ColumnDescriptor>
    </TableDescriptor>
    <QueryDescriptor FETCHLET_ID="OSLineToken">
      <Property NAME="myhost" SCOPE="INSTANCE">host</Property>
      <Property NAME="myport" SCOPE="INSTANCE">port</Property>
      <Property NAME="myuser" SCOPE="INSTANCE"
OPTIONAL="TRUE">user</Property>
      <Property NAME="mypwd" SCOPE="INSTANCE"
OPTIONAL="TRUE">password</Property>
      <Property NAME="scriptDir"
SCOPE="GLOBAL">/opt/oracle/monitor/mysql</Property>
      <Property NAME="startsWith" SCOPE="GLOBAL">em_result=</Property>
      <Property NAME="errStartsWith" SCOPE="GLOBAL">em_error=</Property>
      <Property NAME="delimiter" SCOPE="GLOBAL">|</Property>
      <Property NAME="command"
SCOPE="GLOBAL">%scriptDir%/mysql_response.sh %myhost% %myport% %myuser%
%mypwd%</Property>
    </QueryDescriptor>
  </Metric>
  <Metric NAME="Load" TYPE="TABLE">
    <Display>
      <Label NLSID="mysql_load">Workload</Label>
    </Display>
    <TableDescriptor>
      <ColumnDescriptor NAME="threads_connected" TYPE="NUMBER">
        <Display FOR_SUMMARY_UI="TRUE">
          <Label NLSID="mysql_load_threads_connected">Threads
Connected</Label>
        </Display>
      </ColumnDescriptor>
      <ColumnDescriptor NAME="threads_running" TYPE="NUMBER">
        <Display FOR_SUMMARY_UI="TRUE">
          <Label NLSID="mysql_load_threads_running">Threads
Running</Label>
        </Display>
      </ColumnDescriptor>
      <ColumnDescriptor NAME="connections" TYPE="NUMBER"
TRANSIENT="TRUE" />
      <ColumnDescriptor NAME="connections_ps" TYPE="NUMBER"
COMPUTE_EXPR="(connections - _connections) / __interval">

```

```

        <Display FOR_SUMMARY_UI="TRUE">
            <Label NLSID="mysql_load_connections_ps">Connections per
second</Label>
        </Display>
    </ColumnDescriptor>
</TableDescriptor>
<QueryDescriptor FETCHLET_ID="OSLineToken">
    <Property NAME="myhost" SCOPE="INSTANCE">host</Property>
    <Property NAME="myport" SCOPE="INSTANCE">port</Property>
    <Property NAME="myuser" SCOPE="INSTANCE"
OPTIONAL="TRUE">user</Property>
    <Property NAME="mypwd" SCOPE="INSTANCE"
OPTIONAL="TRUE">password</Property>
    <Property NAME="scriptDir"
SCOPE="GLOBAL">/opt/oracle/monitor/mysql</Property>
    <Property NAME="startsWith" SCOPE="GLOBAL">em_result=</Property>
    <Property NAME="errStartsWith" SCOPE="GLOBAL">em_error=</Property>
    <Property NAME="delimiter" SCOPE="GLOBAL">|</Property>
    <Property NAME="command" SCOPE="GLOBAL">%scriptDir%/mysql_load.sh
%myhost% %myport% %myuser% %mypwd%</Property>
</QueryDescriptor>
</Metric>
<Metric NAME="procact" TYPE="TABLE">
    <Display>
        <Label NLSID="mysql_procact">Current Processes by Action</Label>
    </Display>
    <TableDescriptor>
        <ColumnDescriptor NAME="action" TYPE="STRING" IS_KEY="TRUE">
            <Display>
                <Label NLSID="mysql_procact_action">Action</Label>
            </Display>
        </ColumnDescriptor>
        <ColumnDescriptor NAME="proccount" TYPE="NUMBER">
            <Display>
                <Label NLSID="mysql_procact_proccount">Number of processes in
action</Label>
            </Display>
        </ColumnDescriptor>
    </TableDescriptor>
    <QueryDescriptor FETCHLET_ID="OSLineToken">
        <Property NAME="myhost" SCOPE="INSTANCE">host</Property>
        <Property NAME="myport" SCOPE="INSTANCE">port</Property>
        <Property NAME="myuser" SCOPE="INSTANCE"
OPTIONAL="TRUE">user</Property>
        <Property NAME="mypwd" SCOPE="INSTANCE"
OPTIONAL="TRUE">password</Property>
        <Property NAME="scriptDir"
SCOPE="GLOBAL">/opt/oracle/monitor/mysql</Property>
        <Property NAME="startsWith" SCOPE="GLOBAL">em_result=</Property>
        <Property NAME="errStartsWith" SCOPE="GLOBAL">em_error=</Property>
        <Property NAME="delimiter" SCOPE="GLOBAL">|</Property>
        <Property NAME="command"
SCOPE="GLOBAL">%scriptDir%/mysql_procact.sh %myhost% %myport% %myuser%
%mypwd%</Property>
    </QueryDescriptor>
</Metric>
<InstanceProperties>
    <InstanceProperty NAME="host">
        <Display>
            <Label NLSID="mysql_host">Hostname</Label>
        </Display>

```

```

</InstanceProperty>
<InstanceProperty NAME="port">
  <Display>
    <Label NLSID="mysql_port">Port</Label>
  </Display>
</InstanceProperty>
<InstanceProperty NAME="user" OPTIONAL="TRUE">
  <Display>
    <Label NLSID="mysql_user">MySQL Username</Label>
  </Display>
</InstanceProperty>
<InstanceProperty NAME="password" CREDENTIAL="TRUE" OPTIONAL="TRUE"
HIDE_ENTRY="TRUE" NEED_REENTER="TRUE">
  <Display>
    <Label NLSID="mysql_password">Password</Label>
  </Display>
</InstanceProperty>
</InstanceProperties>
</TargetMetadata>

```

Listing 4 MySQL target type metadata

Display element

This element is used solely for storing a label that should be displayed by the Grid Control console. It also includes `NLSID` which is used for multiple language support. This element is used on different levels and defines a label that is used by Grid Control for the parent element of `Display`. In this case, it tells OEM that it should use the “MySQL” label for targets of this new type. You can see `TargetMetadata.dtd` file for more information regarding this element, for example, how to assign it a longer description, short name, and even an icon.

InstanceProperties element

`InstanceProperties` is a required element that has to be placed *after* all `Metric` elements. However, we discuss it now to better understand the `Metric` section later on. There must be exactly one `InstanceProperties` descriptor in the root element `TargetMetadata` and it consists of several `InstanceProperty` elements and optionally `DynamicProperties` elements. Each `InstanceProperty` entry specifies an attribute of the target that should be defined by a user in the Grid Console when adding a new target instances. A `DynamicProperty` tag defines attributes that are calculated automatically by the Agent instead. We will not cover them in detail, but just mention that they are calculated using the same method as for the metrics – child `QueryDescriptor` element, covered later in details, defines the collection method. As usual, more reference information is available in the `TargetMetadata.dtd` file.

The following attributes should be specified for each `InstanceProperty` element. You can see them in Listing 4 at the end of the `InstanceProperties` section.

- `NAME` – unique string identifying an attribute within a target type, for example, “host” for machine hostname where MySQL is running.
- `CREDENTIAL` can be either `FALSE` or `TRUE`. Specifying it as true will encode the value of an attribute so that clear text cannot be easily retrieved by unauthorized user. Usernames and passwords are good candidates for that. Default value is `FALSE` when value is stored in clear text. We use it for password property.
- `OPTIONAL` can be either `FALSE` or `TRUE`. When defined as `TRUE` instance property is not required and can be left empty when defining new target instances. We set it to

TRUE for user and password properties as it's possible to configure MySQL to connect without username and password. Default is FALSE.

There are few other handy attributes of the InstanceProperty tag but we are not going to cover them all in this article. By now you know already where to look for more information about those – in the TargetMetadata.dtd file.

Finally each instance property can and should include a child Display tag defining a label used by Grid Control for display purposes. It is the same Display element that we discussed previously.

Metric element

This is the most important and complex element of the target type metadata file. Metric element includes type and name unique within a target type. It also has to include a TableDescriptor element if the metric type is defined as TABLE. TableDescriptor defines columns with names, types and labels. Finally, there is a QueryDescriptor tag that describes the Fetchlet used to collect the values of the metric. Note that all Metric tags have to be placed *before* InstanceProperties element in the XML file.

Fetchlet is a term for metric collection components in OEM 10g including SQL, OS scripts, SNMP, HTTP and others. Various fetchlets also exhibit different behavior in parsing output – they can treat all output lines as a single value, or divide it by lines and even columns. We will discuss this later when it comes to describing it in an XML file.

It's possible to define multiple statistics to be collected for a metric using type TABLE which is useful to reduce the number of calls to collectors and also reduces resource consumption. Using TABLE type it's also possible to define metrics collecting statistics grouped by a key value ("procact" metric in our example returns one row per action that processes perform in the database). Other common metric types are NUMBER and STRING. With simple types like STRING and NUMBER, the fetchlet needs to return a single value. Metric of type TABLE requires a fetchlet returning several columns and/or multiple rows.

TableDescriptor is a simple container for a set of ColumnDescriptor tags. Obviously, each ColumnDescriptor element corresponds to a column returned by a collection Fetchlet in the same order as defined in XML file. Attributes of ColumnDescriptor are NAME (unique within target type) and TYPE that is either NUMBER or STRING. There is also a Display element for each column. We have already discussed this generic tag.

It's important to define a special metric with the name "Response", of type TABLE and a column named "Status". The OEM availability framework will use this column for the target up and down status and historical availability statistics. You will also need to define a critical condition for this metric later. You can see in Figure 7 the availability history for the MySQL target – OEM knows that the Target is down if Response=>Status generates critical alert. On a MySQL target home page (Figure 12) you can see arrow up reflecting current status – it originates as well from the Status column of the Response metric.

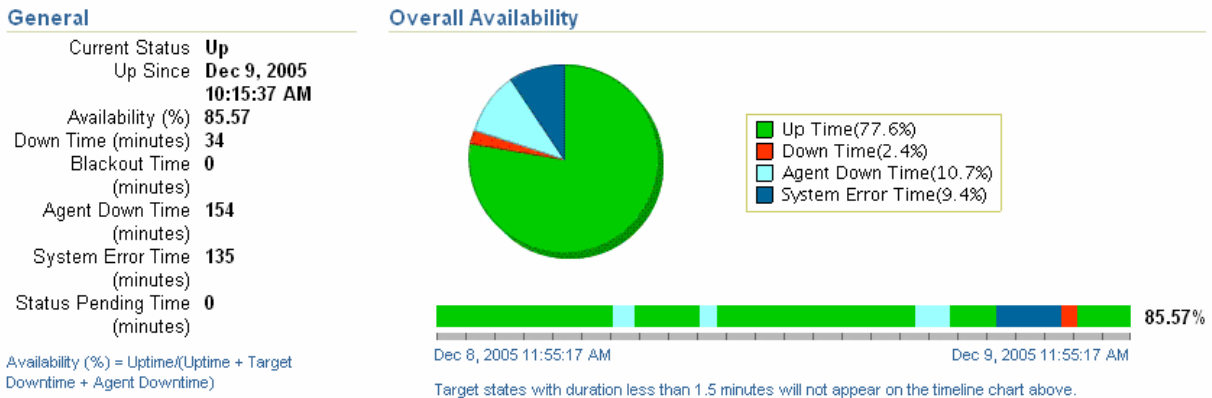


Figure 7 MySQL availability history screen in Grid Control console

We define three metrics – Response, Load and Processes by Action. These are shown later in Figure 8. The metric Response provides “Status” (Up/Down string reflecting state of MySQL engine) and “Response Time” (response to an action for connecting and running simple SELECT). When we are ready with a new target, the Response Time metric will look like one in Figure 10.

Load metric includes “Threads Connected” (kind of analogue for current processes in Oracle), “Threads Running” (i.e. active processes) and “Connections per Second”. Note that the first two reflect the state at the time of collection while the last one represents a value calculated for the period of time since the previous collection. For the latter we used a cumulative statistic provided by MySQL which tracks the number of logons since startup. To implement this functionality, OEM provides transient metrics that are collected but not uploaded to a central OMS. These columns are marked with attribute `TRANSIENT="TRUE"` just like the “connections” columns in Listing 4. Based on current and previous values of this column we can calculate number of user connections for the last collection interval and then divide it by the interval to get number of connections per second. The column “connections_ps” uses attribute `COMPUTE_EXPR="(connections - _connections) / __interval"` to define the formula used to calculate column value. In this formula “connections” identifies the current value of the transient column with the same name. The previous value is referenced by prefixing column name with “_” underscore symbol. The time passed since last collection is referenced via “__interval” (note double underscore in the beginning) and it is returned in seconds.

Finally, there is a metric with `NAME="procact"` to illustrate how OEM can collect multiple rows when some of the columns are defined as key columns. Actions can be considered as one of the work types that the process can do. Using Oracle as an analogy, it looks like a wait event but on a much higher level – more like a step of command execution. The metric returns the number of processes performing each action in the database at the moment of collection. For example, processes can be in “Sleep” state – Idle processes, “Copying to tmp table”, “Locked”, “Sorting for order” and etc. The first column containing action string is marked with the attribute `IS_KEY="TRUE"` so that the values in another column are stored per key value.

All MetricsCollected From Target Dec 9, 2005 6:50:20 AM GMT 

Expand All | Collapse All

Metrics	Thresholds	Collection Schedule	Upload Interval	Last Upload
▼ MySQL-1.lcbdt201				
▼ Current Processes by Action	None	Every 1 Minute	Every Collection	Dec 9, 2005 6:50:20 AM
Number of processes in action	Not Set			
▼ Response	All	Every 1 Minute	Every Collection	Dec 9, 2005 6:54:13 AM
Response time (ms)	Set			
Status	Set			
▼ Workload	Some	Every 1 Minute	Every Collection	Dec 9, 2005 6:50:20 AM
Connections per second	Not Set			
Threads Connected	Set			
Threads Running	Set			

Figure 8 MySQL metrics screen in Grid Control console**QueryDescriptor element**

The `QueryDescriptor` has a single attribute `FETCHLET_ID` that identifies uniquely the fetchlet type used to collect the required information. The various fetchlets are discussed later in this document. Each fetchlet is customized using a set of Property tags.

Each property tag has a `NAME` attribute. Each fetchlet has its own set of required properties with predefined names. The `SCOPE` attribute defines the resolution context. Below are the most important `SCOPE` options:

- **GLOBAL** – the value is resolved in the target type metadata file (i.e. the file that we are building now). In our example, it is a "command" property with value "%scriptDir%/mysql_response.sh %myhost% %myport% %myuser% %mypwd%". Strings within %% are substituted with the property values of the current `QueryDescriptor` element. It is also often used for constant definitions like property delimiter.
- **SYSTEMGLOBAL** – this value is resolved in the context of the `emd.properties` configuration file in directory `$AGENT_HOME/sysman/config`. For example, `emd.properties` contains "perlBin" the path to the Perl executable. This is useful if Perl scripts are used to gather metric values. You can see examples in metadata files of standard targets such as `host.xml`.
- **INSTANCE** – this value is resolved in the context of the target instance, i.e. values defined in the `targets.xml` file (`targets.xml` file is covered later). This value is different for each target instance. For example, the instance property `SID` is different for each instance of the Oracle database target. For the MySQL target, we defined the properties `myhost`, `myport`, `myuser`, and `mypwd` which are calculated for each target instance – remember `InstanceProperty` elements with `NAME` host, port, user, and password?
- **ENV** – the value taken from environment variable.

Other possible options are described in `TargetMetadata.dtd` file.

Fetchlets

The available types of fetchlets in OEM 10g Release 1 are described in [1] chapter 7 in detail. They are listed here to give the reader an overview of how powerful they are.

- OS Command fetchlets are used to run OS scripts or commands and parse the output for the results. There are three types.
 - "OS" fetchlet – the whole output is treated as single value (normally useful for NUMBER or STRING metric types).
 - "OsLines" fetchlet – parses output so that each line is a separate row with one column. Useful for TABLE type metrics with one column and for RAW metrics.
 - "OsLineToken" fetchlet – parses output for lines and columns within a line separated with predefined delimiter ("|" by default).

In our example we use the "OsLineToken" fetchlet and define the following properties:

- "command" – the OS command or script to run
- "delimiter" – separator symbol to parse each line for columns
- startWith – string that a line should begin with, to be included in the result set. The string itself is not included in the metric value. We define it "em_result=" so that all lines without this prefix are ignored.
- errStartWith – string that is similar to "startWith" but used to get an error from the output when it returns non zero exit code.

Other properties are not required for the fetchlet itself but are used to form the value of the global property "command".

- The SQL fetchlet is used to return the result set from the SQL query (similar to what we did with the SQL based UDM).
- SNMP fetchlet is a very powerful tool that allows monitoring devices that provide a SNMP interface. For example, most of modern network components.
- URL Timing fetchlet is a very powerful tool used for the performance monitoring of Web applications. It can return many statistics regarding accessing a web page – status, connect time, avg./max/min response time, etc.
- DMS fetchlet is used to monitor components of Oracle Application Server through Dynamic Monitoring Service.
- HTTP Data set of fetchlets is similar to OS fetchlets but retrieve output from an http:// URL. There are three distinct fetchlet types similar to OS fetchlets.
- URLXML fetchlet retrieves values from XML content retrieved via http:// URL.
- WBEM fetchlet allows monitoring targets that support Web-Based Enterprise Management developed by Distributed Management Task Force, Inc.

Each of OSLineToken fetchlets we use in Listing 4 references a separate shell script. The scripts are included in Listings 5, 6 and 7.

```
#!/usr/bin/sh
# Script to collect response metric for MySQL target. Used for Oracle
EM 10g
# Positional parameters are machine, port, user, password. Last two
are optional
# Author: Alexander Gorbachev, 2005

pHost=$1
pPort=$2
pUser=$3
```

```

pPassword=$4

# Verify required arguments host and port
if [ -z "$pHost" ] ; then
    echo "em_error=MySQL machine is not defined" >&2
    exit 1
fi
if [ -z "$pPort" ] ; then
    echo "em_error=MySQL port is not defined" >&2
    exit 2
fi

# Create connection arguments
connection_arguments="--protocol=tcp --host=$pHost --port=$pPort"
if [ ! -z "$pUser" ] ; then
    connection_arguments="$connection_arguments --user=$pUser"
    if [ ! -z "$pPassword" ] ; then
        connection_arguments="$connection_arguments --password=$pPassword"
    fi
fi

status=`/usr/bin/mysqladmin $connection_arguments ping 2>&1`

if [ "$status" = "mysqld is alive" ] ; then
    time1=`date +%s%N`
    mysql $connection_arguments >/dev/null <<-!
    select * from information_schema.tables limit 1;
    !
    if [ $? -eq 0 ] ; then
        time2=`date +%s%N`
        response_ms=`expr \( $time2 - $time1 \) / 1000000`
    else
        response_ms=""
    fi

    echo "em_result=Up|$response_ms"
else
    # No processes - MySQL is down
    echo "em_result=Down"
fi

```

Listing 5 Shell script for Response metric mysql_resp.sh

```

#!/usr/bin/sh
# Script to collect load metrics for MySQL target. Used for Oracle EM
10g
# Positional parameters are machine, port, user, password. Last two
are optional
# Author: Alexander Gorbachev, 2005

pHost=$1
pPort=$2
pUser=$3
pPassword=$4

```

```

# Verify required arguments host and port
if [ -z "$pHost" ] ; then
    echo "em_error=MySQL machine is not defined" >&2
    exit 1
fi
if [ -z "$pPort" ] ; then
    echo "em_error=MySQL port is not defined" >&2
    exit 2
fi

# Create connection arguments
connection_arguments="--protocol=tcp --host=$pHost --port=$pPort"
if [ ! -z "$pUser" ] ; then
    connection_arguments="$connection_arguments --user=$pUser"
    if [ ! -z "$pPassword" ] ; then
        connection_arguments="$connection_arguments --password=$pPassword"
    fi
fi

output=`mysql $connection_arguments 2>&1 <<-!
show global status;
!`
if [ $? -eq 0 ] ; then
    threads_connected=`echo "$output" | grep "^Threads_connected" |
awk '{print $2}'`
    threads_running=`echo "$output" | grep "^Threads_running" | awk
'{print $2}'`
    connections=`echo "$output" | grep "^Connections" | awk '{print
$2}'`
    echo "em_result=$threads_connected|$threads_running|$connections"
else
    echo em_error=$output
    exit 10
fi

```

Listing 6 Shell script for metric Load mysql_load.sh

```

#!/usr/bin/sh
# Script to collect processes by action metric for MySQL target. Used
for Oracle EM 10g
# Positional parameters are machine, port, user, password. Last two
are optional
# Author: Alexander Gorbachev, 2005

pHost=$1
pPort=$2
pUser=$3
pPassword=$4

# Verify required arguments host and port
if [ -z "$pHost" ] ; then
    echo "em_error=MySQL machine is not defined" >&2
    exit 1
fi
if [ -z "$pPort" ] ; then

```

```

    echo "em_error=MySQL port is not defined" >&2
    exit 2
fi

# Create connection arguments
connection_arguments="--protocol=tcp --host=$pHost --port=$pPort"
if [ ! -z "$pUser" ] ; then
    connection_arguments="$connection_arguments --user=$pUser"
    if [ ! -z "$pPassword" ] ; then
        connection_arguments="$connection_arguments --password=$pPassword"
    fi
fi

# Get process list and take only 5th column - action; sort it
output=`mysql $connection_arguments 2>&1 <<-!
show processlist;
!`

if [ $? -eq 0 ] ; then
    # Take 5th column and sort
    output2="`echo \"$output\" | grep -v '^Id' | awk '{print $5}' |
sort`"
    # For each unique action count number of processes
    for action in `echo "$output2" | uniq` ; do
        echo "em_result=$action|`echo \"$output2\" | grep \"^$action\$\"
| wc -l`"
    done
else
    echo em_error=$output >&2
    exit 10
fi

```

Listing 7 Shell script for metric Processes by Action mysql_procact.sh

Validating Target Type Metadata

There are two steps in the validation process – validate the structure of XML files and test the metric collection. Oracle provides the ILint tool to assist in XML files validation – this can be run while you are writing your metadata file (static validation) to perform continuous checks, as well as ensuring that the collection process works as expected in the end. In addition, the Oracle Agent provides the Metric Browser tool that helps to validate metric collection from your browser.

Validation with ILint

ILint can validate general syntax of an XML file as well as conformance to its respective DTD. For the latter, the XML file should include the DOCTYPE declaration to reference the DTD file which it should be checked against. This is the second line in the Listing 4 and it should include the root element name and relative path to the DTD file. ILINT does more than validation against the DTD; it also validates metadata based on a set of internal rules specific for OEM and reports any errors. For example, ILint makes sure that there is `TableDescriptor` element defined if metric type is defined as `TABLE`.

ILint is called via the `emctl` utility that is also used for other agent management operations such as start, stop and status. A Short help option is available from the command line by typing `emctl ilint`. You can also find more details in [1] chapter 3.

Let's first validate our XML file using `"emctl ilint -m $AGENT_HOME/sysman/admin/metadata/mysql.xml"`. This command reports all violations detected in the content of a target type metadata file. Using `"-c"` option we can also validate the default metric collection definition created in the next step.

In order to verify the collection process we need to create a target instance definition XML file. Normally, the agent keeps these definitions in `$AGENT_HOME/sysman/emd/targets.xml` but we will create file `/opt/oracle/monitor/mysql/test_target.xml` (see Listing 8) only for one target. This instance definition sets target name to `MySQL-1.lcbdt201` and type to `mysql`. Note that it provides values for instance properties which we defined earlier in target type metadata XML file in the `InstanceProperties` section of Listing 4. You should define your own `hostname/port` and perhaps `username/password`.

```
<Targets>
  <Target TYPE="mysql" NAME="MySQL-1.lcbdt201">
    <Property NAME="host" VALUE="lcbdt201"/>
    <Property NAME="port" VALUE="3306"/>
    <Property NAME="user" VALUE="root"/>
  </Target>
</Targets>
```

Listing 8 Target definition XML file test_target.xml

Now we can test metric collection with ILint. See Listing 9 for command line and output. Note that we specified our custom instance definition file with the `-t` option and the metric to be collected with the `-r` option. You can check the "Load" and "procact" metric in the same way.

```
$ emctl ilint -m $AGENT_HOME/sysman/admin/metadata/mysql.sql \
  -i /opt/oracle/monitor/mysql/test_targets.xml
  -t MySQL-1.lcbdt201 -r Response
.....(skipped lines here).....
=====
Target = "MySQL-1.lcbdt201" has the following 3 Properties
=====
Property: host --> Value: lcbdt201 (INSTANCE)
Property: port --> Value: 3306 (INSTANCE)
Property: user --> Value: root (INSTANCE)

=====
Executing Metric = "Response"
=====
Number of Rows = 1

+=====+=====+
| Status      | response_time |
|             | (Response time (ms)) |
| type = STRING | type = NUMBER |
+=====+=====+
| Up          | 17            |
```

```

+-----+-----+
METRIC=Response, TARGET=(mysql,MySQL-1.lcbdt201)
timestamp=2005-12-08 12:40:49
Target Metadata File
/opt/oracle/product/10g/agent10g/sysman/admin/metadata/mysql.xml
Validated Successfully

```

Listing 9 Using ILint to validate metric collection

Validation with Metric Browser

The Oracle agent has an internal component called Metric Browser that shows collected values for all targets. In order to use the Metric Browser, it should be activated first – uncomment the line `enableMetricBrowser=true` in the file `$AGENT_HOME/sysman/config/emd.properties` and restart agent using “`emctl stop agent`” followed by “`emctl start agent`”. In the same `emd.properties` file you should find the variable `EMD_URL` with a value in the format of `http://<host>:<port>/emd/main/`. You will use later `<host>` and `<port>` for Metric Browser URL <http://<host>:<port>/emd/browser/main>.

Before we can view the metrics of our new target instance it should be registered with an agent. We can do it using the file `test_target.xml` created earlier and add it with the command “`emctl config agent addtargets /opt/oracle/monitor/mysql/test_target.xml`”. MySQL-1.lcbdt201 target will be added to the standard `targets.xml` file. Now you can open Metric Browser URL and select MySQL-1.lcbdt201 target to view its metrics. Figure 9 demonstrates the Web page with the “`procact`” metric.

```

METRIC=procact
TARGET=(mysql,MySQL-1.lcbdt201)

```

action (Action)	proccount(Number of processes in action)
STRING	NUMBER
Query	1

```
timestamp=2005-12-08 22:24:56
```

Figure 9 Using Metric Browser for "procact" metric validation

Default Collections

Target type metadata is not enough for a fully functional target. We need to define the schedule of metric collection and notification conditions. This is done using the default collection XML files in `$AGENT_HOME/sysman/admin/default_collection`. These files have normally the same name as the target type metadata file, i.e., `mysql.xml` in our example. Collection attributes can be changed in the Grid Control console for each target instance. The content of the file is shown in Listing 10. The root element must be `TargetCollection` and the `TYPE` attribute should match the `TYPE` attribute of the `TargetMetadata` root element defined earlier in another file. The file `TargetCollection.dtd` describes the structure of default collection XML file and should serve as reference just like `TargetMetadata.dtd`.

```
<?xml version="1.0" ?>
```

```

<!DOCTYPE TargetCollection SYSTEM "../dtds/TargetCollection.dtd">
<TargetCollection TYPE="mysql">
  <CollectionItem NAME="Response" UPLOAD="YES" UPLOAD_ON_FETCH="TRUE">
    <Schedule>
      <IntervalSchedule INTERVAL="5" TIME_UNIT="Min"/>
    </Schedule>
    <Condition COLUMN_NAME="Status" CRITICAL="Down" OPERATOR="CONTAINS"/>
    <Condition COLUMN_NAME="response_time" WARNING="100" CRITICAL="300"
OPERATOR="GT" OCCURRENCES="2"/>
  </CollectionItem>
  <CollectionItem NAME="Load" UPLOAD="YES">
    <Schedule>
      <IntervalSchedule INTERVAL="5" TIME_UNIT="Min"/>
    </Schedule>
    <Condition COLUMN_NAME="threads_connected" WARNING="1000"
OPERATOR="GT"/>
    <Condition COLUMN_NAME="threads_running" WARNING="50" CRITICAL="100"
OPERATOR="GT" OCCURRENCES="2"/>
  </CollectionItem>
  <CollectionItem NAME="procact" UPLOAD="YES">
    <Schedule>
      <IntervalSchedule INTERVAL="5" TIME_UNIT="Min"/>
    </Schedule>
  </CollectionItem>
</TargetCollection>

```

Listing 10 Default collections XML file

[MySQL Engine: MySQL-1.lcbdt201](#) > [All Metrics](#) >
Response time (ms): Last 24 hours

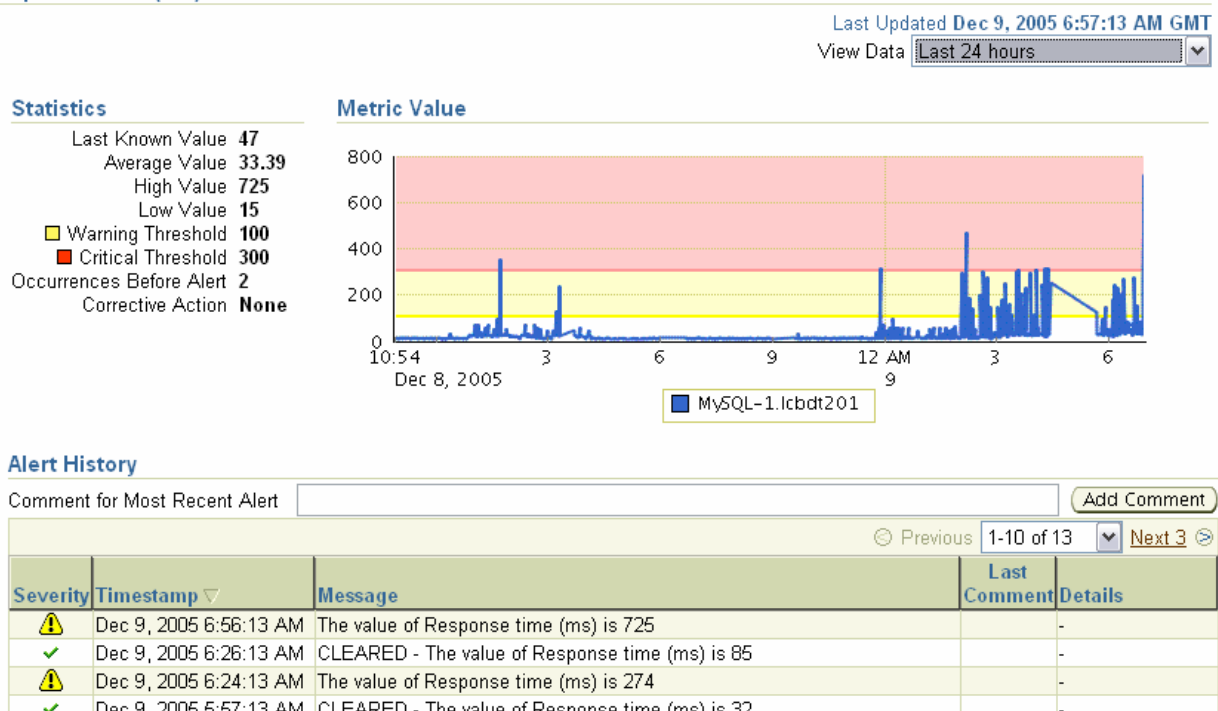


Figure 10 MySQL Response Time metric

The structure is very simple – a root element has one or more `CollectionItem` elements. Each `CollectionItem` corresponds to one metric defined in the target type metadata. The

NAME attribute of `CollectionItem` should match that of the `Metric` element so that OEM knows collection of which metric is described. Inside `CollectionItem` is defined a `Schedule` tag. The most useful component of the `Schedule` is `IntervalSchedule` which defines the frequency of metric collection.

Another child element of `CollectionItem` is `Condition` that optionally can be defined for each column of the metrics. `Condition` includes an OPERATOR that defines logical operation, WARNING and CRITICAL thresholds as well as the number of occurrences that should happen before notification is generated. We use the operator CONTAINS to check if the Status column contains the string "Down" which triggers a critical alert. The Operator GT equates to "greater than" and is a default operator. GT triggers a notification when the column value is greater than the defined warning or critical thresholds. When we finish, the Response Time metric history can look like one shown in Figure 10. Note warning and critical thresholds' colored areas above 100 and 300 ms respectively. Since we set two occurrences required for the alert, short single spikes above 300 didn't produce any critical alerts.

As mentioned already it is important to define a critical condition for the Status column of the Response metric as this is used by the OEM framework for target availability (see Figure 7).

Putting It All Together

Target type metadata and default collection XML files should be distributed to each host in respective directories `$AGENT_HOME/sysman/admin/metadata` and `$AGENT_HOME/sysman/admin/default_collection`. Agents should be reset using `emctl reload agent` command on every host. Note that all the required shell scripts should be distributed as well. For our MySQL target type we have three shell scripts `mysql_resp.sh`, `mysql_load.sh` and `mysql_procact.sh` located in `/opt/oracle/monitor/mysql`.

Add MySQL Engine

Properties

* Name

Type **MySQL Engine**

Name	Value
Hostname	<input type="text" value="lcbdt201"/>
Port	<input type="text" value="3301"/>
MySQL Username	<input type="text" value="root"/>
Password	<input type="text"/>

Monitoring

Oracle has automatically enabled monitoring for this target's availability and performance, so no further monitoring configuration is necessary. You can edit the metric thresholds from the target's homepage.

Figure 11 New MySQL target definition from Grid Control Web console

When distribution is done we can add new targets from the OEM. An example of a new MySQL target screen is in Figure 11. Basically it's the same information we provided in the "test_targets.xml" file earlier during ILint validation. Behind the scenes OMS will create an XML file with the target instance properties we provide on this screen and send it to the agent for registration. The Agent will add this target instance to its standard targets.xml file in \$AGENT_HOME/sysman/emd. It's still possible to use the command `emctl config agent addtargets`. This is a preferred method when adding many similar targets as it can be easily scripted.

MySQL Engine: MySQL-1.lcbdt201

Page Refreshed Dec 9, 2005 6:56:40 AM GMT

Home

General

Status **Up**

Availability (%) **87**
(Last 24 Hours)

Host [lcbdt201](#)

Alerts

Metric	Severity	Alert Triggered	Last Value	Last Checked
Response time (ms)		Dec 9, 2005 6:56:13 AM	725	Dec 9, 2005 6:56:13 AM

Figure 12 MySQL target home page

We are finally done and can enjoy our new target – Figure 12.

Where To Go Next?

You should reference DTD files for additional information on metadata structure. Oracle documentation [1] also covers many details and describes a sample target – a custom web application. The source code for this application is available from [3] so you can download and install it in your own environment. The new extensibility guide for OEM Release 2 is under development at the time of writing.

OEM 10g Release 2 comes with many new features and they are also reflected in the Extensibility area. Besides the few handy enhancements mentioned already, there are many improvements in metadata structure. There are also new fetchlets. An excellent example is the SGA Fetchlet for SGA direct access – isn't that a tool all DBAs have been dreaming about?

Release 2 brings a new important concept of Extensibility Plug-ins. This feature allows third-party vendors to provide easy-to-install plug-ins for their software packages or hardware devices. The Plug-in package is uploaded only once using web GUI of central OMS and then OEM takes care of the deployment to required agents without any manual actions on remote hosts. Plug-in package can also include customized Web pages for the target type – custom home page, custom reports and graphs. You can find more details on the plug-ins concept in [2].

Conclusion

We have traversed the complete route from creating simple User-Defined Metric to adding a fairly sophisticated, completely new target type for MySQL database. Even though we covered only the tip of the iceberg, it's an important step towards getting you started in enterprise monitoring solutions based on Oracle Enterprise Manager 10g Grid Control.

References

1. "Extending Enterprise Manager, 10g Release 1 (10.1) for UNIX and Windows", Part No. B12014-01,
www.oracle.com/technology/products/oem/extensions/extensibility_guide.pdf
2. Extending Enterprise Manager 10g Grid Control for Complete Data Center Management,
www.oracle.com/technology/products/oem/extensions/EM_Extensibility_Whitepaper.pdf
3. Enterprise Manager Extensibility Kit,
www.oracle.com/technology/products/oem/extensions/emx_10.1.0.jar
4. Oracle Streams Advanced Queuing User's Guide and Reference, Part Number B14257-01
5. W3Schools Online Web Tutorials at <http://www.w3schools.com/xml/> and <http://www.w3schools.com/dtd/>
6. MySQL online documentation at <http://dev.mysql.com/doc/>

Author's homepage is www.oracloid.com.

He can be contacted by email – gorbyx [at] gmail [dot] com.