



Operations Agent

Software Version: 12.05

For Windows®, HP-UX, Linux, Solaris, and AIX operating systems

User Guide

Document Release Date: December 2017

Software Release Date: December 2017



Hewlett Packard
Enterprise

Legal Notices

Warranty

The only warranties for Seattle SpinCo, Inc and its subsidiaries ("Seattle") products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Seattle shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Except as specifically indicated, valid license from Seattle required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© 2015-2017 EntIT Software LLC, a Micro Focus company

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of the Microsoft group of companies.

UNIX® is a registered trademark of The Open Group.

Acknowledgements

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>).

This product includes software written by Tim Hudson (tjh@cryptsoft.com).

This product includes an interface of the 'zlib' general purpose compression library, which is Copyright ©1995-2002 Jean-loup Gailly and Mark Adler.

Documentation Updates

To check for recent updates or to verify that you are using the most recent edition of a document, go to: <https://softwaresupport.hpe.com/>.

This site requires that you register for an HPE Passport and to sign in. To register for an HPE Passport ID, click **Register** on the HPE Software Support site or click **Create an Account** on the HPE Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HPE sales representative for details.

Support

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

Most of the support areas require that you register as an HPE Passport user and to sign in. Many also require a support contract. To register for an HPE Passport ID, click **Register** on the HPE Support site or click **Create an Account** on the HPE Passport login page.

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

HPE Software Solutions Now accesses the HPE Software Solution and Integration Portal website. This site enables you to explore HPE Product Solutions to meet your business needs, includes a full list of Integrations between HPE Products, as well as a listing of ITIL Processes. The URL for this website is <https://softwaresupport.hpe.com/km/KM01702731>.

Contents

Part I: Introduction	16
Conventions Used in this Document	17
Part II: Configuration	18
Chapter 1: Working with the Operations Agent	19
Configuring the Monitor Agent	19
Configuring the Agent to Monitor MIB Objects	19
Persistence of Monitored Object	20
Enhancing Security Parameters to Perform SNMPv3 GET	21
Enabling opcmna to perform SNMPv3Get	22
Using the SourceEX API to Add Security Parameters to the Policy	25
Configuring the Performance Collection Component Remotely	28
Before You Begin	28
Deploy the OA-PerfCollComp-opcmmsg Policy	29
Configuring the Performance Collection Component	29
Configure the parm File	29
From Operations Manager for Windows	30
From Operations Manager on UNIX/Linux 9.10	30
Configure the alarmdef File	31
From Operations Manager for Windows	32
From Operations Manager on UNIX/Linux 9.10	32
Remotely Working with the Operations Agent	33
Configuring the SNMP Trap Interceptor	34
Configuring the SNMP Trap Interceptor for SNMPv3 Traps	36
Configuring opctrapi to Intercept SNMPv3 Traps	37
Encrypting Password with the opcpwcrpt Utility	39
Enhancing SNMP Trap Interceptor to Intercept Traps Based on the Object ID of the Varbind	40
Enabling opctrapi to Use NETSNMP Logging	41
Integrating Operations Agent with NNMi	42
Integrating with NNMi Northbound Interface to Associate the Message Severity	44

Configuring SNMP Trap Interceptor to Enhance Message Severity	44
Integrating NNMi Northbound Interface to Add CMA on the OM Console	45
Configure SNMP Trap Interceptor to Create CMAs from NNMi CIA on the OM Console	46
Integrating with NNMi Trap Forwarding Interface to Assign Source of SNMP Trap	48
Configure SNMP Trap Interceptor to Derive the Source Node Name	48
Configuring the Message Storm Suppression	49
Configuring Message Storm Detection and Suppression	51
Checking the Message Rate	54
Configuring the Backup Server	54
Configuring the RTMA Component	56
Checking the License for Perfd Process	56
Modifying the Settings	56
Monitoring Operations Agent on IPv6 connections	59
Restricting Access	59
Configuring the Security Component for Asymmetric Key	60
Configuring the Security Component for Symmetric Key	63
Configuring the Security Component with Hash Algorithm	65
Configuring FIPS Compliant Operations Agent	67
Prerequisites to make Operations Agent FIPS Compliant	68
Enabling FIPS Compliance using FIPS_tool	68
Configuration Settings in FIPS Mode	70
Verifying if Operations Agent is running in FIPS mode	70
Troubleshooting	71
Configuring the OPCMONA_POL_RUN_WITH_SHELL Variable	72
Configuring the Control Component Variable	73
Troubleshooting	74
Registering the user-defined process with OvCtrl	74
Monitoring Windows Event Logs	81
Monitor Applications and Services Event Logs from OM for Windows	83
Monitor Applications and Services Event Logs from OM on UNIX/Linux 9.xx	84

Chapter 2: Adviser for the RTMA Component	87
Alarms and Symptoms	87
Working of the Adviser Script	87
Using Adviser	88
Running the Adviser Script on Multiple Systems	89
Adviser Syntax	89
Syntax Conventions	90
Comments	90
Conditions	90
Constants	91
Expressions	91
Metric Names in Adviser Syntax	92
Printlist	93
Variables	94
ALARM Statement	94
ALERT Statement	95
ALIAS Statement	95
ASSIGNMENT Statement	96
COMPOUND Statement	96
EXEC Statement	96
IF Statement	97
LOOP Statement	97
PRINT Statement	98
SYMPTOM Statement	98
Chapter 3: Performance Alarms	100
Processing Alarms	100
Alarm Generator	101
Sending SNMP Traps to Network Node Manager	103
Sending Messages to OM	103
Executing Local Actions	104
Errors in Processing Alarms	105
Analyzing Historical Data for Alarms	105
Examples of Alarm Information in Historical Data	105
Alarm Definition Components	106
Alarm Syntax Reference	107

Alarm Syntax	107
Syntax Conventions	108
Common Elements	108
Metric Names	109
Messages	110
ALARM Statement	110
Syntax	111
How it is Used	113
Examples	113
ALERT Statement	115
Syntax	115
How it is Used	115
Example	115
EXEC Statement	116
Syntax	116
How it is Used	117
Examples	117
PRINT Statement	118
IF Statement	118
Syntax	118
How it is Used	119
Example	119
LOOP Statement	120
Syntax	120
How it is Used	120
Example	120
INCLUDE Statement	121
Syntax	121
How it is Used	121
Example	121
USE Statement	122
Syntax	122
How it is Used	122
VAR Statement	124
Syntax	124

How it is Used	124
Examples	124
ALIAS Statement	125
Syntax	125
How it is Used	125
Examples	125
SYMPTOM Statement	126
Alarm Definition Examples	126
Example of a CPU Problem	126
Example of Swap Utilization	127
Example of Time-Based Alarms	127
Example of Disk Instance Alarms	128
Customizing Alarm Definitions	128
Chapter 4: Operations Agent in a Secure Environment	130
Policies	130
HTTPS Mode of Communication	131
Benefits of the HTTPS Communication	131
Communication Broker	133
Firewall Scenarios	134
HTTPS-Based Security Components	135
Certificates	137
Operations Agent Certificate Server	138
Certification Authority	138
Certificate Client	138
Root Certificate Update and Deployment	139
Part III: Using the Operations Agent Performance Collection	
Component	140
Chapter 5: Managing Data Collection	141
Using the Metrics Datastore	141
Collecting Data	142
Verifying the Status of the oacore Process	143
Starting the oacore Process	144
Verifying Data Logging	145
Controlling Disk Space Used by Database Files	146

Controlling Disk Space Used by Database Files that Store the Default Performance Metric Classes	146
Controlling Disk Space Used by Database Files that Store the Custom Data	148
Stopping and Restarting Data Collection	149
Stopping Data Collection	149
Restarting Data Collection	150
Daylight Savings	150
Changing System Time Manually	151
Using the parm File	151
Installing the Operations Agent 12.xx	151
Upgrading to the Operations Agent 12.xx	152
Modifying the parm File	153
parm File Parameters	154
Parameter Descriptions	158
ID	159
Log	159
Thresholds	161
Procthreshold	161
appthreshold	163
diskthreshold	163
bynetifthreshold	163
fsthreshold	163
lvthreshold	164
bycputhreshold	164
subprocinterval	164
gapapp	165
fstypes	165
wait	167
Size	167
javaarg	167
Flush	168
project_app	168
proclist	168
appproc	169

proccmd	169
ignore_mt	169
cachemem	173
Application Definition Parameters	174
Application Name	175
File	176
argv1	177
cmd	178
User	178
Group	179
Or	179
Priority	179
Application Definition Examples	180
Configuring Data Logging Intervals	181
Configuring Data Collection for Frames	181
Task 1: Configure Passwordless SSH Access	182
Configuring passwordless SSH Access	182
Verifying passwordless SSH Access	183
Task 2: Enable Frame Utilization Monitoring on the HMC System	183
Task 3: Configure the Operations Agent	183
Enabling the Global and Process System Call Metrics for GlancePlus on Linux	184
Configuring the Metric Collection using init_fttrace.sh	184
Configuring the Metric Collection using Manual Steps	186
Normalizing CPU Metrics on Hyper-Threading or Simultaneous Multi- Threading-Enabled Systems	188
Logging Metrics Calculated with the Core-Based Normalization	189
Chapter 6: Using the Utility Program	191
Running the Utility Program	191
Utility Scan Report	193
Utility Commands	194
analyze	194
checkdef	195
detail	195
help	196

filename	196
parmfile	196
scan	197
Chapter 7: Using the Extract Program	198
Running the Extract Program Using Command Line Interface	198
Using the Export Function	200
How to export data?	200
Default Performance Metric Classes	201
Output Files	202
Export Template File Syntax	204
Parameters	205
Output of Exported Files	206
Producing a Customized Output File	208
Notes on ASCII Format	208
Extract Commands	209
Chapter 8: Using the cpsh Program	214
Using the Interactive Mode	214
View Real-Time Metrics	215
Modify a Metric Class	215
View All the Available Metrics	216
Organize a Metric Class	216
View Metric Help	216
View Summarized Metric Data	217
Enabling Threshold and Filter Conditions	217
Chapter 9: Building Collections of Performance Counters on Windows ...	219
Building a Performance Counter Collection	219
Managing a Performance Counter Collection	219
Tips for Using Extended Collection Builder and Manager	220
Administering ECBM from the Command line	221
Chapter 10: Overview of Baselineing	223
Configuring Baseline on the Operations Agent Node	223
How to troubleshoot when Baselineing is not functioning?	227
Chapter 11: Overview of Node Resolution	230
Chapter 12: Logging and Tracing	236
Logging	236

Configure the Logging Policy	237
Tracing	238
Identify the Application	239
Set the Tracing Type	240
Introduction to the Trace Configuration File	241
Syntax	241
Create the Configuration File	241
Enabling Tracing and Viewing Trace Messages with the Command-Line Tools	242
Enabling Tracing and Viewing Trace Messages with the Tracing GUI ..	243
Enable the Tracing Mechanism	243
View Trace Messages	244
Use the Trace List View	246
Use the Procedure Tree View	247
Filter Traces	248
Using the Tracing GUI	249
Part IV: Logging Custom Data	255
Chapter 13: Using Perl Application Programming Interface for Submitting Custom Data	256
Submitting Custom Data	257
How to Submit Custom Data to the Metrics Datastore	259
Use Case to Submit Data to Datastore Using APIs	260
Special Characters Usage in Domain, Class and Metric Names	263
Chapter 14: Overview of Data Source Integration	264
How DSI Works	265
Chapter 15: Creating a Model to Log DSI Metrics into the Metrics Datastore	267
Creating a Class Specification File	267
Class Specification Syntax	268
Class Description	269
Syntax	269
CLASS	269
LABEL	270
Records per Hour	270
Default Settings	271

Sample Class Specification	271
Metrics Descriptions	273
LABEL	274
Summarization Method	275
Compiling the Class Specification File Using the DSI Compiler	276
sdlcomp Compiler	276
Changing a Class Specification	276
Chapter 16: Logging DSI Metrics into the Metrics Datastore	278
Syntax	278
dsilog Logging Process	279
How dsilog Processes Data	279
Managing Data with sdlutil	280
Syntax	280
Chapter 17: Using the DSI Data Logged into the Metrics Datastore	281
Defining Alarms for DSI Metrics	281
Alarm Processing	282
Exporting DSI Data	282
Example of Using Extract to Export DSI Log File Data	282
Viewing Data in Performance Manager	283
Chapter 18: Examples of Data Source Integration	284
Writing a dsilog Script	284
Example 1 - Problematic dsilog Script	284
Example 2 - Recommended dsilog Script	285
Logging vmstat Data	285
Creating a Class Specification File	286
Compiling the Class Specification File	286
Starting the dsilog Logging Process	287
Accessing the Data	287
Logging the Number of System Users	288
Chapter 19: Using Metric Streaming	290
Streaming Metrics	290
Registration	296
Data Submission	298
Special Characters in Registration and Data Submission	301
Configuring Publish Interval	302

Resource Utilization and Scalability	302
Chapter 20: Working of Streaming Edition of Operations Agent	303
Part V: Transaction Tracking	309
Chapter 21: What is Transaction Tracking?	310
Improving Performance Management	310
Benefits of Transaction Tracking	310
Client View of Transaction Times	311
Transaction Data	311
Service Level Objectives	312
A Scenario: Real Time Order Processing	312
Requirements for Real Time Order Processing	312
Preparing the Order Processing Application	313
Monitoring Transaction Data	313
Guidelines for Using ARM	314
Chapter 22: How Transaction Tracking Works	316
Support of ARM 2.0	316
Support of ARM API Calls	317
arm_complete_transaction Call	317
Sample ARM-Instrumented Applications	318
Specifying Application and Transaction Names	318
Transaction Tracking Daemon (ttd)	319
ARM API Call Status Returns	320
Measurement Interface Daemon (midaemon)	321
Transaction Configuration File (ttd.conf)	322
Adding New Applications	322
Adding New Transactions	322
Changing the Range or SLO Values	323
Configuration File Keywords	323
tran	323
range	324
slo	324
Configuration File Format	325
Configuration File Examples	326
Overhead Considerations for Using ARM	328
Guidelines	328

Disk I/O Overhead	328
CPU Overhead	329
Memory Overhead	329
Chapter 23: Getting Started with Transactions	331
Before you start	331
Setting Up Transaction Tracking	331
Defining Service Level Objectives	332
Modifying the Parm File	333
Collecting Transaction Data	333
Error Handling	333
Limits on Unique Transactions	334
Customizing the Configuration File (optional)	334
Monitoring Performance Data	336
Alarms	337
Chapter 24: Transaction Tracking Messages	338
Chapter 25: Transaction Metrics	339
Chapter 26: Transaction Tracking Examples	340
Pseudocode for Real Time Order Processing	340
Configuration File Examples	343
Example 1 (for Order Processing Pseudocode Example)	343
Example 2	343
Example 3	344
Example 4	344
Chapter 27: Advanced Features	346
How Data Types are Used	346
User-Defined Metrics	347
Chapter 28: Transaction Libraries	349
ARM Library (libarm)	349
C Compiler Option Examples by Platform	353
ARM NOP Library	354
Using the Java Wrappers	354
Examples	355
Setting Up an Application (arm_init)	355
Syntax:	355
Setting Up a Transaction (arm_getid)	355

Setting Up a Transaction With UDMs	355
Adding the Metrics	356
Setting the Metric Data	356
Setting Up a Transaction Without UDMs	357
Setting Up a Transaction Instance	357
Starting a Transaction Instance (arm_start)	357
Starting the Transaction Instance Using Correlators	358
Requesting a Correlator	358
Passing the Parent Correlator	358
Requesting and Passing the Parent Correlator	358
Retrieving the Correlator Information	359
Starting the Transaction Instance Without Using Correlators	359
Updating Transaction Instance Data	359
Updating Transaction Instance Data With UDMs	359
Updating Transaction Instance Data Without UDMs	360
Providing a Larger Opaque Application Private Buffer	360
Stopping the Transaction Instance (arm_stop)	360
Stopping the Transaction Instance With a Metric Update	360
Stopping the Transaction Instance Without a Metric Update	361
Using Complete Transaction	361
Using Complete Transaction With UDMs:	361
Using Complete Transaction Without UDMs:	362
Further Documentation	362
Part VI: Troubleshooting	363
Operations Monitoring Component	363
Performance Collection Component	368
RTMA	370
GlancePlus	370
hpsensor	371
Other	371
Send documentation feedback	374

Part I: Introduction

Operations Agent helps you to monitor a system by collecting metrics that indicate the health, performance, availability, and resource utilization of essential elements of the system.

With its embedded data collector, **oacore**, the Operations Agent continuously collects performance and health data across your system and stores the collected data into the [Metrics Datastore](#).

When you use the Operations Agent in conjunction with OM, Smart Plug-ins (SPIs) and Operations Manager i (OMi) you can add the capability to monitor business applications, infrastructure (system resources) as well as application workloads running on the monitored systems.

The following features enhance the data collection and monitoring capabilities of Operations Agent:

Feature	Description
parmfile	The data collection mechanism of the oacore data collector is controlled by the settings in the parm file. Based on the classes defined in the parm file, the oacore data collector, collects a large set of data that represents a wide view of the health and performance of the system.
Utility and Extract	You can use tools such as Utility and Extract to view specific information stored in the Metrics Datastore.
Custom Data Logging	You can either use custom data logging APIs or DSI to log custom data into the Metrics Datastore.
Baselining	You can use the process of baselining to compute and provide reference values to analyze performance trends and dynamically set optimal threshold values.
Real Time Metrics Access	The Real Time Metrics Access (RTMA) component provides real time access to system performance metrics.
alarmdef file	You can define alarms in the alarmdef file. As data is logged by oacore or other collectors, it is compared with the alarm definitions in the alarmdef file. When the monitored metrics meet or exceed the defined conditions, an alert or action is triggered.
Transaction Tracking	You can track transactions as they progress through applications. Transaction tracking provides a client view of elapsed time from the beginning to the end of a transaction, helps you manage Service Level Agreements (SLAs), and generate alarms when Service Level Objectives (SLOs) exceed the conditions defined in the alarmdef file.

Conventions Used in this Document

The following conventions are used in this document:

Convention	Description
<p><OvInstallDir></p> <p>The installation directory for the Operations Agent.</p>	<p><OvInstallDir> is used in this document to denote the following location:</p> <ul style="list-style-type: none">• <i>On Windows:</i> %ovinstalldir%• <i>On HP-UX/Linux/Solaris:</i> /opt/OV/• <i>On AIX:</i> /usr/lpp/OV/
<p><OvDataDir></p> <p>The directory for Operations Agent configuration and runtime data files.</p>	<p><OvDataDir> is used in this document to denote the following location:</p> <ul style="list-style-type: none">• <i>On Windows:</i> %ovdatadir%• <i>On HP-UX/Linux/Solaris:</i> /var/opt/OV/• <i>On AIX:</i> /var/opt/OV/
<p><OvInstallBinDir></p> <p>The bin directory contains all the binaries (executables) of Operations Agent.</p>	<p><OvInstallBinDir> is used in this document to denote the following location:</p> <ul style="list-style-type: none">• <i>On Windows x64:</i> %ovinstalldir%\bin\win64\• <i>On Windows x86:</i> %ovinstalldir%\bin\• <i>On HP-UX/Linux/Solaris:</i> /opt/OV/bin/• <i>On AIX:</i> /usr/lpp/OV/bin/

Part II: Configuration

When Operations Agent is installed in an OM based management environment, you can monitor and manage systems and applications deployed in your network environment from a central console. You can use different components of the Operations Monitoring Component after deploying the OM policies on the node. The configuration variables enable you to configure the default behavior of the Operations Agent. You can use the `ovconfchg` command to assign desired values to these variables. You can configure the Monitor Agent, SNMP Trap Interceptor, Message Storm suppression, Backup Server, RTMA Component, and the Security Components.

Chapter 1: Working with the Operations Agent

After configuring the data collection mechanism, if you want to use the agent in conjunction with OM, you can use different components of the Operations Monitoring Component by deploying OM policies on the node. For example, if you deploy a measurement threshold policy, the monitor agent component starts monitoring. Although you can provide most of the monitoring details in the OM policies, some components might still require additional configuration to be performed on the node.

Configuring the Monitor Agent

You can start and configure the monitor agent to monitor different sources. When you deploy a measurement threshold policy on a node, the monitor agent component starts operating. Based on the specification in the policies, the agent starts monitoring objects from the following types of sources:

- **External:** An external program that can send numeric values to the agent.
- **Embedded Performance Component:** The data available in the agent's datastore.
- **MIB:** Entries in the Management Information Base (MIB).
- **Real-Time Performance Management:** Windows performance logs and alerts.
- **Program:** An external program that is scheduled on the agent system and sends numeric values to the agent.
- **WMI:** The WMI database.

To use OM policies to monitor the objects from the above sources, see the following topics:

- *For OM on Windows:* See the *Event Policy Editors* section in the *OM for Windows Online Help*.
- *For OM on UNIX/Linux:* See the *Implementing Message Policies* section in the *OM for UNIX 9.10 Concepts Guide*.

Configuring the Agent to Monitor MIB Objects

After you deploy the measurement threshold policies (with the Source type set to MIB) on the node, the monitor agent starts querying the MIB objects that can be accessed with the public community string. If you want to configure the monitor agent to use a non-default community string, follow these steps:

1. Log on to the node with the root or administrative privileges.
2. Go to the command prompt (shell).
3. Go to the `<OvInstallBinDir>` directory.
4. Run the following command:

- To use a non-default community string:

```
ovconfchg -ns eaagt -set SNMP_COMMUNITY <community_string>
```

In this instance, `<community_string>` is the non-default community string of your choice.

- To use different community strings:

```
ovconfchg -ns eaagt -set SNMP_COMMUNITY_LIST <list_of_community_strings>
```

In this instance, `<list_of_community_strings>` is a comma-separated list of community strings of your choice. The Operations Agent processes the list of community strings in the order you specified them with the above command.

For example:

```
ovconfchg -ns eaagt -set SNMP_COMMUNITY_LIST "C1,C2,C3"
```

The Operations Agent first tries to establish an SNMP session with the nodes and attempts to perform an SNMP Get operation for the OIDs using the community string C1. If the operation is not successful, the Operations Agent performs the same operation with the community string C2, and so on.

Note: If the Operations Agent fails to use all the community strings specified with `SNMP_COMMUNITY_LIST`, it tries to use the community string specified with `SNMP_COMMUNITY`. If the agent fails to get data with all the specified community string, it starts using the default public community string.

Persistence of Monitored Object

You can configure the Operations Agent to periodically store the values of monitored objects and session variables. Storing the values of monitored objects and session variables ensures that the values are preserved and available for use in the event of an interruption or failure.

The `OPC_MON_SAVE_STATE` variable enables you to configure the agent to preserve the values of monitored objects and session variables.

To make sure that the agent is configured to periodically store values of monitored objects and session variables, follow these steps:

1. Log on to the node as root or administrator.
2. Run the following command:

```
ovconfchg -ns eaagt -set OPC_MON_SAVE_STATE TRUE
```

The agent starts preserving the values of monitored objects and session variables.

Installation of the Operations Agent 12.05 affects the OPC_MON_SAVE_STATE variable in the following way:

- If you did not set the variable to any values prior to installing the Operations Agent 12.05, the OPC_MON_SAVE_STATE variable assumes the value FALSE.
- If you used the ovconfchg command to set a value (TRUE or FALSE) for the variable prior to installing the Operations Agent 12.05, the configured value remains in effect after the installation process is complete.

For example, if you used the command `ovconfchg -ns eaagt -set OPC_MON_SAVE_STATE TRUE` before installing the version 12.05, the same value (TRUE) is retained after the installation of the Operations Agent 12.05.

Enhancing Security Parameters to Perform SNMPv3 GET

On a Operations Agent node, the monitoring agent component (**opcmona**) monitors MIB objects after the Measurement Threshold policy is deployed from the OM management server.

Note: Creating SNMPv3 **opcmona** policies is not supported on Operations Manager for Windows and Operations Manager i.

To query the MIB data, **opcmona** performs **SNMPv3 GET** on the specified Object IDs (OIDs). Additional security parameters are required to enable **opcmona** to perform **SNMPv3Get**. These parameters are listed in the following table:

Parameters	Description
SNMPV3_ENGINEID	Specifies the SNMP Engine ID that is used to uniquely identify SNMP entities.
SNMPV3_USER	Specifies the SNMPv3 user name created by the administrator.

SNMPV3_AUTHTYPE	<p>Specifies the protocols used to encrypt the password.</p> <p>You can either use Message Digest Algorithm 5 (MD5) or Secure Hash Algorithm (SHA) protocols to encrypt the password.</p>
SNMPV3_AUTHPASSPHRASE	<p>Specifies the password encrypted using the opcpwcrpt utility.</p> <p>Note: For more information see, "Enhancing Security Parameters to Perform SNMPv3 GET" on the previous page.</p>
SNMPV3_ENCRYPTTYPE	<p>Specifies the protocols used to encrypt the Protocol Data Unit (PDU).</p> <p>You can either use Data Encryption Standard (DES) or Advanced Encryption Standard (AES) protocols to encrypt the PDU.</p>
SNMPV3_ENCRYPTPASSPHRASE	<p>Specifies the key used by DES and AES to encrypt the PDU.</p> <p>Note: Encryption key created by the administrator is encrypted with the opcpwcrpt utility.</p> <p>For more information, see "Enhancing Security Parameters to Perform SNMPv3 GET" on the previous page.</p>

Enabling **opcmona** to perform **SNMPv3Get**

You can enable **opcmona** to perform **SNMPv3Get** in one of the following scenarios:

- [When opctrapi is configured using the Node Info policy](#)
- [When opctrapi is not configured using the Node Info policy](#)

When opctrapi is Configured Using the Node Info policy

The Engine ID of a SNMP entity is available on the node if **opctrapi** is configured using the Node Info policy. For more information, see [Configuring opctrapi using the Node Info Policy](#). Add the Engine ID in the Measurement Threshold policy to enable **opcmona** to perform **SNMPv3Get**.

Note: You can run the `ovconfget eaagt` command to retrieve the Engine ID. You get the following output:

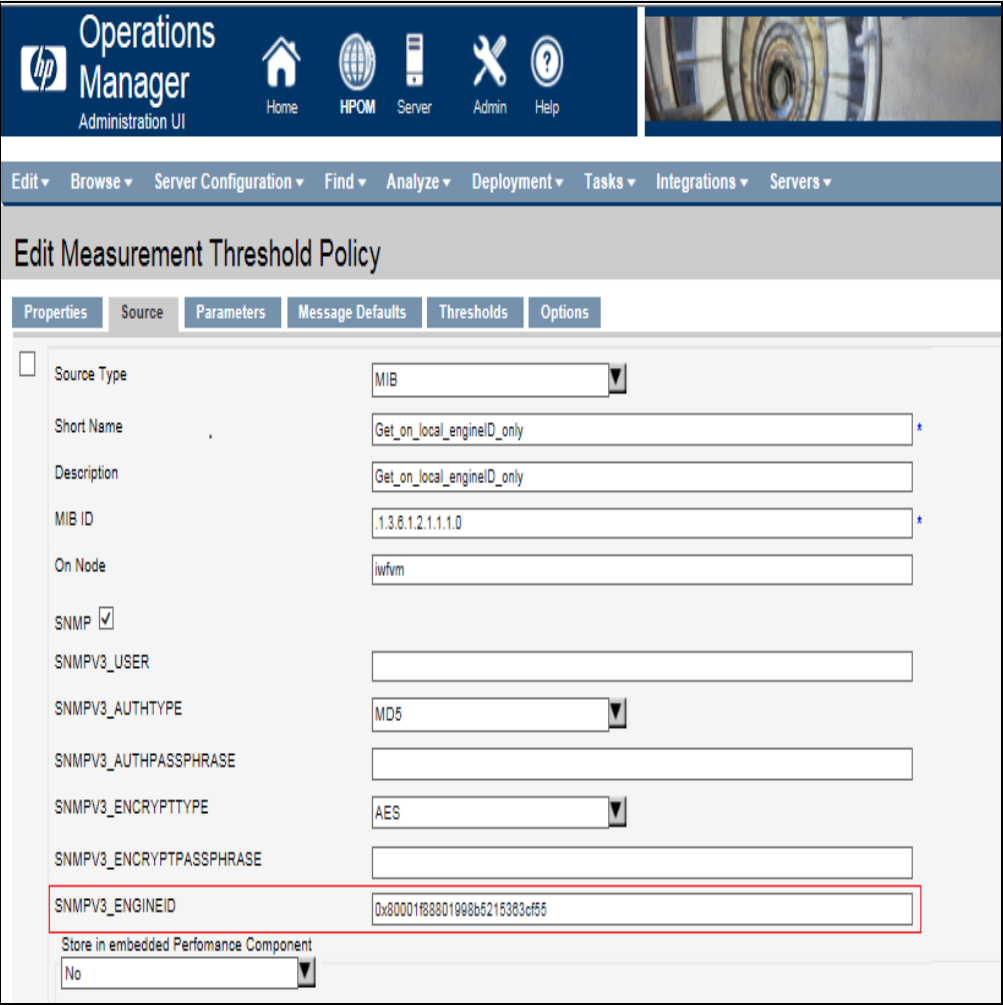
```
OPC_INSTALLATION_TIME=Wed Jan 7 22:42:20 IST 2015
OPC_NODENAME=<example_node.com>
SNMP_V3_
USERSDATA=demosnmpv3user:MD5:7*=*C61Ntd=@*Eb#@*E0##D:DES:7*=*C61Ntd=@*Eb#@*E0
##D:0x80001f88801998b5215363cf55
```

In this instance Engine ID is 0x80001f88801998b5215363cf55.

For example

Let us assume that the Engine ID that you have used to configure **opctrapi** is 0x80001f88801998b5215363cf55.

Add the Engine ID in the Measurement Threshold policy as shown:



After you add the Engine ID of a SNMP entity to the Measurement Threshold policy, **opcmona** is able to perform **SNMPv3Get** on OIDs present on that SNMP entity.

When opctrapi is Not Configured Using the Node Info policy

If **opctrapi** is not configured using the Node Info policy, you must add the following to the Measurement Threshold policy:

SNMPV3_USER <user name>

SNMPV3_AUTHTYPE <authentication method>

SNMPV3_AUTHPASSPHRASE <authentication password>

SNMPV3_ENCRYPTTYPE <encryption method>

SNMPV3_ENCRYPTPASSPHRASE <encryption key>

Note:

SNMPV3_AUTHPASSPHRASE and SNMPV3_ENCRYPTPASSPHRASE are encrypted automatically.

If encryption and authentication passwords are same, then you can leave the value of the SNMPV3_ENCRYPTPASSPHRASE variable blank.

For example:

The screenshot displays the 'Edit Measurement Threshold Policy' interface in the HP Operations Manager Administration UI. The 'Parameters' tab is selected, showing the following configuration details:

- Source Type:** MIB
- Short Name:** Get_on_remote_node_same_pass
- Description:** Get_on_remote_node_same_pass
- MIB ID:** .1.3.6.1.2.1.1.5.0
- On Node:** iwfvm01689.hpswlab.adapps.hp.com
- SNMP:**
- SNMPV3_USER:** testuser1
- SNMPV3_AUTHTYPE:** MD5
- SNMPV3_AUTHPASSPHRASE:** 7##_C81_*X_*#Aqe#*=AZ
- SNMPV3_ENCRYPTTYPE:** DES
- SNMPV3_ENCRYPTPASSPHRASE:** 7##_C81_*X_*#Aqe#*=AZ
- SNMPV3_ENGINEID:** (empty)
- Store in embedded Performance Component:** No

After you add the parameters of a SNMP entity in the Measurement Threshold policy, **opcmona** is able to perform SNMPv3Get on OIDs present on that SNMP entity.

Using the SourceEX API to Add Security Parameters to the Policy

A new interface, **SourceEx_SNMPV3** is introduced, to support multiple security parameters used for monitoring SNMPv3 MIBs. The SourceEx_SNMPV3 API enables you to use the Perl scripts to add security parameters to the Measurement Threshold policy.

Note: You cannot use the SourceEx_SNMPV3 APIs in vbscript of the Measurement Threshold policy.

Use the following syntax to add security parameters to the Measurement Threshold policy in the following scenarios:

1. When **opctrapi** is configured using the Node Info policy:

```
Policy-> SourceEx_SNMPV3 ("SNMP\\<object id>[\\<hostname>]", "<Engine ID>");
```

Note: When only the engine ID is specified, **opcmoma** will fetch other security parameters from the Node Info policy deployed for **opctrapi** to receive SNMPv3 traps. For more information, see [Configuring opctrapi using the Node Info Policy](#).

For example:

```
ADVMONITOR "TestMIBviaPerl_V3"
DESCRIPTION "<initial policy version>"
SCRIPTTYPE "Perl"
INSTANCEMODE SAME
MAXTHRESHOLD
SEPARATORS "          "
SEVERITY Unknown
EXTERNAL "SNMPPolicy"
DESCRIPTION ""
MSGCONDITIONS
DESCRIPTION "New threshold level"
CONDITION_ID "c5e0a2e4-a401-4186-876f-8b20b1ffcd8b"
CONDITION
THRESHOLD
SCRIPT "
use warnings;
my $MetricEx = $Policy->SourceEx_SNMPV3
("\\SNMP\\\\\\\\\\\\\\\\.1.3.6.1.2.1.1.1.0\\\\\\\\\\\\\\\\1x.18x.28.7x
\\", '0x800000001020304');
```

In this instance:

Parameter	Value
<object id>	\\1.3.6.1.2.1.1.1.0
<hostname>	\\1x.18x.28.7x
<Engine ID>	0x800000001020304

2. When **opctrapi** is not configured using the Node Info policy:

```
$Policy-> SourceEx_SNMPV3 ("SNMP\\<object id>[\\<hostname>]", "<user
name>","<authentication method>","<authentication password>","<encryption
method>","<encryption key>");
```

For example:

```

ADVMONITOR "TestMIBviaPerl_V3"
DESCRIPTION "<initial policy version>"
SCRIPTTYPE "Perl"
INSTANCEMODE SAME
MAXTHRESHOLD
SEPARATORS "          "
SEVERITY Unknown
EXTERNAL "SNMPPolicy"
DESCRIPTION ""
MSGCONDITIONS
DESCRIPTION "New threshold level"
CONDITION_ID "c5e0a2e4-a401-4186-876f-8b20b1ffcd8b"
CONDITION
THRESHOLD
SCRIPT "
use warnings;
my $MetricEx = $Policy->SourceEx_SNMPV3
("\SNMP\\\\\\\\\\\\.1.3.6.1.2.1.1.1.0\\\\\\\\\\\\\\\\1x.18x.28.7x\",
\"TestUser\", \"MD5\", '7*=*C61Ntd=@*Eb#@*E0##D', \"DES\", '7*=*C61Ntd=@*Eb#@*E0
##D');

```

In this instance:

Parameter	Value
<object id>	\\.1.3.6.1.2.1.1.1.0
<hostname>	\\1x.18x.28.7x
<user name>	TestUser
<authentication method>	MD5
<authentication password>	'7*=*C61Ntd=@*Eb#@*E0##D'
<encryption method>	DES
<encryption key>	'7*=*C61Ntd=@*Eb#@*E0##D'

Note: Ensure that passwords are enclosed within single quotation marks.

Configuring the Performance Collection Component Remotely

You can perform certain configuration tasks on the managed node remotely from the management server. Instead of performing the configuration tasks for the Performance Collection Component locally on every node, you can use a special set of policies and tools from the Operations Manager console to configure and work with the Performance Collection Component multiple nodes.

This feature is available only if you install the Operations Agent deployment package on the Operations Manager for Windows or Operations Manager on UNIX/Linux management servers.

Before You Begin

Before you begin configuring and controlling the Performance Collection Component remotely from the Operations Manager console, you must deploy the instrumentation files in the Operations Agent instrumentation group on the nodes where the agent is running.

To deploy the instrumentation from the Operations Manager for Windows console, follow these steps:

Note: If you monitor cluster nodes, make sure you deploy the instrumentation on all the nodes that constitute the cluster and not on the virtual node

1. In the console tree, right-click the node or the node group (where the agent is running), and then click **All Tasks > Deploy Instrumentation**. The Deploy Instrumentation dialog box opens.
2. Click Operations Agent, and then click **OK**. The deployment of the necessary instrumentation files begins on the nodes.

To deploy the instrumentation from Operations Manager on UNIX/Linux Console, follow these steps:

Note: If you monitor cluster nodes, make sure you deploy the instrumentation on all the nodes that constitute the cluster and not on the virtual node

1. Log on to the Administration UI.
2. Click **Deployment > Deploy Configuration**.

3. In the Distribution Parameters section, select **Instrumentation**, and then click **Please Select**. The **Selector** pop-up box opens.
4. In the **Selector** pop-up box, select the nodes where the agent program is running.
5. Select the **Force Update** option to overwrite the old instrumentation files.
Select this option on a node that was upgraded from an older version of the agent.
6. Click **Distribute**.

Deploy the OA-PerfCollComp-opcmsg Policy

The OA-PerfCollComp-opcmsg policy sends the alert messages to the OM message browser when the Performance Collection Component generates alarms. The policy is located in the **Operations Agent > Performance Collection Component > Message Interceptor** policy group. Before deploying other policies for the Performance Collection Component, deploy this policy on the nodes.

Note: If you monitor cluster nodes, make sure you deploy the policy on all the nodes that constitute the cluster and not on the virtual node.

Configuring the Performance Collection Component

The behavior of the Performance Collection Component of the Operations Agent depends on the configuration settings specified in the following files:

- Collection parameter file (**parm**)
- Alarm definition file (**alarmdef**)

See the *Performance Collection Component* section in the *Operations Agent Concepts Guide* for more information on the collection parameter and alarm definition files.

Configure the parm File

The **parm** file defines the data collection mechanism of the **oacore** collector. The Operations Agent deploys a **parm** file on every node, which is available in the following path:

On HP-UX, Solaris, AIX, and Linux: **/var/opt/perf**

On Windows: **%ovdatadir%**

You can modify the settings specified in the **parm** file to customize the data collection mechanism. However, if you manage a large number of nodes with the Operations Agent, it becomes difficult to modify every single copy of the **parm** file on every node.

With the help of the Operations Manager console, you can deploy the modified **parm** file on multiple node centrally from the management server.

From Operations Manager for Windows

The Operations Manager for Windows console provides you with ConfigFile policies which help you deploy any changes to the **parm** file across multiple nodes from the central management server. Different ConfigFile policies are available for different node operating systems.

To modify the collection mechanism by editing the **parm** file, follow these steps:


1. Identify the nodes where you want the modified collection mechanism to take effect.
2. In the console tree, click **Policy management > Policy groups > Operations Agent > Performance Collection Component > Collection configuration**. ConfigFile policies for configuring the **parm** file appear in the details pane.
3. Double-click the ConfigFile policy for the platform on which you want the modified collection mechanism to take effect (for example: **parm** file for HP-UX). The **parm** file for *<platform>* dialog box opens.
4. In the Data tab, modify the settings. See the *parm File Parameters* section in the *Operations Agent User Guide* for more details on configuration parameters in the **parm** file.
5. Click **Save and Close**. In the details pane, the version of the policy gets increased by .1.
6. Deploy the updated policy on the nodes of your choice.

Note: If you monitor cluster nodes, make sure you deploy the policy on all the nodes that constitute the cluster and not on the virtual node

From Operations Manager on UNIX/Linux 9.10

The Operations Manager on UNIX/Linux 9.10 console provides you with ConfigFile policies which help you deploy any changes to the **parm** file across multiple nodes from the central management server. Different ConfigFile policies are available for different node operating systems.

To modify the collection mechanism by editing the **parm** file from the Operations Manager for UNIX 9.10 console, follow these steps:

1. Identify the nodes where you want the modified collection mechanism to take effect.
2. In the console, click **Browse > All Policy Groups**. The list of all available policy groups appears on the page.
3. Click **Operations Agent**, click **Performance Collection Component**, and then click **Collection Configuration**. The list of available ConfigFile policies for the **parm** file appears.
4. Click the ConfigFile policy for the platform on which you want the modified collection mechanism to take effect. The Policy “OA_<platform>ParmPolicy” page appears.
5. Click  , and then click **Edit (Raw Mode)**. The Edit Config File policy... page appears.
6. In the Content tab, modify the settings

See the *parm File Parameters* section in the *Operations Agent User Guide* for more details on configuration parameters in the **parm** file.
7. Click **Save**.
8. Deploy the updated policy on the nodes of your choice.

Note: If you monitor cluster nodes, make sure you deploy the policy on all the nodes that constitute the cluster and not on the virtual node

Configure the alarmdef File

The alarm definition file (**alarmdef**) provides the performance subagent with the default specification for the alarm generation process. The Operations Agent deploys an **alarmdef** file on every node, which is available in the following path:

On HP-UX, Solaris, AIX, and Linux: `/var/opt/perf/`

On Windows: `%ovdatadir%`

You can modify the default settings in the **alarmdef** file to customize the alarm generation mechanism. You can use the Operations Manager console to centrally distribute the modified **alarmdef** file on multiple nodes.

From Operations Manager for Windows

The Operations Manager for Windows console provides you with ConfigFile policies which help you deploy any changes to the **alarmdef** file across multiple nodes from the central management server. Different ConfigFile policies are available for different node operating systems.

To modify the collection mechanism by editing the **alarmdef** file, follow these steps:

Identify the nodes where you want the modified collection mechanism to take effect.

1. In the console tree, click **Policy management > Policy groups > Operations Agent > Performance Collection Component > Alarm definition**. ConfigFile policies for configuring the **alarmdef** file appear in the details pane.
2. Double-click the ConfigFile policy for the platform on which you want the modified collection mechanism to take effect (for example: Alarmdef file for HP-UX). The Alarmdef file for *<platform>* dialog box opens.
3. In the Data tab, modify the settings. See the *alarmdef File Parameters* section in the *Operations Agent User Guide* for more details on configuration parameters in the **alarmdef** file.
4. Click **Save and Close**. In the details pane, the version of the policy gets increased by .1.
5. Deploy the updated policy on the nodes of your choice.


Note: If you monitor cluster nodes, make sure you deploy the policy on all the nodes that constitute the cluster and not on the virtual node

From Operations Manager on UNIX/Linux 9.10

The Operations Manager on UNIX/Linux 9.10 console provides you with ConfigFile policies which help you deploy any changes to the **alarmdef** file across multiple nodes from the central management server. Different ConfigFile policies are available for different node operating systems.

To modify the collection mechanism by editing the **alarmdef** file from the Operations Manager for UNIX 9.10 console, follow these steps:

1. Identify the nodes where you want the modified alert mechanism to take effect.
2. In the console, click **Browse > All Policy Groups**. The list of all available policy groups appears on the page.

3. Click **Operations Agent**, click **Performance Collection Component**, and then click **Alarm Definition**. The list of available ConfigFile policies for the **alarmdef** file appears.
4. Click the ConfigFile policy for the platform on which you want the modified collection mechanism to take effect. The Policy “OA_<platform>AlarmdefPolicy” page appears.
5. Click  , and then click **Edit (Raw Mode)**. The Edit Config File policy... page appears.
6. In the Content tab, modify the settings. See the *alarmdef File Parameters* section in the *Operations Agent User Guide* for more details on configuration parameters in the **alarmdef** file.
7. Click **Save**.
8. Deploy the updated policy on the nodes of your choice.

Note: If you monitor cluster nodes, make sure you deploy the policy on all the nodes that constitute the cluster and not on the virtual node

Remotely Working with the Operations Agent

You can use the Operations Manager console to start, stop, monitor, and view the details of the Operations Agent. From the Operations Manager console, you can use different tools to manage the operation of the Operations Agent. You must launch these tools on the nodes where the agent is deployed. The result of running a tool is displayed in the following section:

Operations Manager for Windows

Tool Output section in the Tool Status window

Operations Manager on UNIX/Linux

In the Application Output window in the Java GUI (Operations Manager for UNIX Operational UI)

You can use the following tools from the Operations Manager console:

Tool	Description
Start Agent	Enables you to start the Operations Agent on the managed node.
Stop Agent	Enables you to stop the Operations Agent on the managed node.
Restart Agent	Enables you to restart the Operations Agent on the managed node.
View Status	Enables you to view the status of the Operations Agent process, services, and daemons on the managed node.

View Version Information	Enables you to view the version of the Operations Agent on the managed node.
Refresh Alarm Service	Refreshes the Alarm service of the Performance Collection Component.
Scan Performance Component's Log Files	Scans the log files used by the scope collector on the node.
Check Performance Component's Parameter File Syntax	Helps you check the syntax of the parameter file in the managed node.
Check Performance Component's Alarmdef File Syntax	Helps you check the syntax of the alarmdef file in the managed node.
View status of post policy deploy action	<p>Helps you check the status of deployment of the parm or alarmdef policies on nodes. While launching this tool, make sure to specify either parm or alarmdef (as appropriate) as the tool parameter.</p> <p>You can set the tool parameter in the Parameter box in the Edit Parameters window when you use Operations Manager for Windows.</p> <p>When you use Operations Manager on UNIX/Linux, open the Edit Tool Status page for the tool, go to the OVO Tool tab, and then specify the tool parameter in the Parameters box</p>
Set Realtime Permanent License	Sets the permanent license for the HP Ops OS Inst to Realtime Inst LTU.
Set Glance Permanent License	Sets the permanent license for the Glance Software LTU.
Get License Status	Shows the status of LTUs on the node.

Configuring the SNMP Trap Interceptor

By default, the SNMP trap interceptor can collect SNMP traps originating from remote management stations or SNMP-enabled devices, and then can generate appropriate events based on the configuration.

Note: The SNMP trap interceptor (`opctrapi`) does not format the textual part of MIB. For example, the message text shows the MIB as `.1.3.6.1.4`, and not as `.iso.identified-organization.dod.internet.private`.

You can modify the default behavior of the SNMP trap interceptor by configuring the following properties:

- **SNMP_TRAP_PORT**: The default port is **162**. You can modify this value to any available port on the Operations Agent node.
- **SNMP_TRAP_FORWARD_DEST_LIST**: Use this property to set the address of the remote management station where you want to forward all the available SNMP traps. You can specify multiple system names (with port details) separated by commas.
- **SNMP_TRAP_FORWARD_ENABLE**: By default, this property is set to **FALSE**. By setting this property to **TRUE**, you can enable the SNMP trap interceptor to forward the SNMP traps available on the Operations Agent node to remote machines or management stations.
- **SNMP_TRAP_FORWARD_COMMUNITY**: Use this property to specify the community string of the source machines of the incoming traps and the target machine where you want to forward the SNMP traps. The community strings of the source machines must match with the community strings of the target machines.
- **SNMP_TRAP_FORWARD_FILTER**: Use this property to filter the available SNMP traps by their OIDs and forward only the select traps to the remote machine. The filtering mechanism takes effect with the wildcard (*) character. For example, if you set this property to **1.2.3.*.***, the SNMP trap interceptor will forward all the SNMP traps with the OIDs that begin with 1.2.3. By default, all the available traps are forwarded when you enable the SNMP trap interceptor to forward traps.

Note: If the community string of the source machines do not match with the community string of the target machines, the trap forwarding function fails.

To modify the default behavior of the SNMP trap interceptor, follow these steps:

1. Log on to the node with the necessary privileges.
2. In the command prompt, run the following commands:
 - To modify the port number, run the following command:

```
ovconfchg -ns eaagt -set SNMP_TRAP_PORT <port_number>
```

You must specify an integer value for *<port_number>*. Make sure the specified *<port_number>* is available for use.

- To enable the SNMP trap interceptor to forward SNMP traps to remote machines, run the following command:

```
ovconfchg -ns eaagt -set SNMP_TRAP_FORWARD_ENABLE TRUE
```

- If you enable the SNMP trap interceptor to forward SNMP traps to a remote machine, run the following command to specify the details of the target machine:

```
ovconfchg -ns eaagt -set SNMP_TRAP_FORWARD_DEST_LIST "<machinename>:<port>"
```

<machinename> is the fully-qualified domain name of the machine where you want to forward the SNMP traps and <port> is the HTTPS port of the machine. If you want to specify multiple targets, separate the machine details with commas.

- If you want to forward only selected SNMP traps available on the node to the remote machine, run the following command:

```
ovconfchg -ns eaagt -set SNMP_TRAP_FORWARD_FILTER "<OID Filter>"
```

<OID Filter> is an OID with the wildcard characters. The SNMP trap interceptor filters the traps that match the specified OID (with the wildcard characters) from the available traps, and then forwards them to the target machine.

Configuring the SNMP Trap Interceptor for SNMPv3

Traps

In a networked environment, it is important to ensure secure communication between the trap-sending device and the Operations Agent. The Simple Network Management Protocol version 3 (SNMPv3) provides secure access to devices that send traps by authenticating users and encrypting data packets sent across the network.

In earlier versions of the Operations Agent, the **opctrapi** process was configured to intercept SNMPv1 and SNMPv2 traps. With the Operations Agent 12.05, **opctrapi** can also intercept both SNMPv3 *trap* and *inform* messages.

Configure the following variables to enable **opctrapi** to intercept SNMPv3 traps:

Note: The **opctrapi** process intercepts SNMPv3 traps only in the NETSNMP mode.

- **SNMP_V3_ENABLE** - You can set this configuration variable in the eaagt namespace. The default value of this variable is *TRUE*.

To disable **opctrapi** from intercepting SNMPv3 traps, set **SNMP_V3_ENABLE** to *FALSE*. If the variable is set to *FALSE*, **opctrapi** intercepts only SNMPv1 and SNMPv2 traps.

- **SNMP_V3_USERSDATA** - This variable is mandatory and must be set in the eaagt namespace. Use the variable to configure users.

```
SNMP_V3_USERSDATA=<Parameters for User 1>;<Parameters for User  
2>;<Parameters for User n>
```

In this instance, <Parameters for User> includes the following:

```
<user name>:<authentication method>:<authentication password>:<encryption method>:<encryption key>:<Engine ID>
```

Parameter	Description
<user name>	Specifies the SNMPv3 user name created by the administrator.
<authentication method>	Specifies the protocols used to encrypt the password. You can either use Message Digest Algorithm 5 (MD5) or Secure Hash Algorithm (SHA) protocols to encrypt the password.
<authentication password>	Specifies the password encrypted using the opcpwcrpt utility. Note: For more information, see Encrypting Password with the opcpwcrpt Utility .
<encryption method>	Specifies the protocols used to encrypt the Protocol Data Unit (PDU). You can either use Data Encryption Standard (DES) or Advanced Encryption Standard (AES) protocols to encrypt the PDU.
<encryption key>	Specifies the key used by DES and AES to encrypt the PDU. Note: Encryption key created by the administrator is encrypted with the opcpwcrpt utility. For more information, see Encrypting Password with the opcpwcrpt Utility .
<Engine ID>	Specifies the SNMPEngineID that is used to uniquely identify SNMP entities.

Configuring opctrapi to Intercept SNMPv3 Traps

You can use one of the following methods to configure opctrapi to intercept SNMPv3 traps:

- [Using Node Info Policy](#)
- [Using the XPL variables](#)

Using the Node Info Policy

Follow these steps:

1. Log on to the OM server.
2. Add the following content to the Node Info Policy:

```
;XPL config
[eaagt]
SNMP_V3_ENABLE=TRUE
SNMP_V3_USERSDATA=<Parameters for User 1>;<Parameters for User 2>;<Parameters
for User n>
```

3. Deploy the Node Info Policy on the nodes. `opctrapi` is restarted automatically.

Following is an example of a snippet from the Node Info Policy to configuring **opctrapi** to intercept SNMPv3 traps:

```
;XPL config
[eaagt]
SNMP_V3_ENABLE=TRUE
SNMP_V3_
USERSDATA=user1:SHA:7###C61###X.###AqV###A,w##I:DES:7###C61###X.###AqV###A,w##I:0
x800000001020304;user2:SHA:7###C61###X.###AqV###A,w##I:DES:7###C61###X.###AqV###
A,w##I;
```

Note: The Node Info Policy type provides a way to modify configuration information on a managed node where the Operations Agent is running.

Using the XPL variables

Follow these steps:

1. Log on to the Operations Agent node.
2. To enable SNMPv3 trap interception, run the following command:

```
ovconfchg -ns eaagt -set SNMP_V3_ENABLE TRUE
```

3. To set users, run the following command:

```
ovconfchg -ns eaagt -set SNMP_V3_USERSDATA <Parameters for User 1>; <Parameters for
User 2>;<Parameters for User n>
```

opctrapi is restarted automatically.

For example:

```
ovconfchg -ns eaagt -set SNMP_V3_ENABLE TRUE
```

```
ovconfchg -ns eaagt -set SNMP_V3_USERSDATA
"snmpv3User1:SHA:7*=@*C61Ntd=@*Eb#@*E0##D:DES:7*=@*C61Ntd=@*Eb#@*E0##D:0x800000
00001020304;snmpv3User2:SHA:8*=@*D61Ntd=@*Eb#@*E0##D:DES:8*=@*D61Ntd=@*Eb#@*E0
##D:0x8000000001030404;"
```

In this instance:

Parameter	Value	Value
<user name>	snmpv3User1	snmpv3User2
<authentication method>	SHA	SHA
<authentication password>	7*=@*C61Ntd=@*Eb#@*E0##D	8*=@*D61Ntd=@*Eb#@*E0##D
<encryption method>	DES	DES
<encryption key>	7*=@*C61Ntd=@*Eb#@*E0##D	8*=@*D61Ntd=@*Eb#@*E0##D
<Engine ID>	0x8000000001020304	0x8000000001030404

Note: Each parameter is separated from the next by a colon (:), and each user is separated from the next by a semicolon (;).

Encrypting Password with the **opcpwcrpt** Utility

1. Log on to the OM server with administrator privileges.
2. Open a command prompt.
3. Go to the following location:

On Windows:

```
"%OvBinDir%\OpC\install"
```

On Linux:

```
/opt/OV/bin/OpC/install
```

4. Run the following command to encrypt your password:

```
# opcpwcrpt <your password>
```

The output string is the encrypted password. Use the encrypted password appropriately in place of <authentication password> or <encryption key>.

Enhancing SNMP Trap Interceptor to Intercept Traps Based on the Object ID of the Varbind

The SNMP Trap Interceptor **opctrapi**, is enhanced to intercept SNMP traps based on the object ID of the varbinds (OID) along with the position.

To enable **opctrapi** to intercept SNMP traps based on the object ID of the varbind, you can either create or modify a SNMP trap interceptor policy and add (or modify) the following as shown in the example:

<keyword><object id><separator><pattern string>

For example:

Type	Description
+ Message On Matched Condition	
Condition	Message
Actions	Custom Attributes
Correlation	Instruction
Node	
Enterprise OID	
Generic Trap	
Specific Trap	
\$1	#oid#.1.3.6.1.2.1.2.2.1.4.<_><<#> -eq 8192>
\$2	#oid#.1.3.6.1.2.1.2.2.1.1.<_><<#> -lt 2>
\$3	24
\$4	#oid#.1.3.6.1.2.1.2.2.1.1.<_><<#> -le 1>
\$5	#oid#.1.3.6.1.2.1.2.2.1.1.<_><<#> -gt 0>
\$6	#oid#.1.3.6.1.2.1.2.2.1.1.<_><<1#var1> -ge 1>
\$7	#oid#.1.3.6.1.2.1.2.2.1.2.1.<_><lo<<1#> -le 2>
\$8	<<#var8> -ge 100>
\$9	#oid#.1.3.6.1.2.1.2.2.1.2.1.<_><@.var2><#var3>
\$10	#oid#.1.3.6.1.2.1.2.2.1.2.1.<_><[lo]var4>
\$11	

In the instance marked in the screenshot:

<keyword> is always "#oid#"

<object id> is .1.3.6.1.2.1.2.2.1.4.1

<separator> is <_>

<pattern string> is <<#> -eq 8192>

Value is 8192

While matching the varbind, **opctrapi** checks if the field starts with the keyword **#oid#**. If the field starts with the keyword **#oid#**, varbind OID in the trap is compared against the pattern string mentioned in the field and the condition is verified.

Note: In the SNMP Trap Interceptor policy, you can also use the OID in:

- Message attributes (Message Text, Message group, Service name, Message object, Message key, Object, Application)
- Custom Message Attributes (CMA)

In the following example OID is used in the Message Text:

The screenshot shows a configuration window for an SNMP trap policy. The 'Type' is set to '+ Message On Matched Condition' and the 'Description' is 'Enterprise trap - with variab'. The 'Message' tab is selected, showing various fields for configuration. The 'Object' field contains '<#oid#.1.3.6.1.2.1.2.2.1.14.1>'. The 'Message Text' field contains 'Value of .1.3.6.1.2.1.2.2.1.14.1 is <#oid# 1.3.6.1.2.1.2.2.1.14.1>,' and this text is highlighted with a red rectangular box. Other fields include Severity (Normal), Node, Application, Message Group, Service ID, Service hosted on, and Message type (SNMP_enterprise).

Enabling opctrapi to Use NETSNMP Logging

The **opctrapi** process logs the trace messages for SNMPv3 traps in the `<%ovdatadir%/tmp/opc/trace` file. These messages do not provide sufficient information required to troubleshoot problems related to security parameters associated with the incoming SNMPv3 traps.

To resolve this issue, the **opctrapi** process is enhanced to use NETSNMP logging. You must set the following trace variables in the `eaagt` namespace to enable **opctrapi** to use NETSNMP logging:

- `OPC_TRACE` - By default, this variable is set to `FALSE`. Set this variable to `TRUE` to enable tracing.

- `OPC_TRC_PROCS` - This variable lists the names of all processes that must be traced. The process names must be separated by commas.
- `OPC_DBG_PROCS` - This variable lists the names of all traced processes that you must debug. The process names must be separated by commas.

For example:

```
OPC_TRACE=TRUE
OPC_TRC_PROCS=opctrapi
OPC_DBG_PROCS=opctrapi
```

After you set these variables; a new log file `netsnmp.log` is created at `<%datadir%/tmp/OpC`. The `netsnmp.log` file contains comprehensive information about errors and incoming traps that can be used to troubleshoot problems.

For example:

You can check the following trace conditions in the `netsnmp.log` file:

- Unknown user error - This error is logged if a SNMPv3 user is not configured and a SNMPv3 trap is received.
- Authentication failure error - This error is logged if SNMPv3 password or Authentication method is incorrect.
- Decryption error - This error is logged if SNMPv3 encryption method or key is incorrect.

Note: The `netsnmp.log` file is not rolled over at regular intervals. Trace messages are added to the same file even if you re-enable tracing.

Integrating Operations Agent with NNMi

The Network Node Manager i software (NNMi) is a network management software that uses existing SNMP agents for network element discovery and status. Integration with Operations Agent helps you to monitor the traps forwarded by NNMi and view the enriched SNMP traps on the OM console.

The Operations Agent integration with NNMi is available with the following:

- Northbound interface - NNMi northbound interface is used for forwarding NNMi incidents to any application that can receive SNMPv2c traps. SNMP events are sent to SNMP trap interceptor (**opctrapi**) from NNMi using the northbound interface. To ensure the correct association between the trap-sending device and the event in the receiving application, the rules for these traps must be customized by the varbinds. Integration is done by using the policy which contains the rules for **opctrapi** to intercept traps forwarded by NNMi northbound interface. **opctrapi** will also set various parameters of the outgoing message based on the various configuration such as severity, node name and so on defined in the policy. For more information about using Northbound interface, see *NNMi Deployment Reference, version 9.22*.
- Event forwarding - NNMi SNMP trap forwarding mechanism enriches each SNMP trap before forwarding it to the trap destination. There are two types of event forwarding mechanisms:
 - Original trap forwarding - No additional varbinds are added.
 - Default trap forwarding - NNMi adds varbinds to identify the original source object. This mechanism is used by **opctrapi** to enhance the SNMP traps.

For more information about the default trap forwarding, see *Configuring Trap Forwarding in the NNMi Online Help*.

NNMi forwards the incidents either by NNMi event forwarding or Northbound interface and adds the varbinds to the original event. These varbinds contain additional trap information. To process and use this information with the Operations Agent, you must configure the SNMP interceptor (**opctrapi**) for the following enhancements:

- Associate the severity that is available with SNMP trap to the OM message. This works with the Northbound interface only. For more information, see [Integrating with NNMi Northbound Interface to Associate the Message Severity](#).
- Create OM Custom Message Attributes (CMA) from NNMi incidents- This works with the Northbound interface only. For more information, see [Integrating with NNMi Northbound Interface to Add CMA on the OM Console](#).
- Derive the source node name where the SNMP trap originated. This works with the default trap forwarding only. For more information, see [Integrating with NNMi Trap Forwarding Interface to Assign Source of SNMP Trap](#).

Integrating with NNMi Northbound Interface to Associate the Message Severity

Operations Agent uses the policy that is generated by the NNMi tool `nmopcxport.ovp1`. This tool sets the severity of the SNMP traps. If the severity generated by NNMi tool is Normal, Critical, Warning, Major, or Minor then the message severity would appear as Normal, Critical, Warning, Major, or Minor.

Only when the tool indicates the severity of the message generated by the OM policy as **Unknown**, configure **opctrapi** to use the available varbind in the trap to derive the value of severity. This configuration helps to associate the severity available with the SNMP trap to OM message severity.

Configuring SNMP Trap Interceptor to Enhance Message Severity

To set the message severity based on severity level available in SNMP trap, you can configure SNMP trap interceptor (**opctrapi**). Read the severity from the specific OID of SNMP trap and then associate the same severity level to the OM messages. This configuration step helps `opctrapi` to use the severity level available in SNMP traps.

Note: The configuration is applicable for the rules where the message severity set in the OM is Unknown.

You can modify the default behavior of **opctrapi** by configuring the following properties:

- `OPC_SNMP_SET_SEVERITY`: By default, this property is set to **FALSE**. By setting this property to **TRUE**, you can enable the SNMP trap interceptor to read the SNMP traps with the specific varbind OID (**.1.3.6.1.4.1.11.2.17.19.2.2.12**) and set the severity of the message. If this default OID value is not available in the SNMP trap, the message severity remains as Unknown.
- `OPC_SNMP_OVERRIDE_SEVERITY_OID`: You can set the new OID value. The new value will specify the severity of the message.

To configure the SNMP interceptor, follow these steps:

1. Log on to the node with the necessary privileges.
2. In the command prompt,

- To enable the SNMP interceptor to read severity levels defined in SNMP traps, run the following command:

```
ovconfchg -ns eaagt.integration.nnm -set OPC_SNMP_SET_SEVERITY TRUE
```

- To set the new OID value, run the following command:

```
ovconfchg -ns eaagt.integration.nnm -set OPC_SNMP_OVERRIDE_SEVERITY_OID<OID>
```

In the above command, <OID> is the object identifier. Here, OID is used to derive the severity level of SNMP traps.

By default, the severity level is derived from the default OID “.1.3.6.1.4.1.11.2.17.19.2.2.12”.

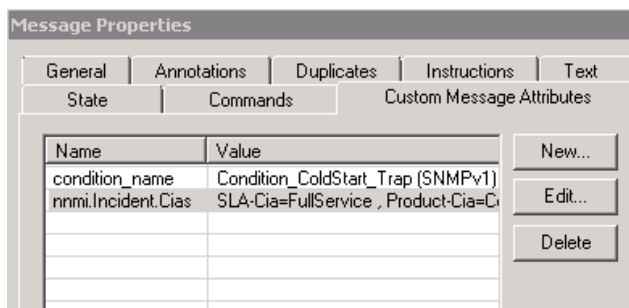
You can change the default OID by running the following command:

```
ovconfchg -ns eaagt.integration.nnm -set OPC_SNMP_OVERRIDE_SEVERITY_OID<.1.3.6.1.4.1.11.2.17.19.2.2.22>
```

Henceforth, the severity of the message will be based on new OID value <.1.3.6.1.4.1.11.2.17.19.2.2.22>.

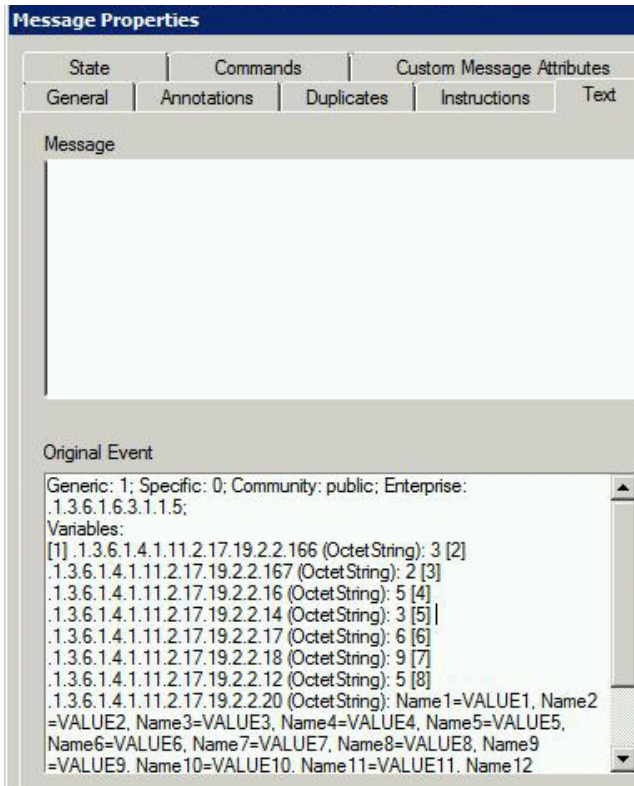
Integrating NNMi Northbound Interface to Add CMA on the OM Console

When a SNMP trap is forwarded in NNMi’s north-bound integration, the trap includes a varbind that represents the Custom Incident Attributes (CIAs). NNMi CIAs appear in a single custom message attribute (CMA) on the OM console if CMAs¹ are configured in the OM policy. The list appears with CMA name and value pair. This is the default behavior. The messages in CMA tab appear as:



If CMAs are not configured in OM policy, NNMi CIA message appears as a text message:

¹A custom message attribute (CMA) can be any information that is meaningful to you and you can have more than one CMA attached to a single message.



After you configure **opctrapi**, the SNMP trap interceptor should access the CIAs and represent each CIA as a Custom Message Attribute (CMA) in the OM message.

Configure SNMP Trap Interceptor to Create CMAs from NNMi CIA on the OM Console

You can configure **opctrapi** and display NNMi CIA as CMA name and value pair in the CMA properties tab by configuring the following:

- **OPC_SPLIT_NNM_CUSTOM_ATTR**: By default, this property is set to **FALSE**. By setting this property to **TRUE**, all NNMi CIAs values present in the varbind (**.1.3.6.1.4.1.11.2.17.19.2.2.20**) will appear as individual CMAs.
- **OPC_SPLIT_NNM_CUSTOM_ATTR_MAX**: This variable is **optional**. This property is enabled only if the **OPC_SPLIT_NNM_CUSTOM_ATTR** variable is set to **TRUE**. The variable defines the number of NNMi custom attributes that can be read and interpreted by OM message. By default, the value is set to 20. This means that only 20 CMAs will appear separately with respective values and rest of the NNMi CIA will appear in a single CMA. You can specify the value as required.

Follow the steps to modify the default behavior of the **opctrapi** to create separate CMAs from the NNMi CIAs:

1. Log on to the node with the necessary privileges.
2. In the command prompt,
 - To enable **opctrapi** to read and create a separate CMA from the list that appears in varbind (**.1.3.6.1.4.1.11.2.17.19.2.2.20**), run the following command:

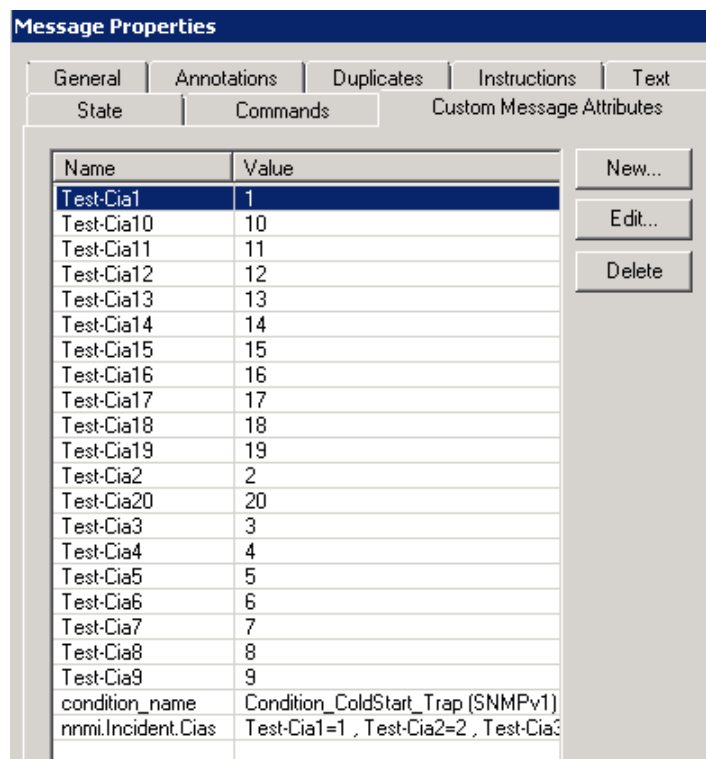
```
ovconfchg -ns eaagt.integration.nnm -set OPC_SPLIT_NNM_CUSTOM_ATTR TRUE
```

- To set the value of the variable to read the message and create separate CMA name and value pair, run the following command:

```
ovconfchg -ns eaagt.integration.nnm -set OPC_SPLIT_NNM_CUSTOM_ATTR_MAX<Value>
```

In the above command, *<Value>* is an integer. By default, the value is 20.

After configuration, the messages in CMA tab appear with name and value attributes. In addition, the NNMi CIA's untruncated message `nnmi.Incident.Cias` also appears in the list.



Integrating with NNMi Trap Forwarding Interface to Assign Source of SNMP Trap

In the previous versions, whenever NNMi forwarded the SNMP v2c traps to the node (Operations agent is available on the node), the source node name in the OM message will appear as NNMi node name.

Operations Agent is integrated with the NNMi forwarding interface. The NNMi forwarding interface adds varbinds to identify the source where the trap was originated.

You can configure **opctrapi** to derive the source node that generated the trap. When an OM message is generated, the SNMP trap interceptor (**opctrapi**) should set the source node name as the node where the trap was generated and not NNMi's node name.

Configure SNMP Trap Interceptor to Derive the Source Node Name

You can configure the following property:

OPC_NODENAME_FROM_NNM_FRWD_TRAP:

By default, this property is set to **FALSE**. By setting this property to **TRUE**, all the traps are searched for the varbinds “.1.3.6.1.4.1.11.2.17.2.19.1.1.2.0” and .1.3.6.1.4.1.11.2.17.2.19.1.1.3.0. **Opctrapi** uses the following varbinds:

- (.1.3.6.1.4.1.11.2.17.2.19.1.1.2.0) - To identify the IP address type.
- (.1.3.6.1.4.1.11.2.17.2.19.1.1.3.0) - To derive the IP address to set the node as the source node.

To configure **opctrapi** to assign the source node name when NNMi forwards SNMPv2 events, follow these steps:

1. Log on to the node with the necessary privileges.
2. In the command prompt, run the following command to enable **opctrapi** to read the varbinds and assign the node from where the trap is originating:

```
ovconfchg -ns eaagt.integration.nnm -set OPC_NODENAME_FROM_NNM_FRWD_TRAP TRUE
```

Note: If the variable is not set or set to False, **opctrapi** will not derive the original source (node name) from where the trap was generated.

Configuring the Message Storm Suppression

Message storm occurs when an unusually high number of new messages arrive on the management server within a short time interval and flood the active message browser. This phenomenon can lead to management server outages.

The Operations Agent can detect and suppress the message storm on a managed node. Configure the following variables:

- **OPC_MSG_STORM_DETECTION** - This variable is **mandatory**. By default, this property is set to **FALSE**. By setting this property to **TRUE**, you can enable the message storm detection.
- **OPC_MSG_STORM_DETECTION_CATEGORY** - This variable is **mandatory**. This property is enabled only if the **OPC_MSG_STORM_DETECTION** variable is set to **TRUE**. You can set the variable for any one of the message attributes such as **POLICY**, **MSGGROUP**, **APPLICATION**, **OBJECT** or **SEVERITY**.
- **OPC_MSG_STORM_RATE** - This variable is **mandatory**. The variable defines the following parameters:
 - **Threshold** - Defines a limit of incoming messages. When the count of incoming messages exceeds the defined limit, message storm condition is detected.
 - **Time** - Interval for which the incoming messages are counted to detect the message storm condition.
 - **Reset** - Defines a limit when the number of messages are below the defined value. This parameter is used to detect when the storm condition is resolved.
- **OPC_SEND_INTERNAL_MSG_ON_MSGSTORM** - This variable is **optional**. Defines whether to send or stop the internal messages. By default, the value is set to **TRUE**.
- **OPC_SUPPRESS_MSG_ON_MSG_STORM** - This variable is **optional**. Defines whether to send or suppress the messages. The default behavior is that if the threshold condition is met and the message storm state is detected, all the messages beyond the threshold value will be suppressed. By default, the value is **TRUE**.
- **OPC_MSG_STORM_TRACE_SUPPRESSED_MSGS** - This variable is **optional**. Defines whether to log the messages to the log file only when the `<OPC_SUPPRESS_MSG_ON_MSG_STORM >` is set to **TRUE**. By default, the value is **FALSE**.

When message agent detects a message storm or when the message storm is resolved, a message is logged into the log file (*System.txt*):

On Windows:

%OvDataDir%\log

On HP-UX/Linux/Solaris:

/var/opt/OV/log

These parameters are available in the `eaagt.msgstorm` namespace.

The advantages of the detection of the message storm on the managed node are:

- No ECS circuits are required
- Message storm is identified at the source node
- Easy configuration steps and can be configured for various message attributes

Note: Example to detect and suppress the message storm on the managed node.

You can detect the storm condition by setting the `OPC_MSG_STORM_DETECTION` parameter as `TRUE`.

After enabling the message storm condition, define the following parameters -

`OPC_MSG_STORM_DETECTION_CATEGORY` as `POLICY`. Policies deployed are `Opclepolicy` and `Opcmsgipolicy`.

To set the `OPC_MSG_STORM_RATE` parameter, you can calculate the incoming message rate to set the parameters. See ["Configuring the Message Storm Suppression" on the previous page](#).

You can set the values as per the message rate.

Set the values of the `OPC_MSG_STORM_RATE` parameter as `Threshold = 100`, `Time = 20` and `Reset Value = 50`

The `Opclepolicy` sends 50 messages and `Opcmsgipolicy` sends 101 messages. The storm is detected as the messages from `Opclepolicy` are 101 compared to the threshold value (100).

You can get the notification when the message storm is detected and resolved by the parameter `OPC_SEND_INTERNAL_MSG_ON_MSGSTORM`. By default, the value is set to `TRUE`.

By default, the message count beyond 100 will be suppressed or ignored. You can set the parameter (`OPC_SUPPRESS_MSG_ON_MSG_STORM`) to `FALSE` to get the messages that are getting suppressed.

You can log the suppressed messages in the log file by setting the parameter (`OPC_MSG_STORM_TRACE_SUPPRESSED_MSGS`).

Note: The maximum size limit of the object used in `opcmon` command must not exceed 50 characters. If the object size exceeds 50 characters, the subsequent characters after the 50th

character is truncated.

Configuring Message Storm Detection and Suppression

Note: Make sure that you restart the message agent whenever you change any parameter for the configuration. If the message agent is restarted, the content associated with message storm detection is reset.

To configure the message storm detection or suppression, follow these steps:

1. Log on to the node with the necessary privileges or you can configure remotely.
2. To enable the message storm detection, run the following command:

```
ovconfchg -ns eaagt.msgstorm -set OPC_MSG_STORM_DETECTION TRUE
```

The default value is set to FALSE.

3. To set the category of the message, run the following command.

```
ovconfchg -ns eaagt.msgstorm -set OPC_MSG_STORM_DETECTION_CATEGORY<CATEGORY>
```

Note: This is a mandatory variable.

The category must be defined as **one** of the following:

- POLICY
- GROUP
- APPLICATION
- OBJECT
- SEVERITY

Note: You can only define one of the available options. The combination of various values is not available.

4. Run the command to set the variables of the message rate:

```
ovconfchg -ns eaagt.msgstorm -set OPC_MSG_STORM_RATE<ThresholdValue:Time:ResetValue>
```

Note: These variables are mandatory.

The variables are defined as follows:

- **Threshold Value** - Set a numeric value. If the number of messages in the set time interval are more than the threshold value, this condition is known as message storm condition. In this condition, the message agent suppresses the messages till the reset condition is met. The threshold value is checked for the message storm state for all the groups available in the set category. You can check the message rate to set this parameter. See [Checking the Message Rate](#).

Example: The category is set to SEVERITY. In this message severity category, there are five types of groups such as Critical, Major, Minor, Warning, and Normal. The messages are grouped based on the severity and are checked for the configured threshold value.

Set the threshold value as 100. The messages received for Critical, Major, Minor, Warning, and Normal severity are 80, 60, 40, 50, and 110 respectively. Here the message storm is detected only for the normal state as the number of messages (110) are more than the set threshold value (100). The remaining 10 messages of the normal severity will be suppressed.

- **Time** - Set the time interval in seconds. The time interval in which the count of incoming messages are recorded. The recommended value is below 900 seconds.
- **Reset Value** - Set a numeric value that must be less than or equal to the threshold value. Message storm detection gets reset if the following conditions are met:
 - Message rate is checked for the set time interval.
 - Message rate should be less than the reset value.

Example to set the parameters for message storm condition.

```
set OPC_MSG_STORM_DETECTION=TRUE
set OPC_MSG_STORM_DETECTION_CATEGORY=SEVERITY
set OPC_MSG_STORM_RATE=100:60:45
```

where Threshold value is 100, Time is 60, and Reset value is 45.

After the message storm detection, the number of incoming messages are checked for the set periodic interval (60 seconds). In the set time interval, if the messages are less than the reset value (45), the message agent (opcmsga) stops ignoring the messages and the message storm state is over.

Note: Step 5, 6 and 7 are optional steps.

5. To receive and stop the internal messages, run the following command:

```
ovconfchg -ns eaagt.msgstorm -set OPC_SEND_INTERNAL_MSG_ON_MSGSTORM<Value>
```

The *<Value>* must be defined as *one* of the following:

- TRUE- Internal messages are generated whenever the message storm state is detected and whenever the state is resolved. The default value is TRUE.
- FALSE- No internal messages are generated for the message storm state and whenever the state is resolved.

6. In the message storm condition, run the following command to suppress or receive the messages:

```
ovconfchg -ns eaagt.msgstorm -set OPC_SUPPRESS_MSG_ON_MSG_STORM<Value>
```

The value must be defined as *one* of the following:

- TRUE- Message agent will suppress messages in the message storm condition. The default value is TRUE.
- FALSE- Message agent will not suppress the messages in the message storm condition.

7. **Skip this step** if OPC_SUPPRESS_MSG_ON_MSG_STORM variable is set to FALSE.

In the message storm condition, run the following command to log the suppressed messages:

```
ovconfchg -ns eaagt.msgstorm -set OPC_MSG_STORM_TRACE_SUPPRESSED_MSGS <Value>
```

The value must be defined as *one* of the following:

- TRUE- Message agent will save all the suppressed messages in the log file.
- FALSE- Message agent will not save the suppressed messages. By default, the value is FALSE.

The suppressed messages are available in (*msgsuppress.log <process id of the process>*) in the following directory:

On Windows:

```
%OvDataDir%\tmp\OpC
```

On HP-UX/Linux/Solaris:

```
/var/opt/OV/tmp/OpC
```

8. Run the following command to restart the message agent:

```
ovc -restart opcmsga
```

Checking the Message Rate

Message rate is used to measure the average rate of messages through the system. After checking the average message rate, you can configure an ideal threshold limit and reset limit to detect and suppress the message storm condition. You can only check the message rate after configuring the message storm detection.

To check the message rate at a particular time, run the following command:

On Windows:

```
%O\InstallDir%\lbin\eaagt\opcmsga -message_rate
```

On HP-UX/Linux/Solaris:

```
/opt/OV/lbin/eaagt/opcmsga -message_rate
```

On AIX:

```
/usr/lpp/OV/lbin/eaagt/opcmsga -message_rate
```

Configuring the Backup Server

The message agent sends the messages to the primary server, which is your Operations Manager (OM). If you have more than one OM server in your environment, you can also configure another server to act as the backup server in your environment. When you have a backup server configured in your environment, if the communication link to the primary server is down and the messages cannot be sent to the primary server, the message agent sends the messages to the backup server(s).

You can also configure the backup server for load distribution of the message sync between both the primary and backup servers (for example, in the manager-of-manager (MoM) scenario). For more information on the MoM scenario, see *Operations Manager for UNIX Concepts Guide* or *Operations Manager Online Help*. A backup server can be another OM server in your environment. You can configure one or more servers as the backup server(s).

You can enable the Operations Agent to send messages to the backup server by configuring values for the following variables:

- **OPC_BACKUP_MGRS** - You can specify one or a list of servers to be configured as the backup servers. The values in the list can be comma or semicolon separated.
- **OPC_BACKUP_MGRS_FAILOVER_ONLY** - When you set the value for this variable to **TRUE**,

the message agent forwards the messages to the backup servers only if the primary server is down. And when the value is set to **FALSE**, messages are sent to the backup servers, irrespective of the status of the primary server. The default value for this variable is **FALSE**.

If you have configured the list of backup servers, during initialization, the message agent creates a list of the backup servers and then checks for the value set for the `OPC_BACKUP_MGRS_FAILOVER_ONLY` variable. When a message arrives at the `msgagt.dcf` file and the value for the `OPC_BACKUP_MGRS_FAILOVER_ONLY` variable is:

- **FALSE**- The message agent sends the message to the primary server and then to the backup server. After the message is successfully sent to both the servers, the message entries are removed from the `msgagt.dcf` file.
- **TRUE**- The message agent sends the messages to the primary server. If the message sending fails, message agent tries to forward the message to the backup servers. If there are more than one backup servers listed, and the message is delivered to at least one of the backup server, then the message entry is removed from the `msgagt.dcf` file. Message agent does not try to resend the message to the other backup servers and also does not send the message to the primary server, when it is up again.

Note: When `OPC_BACKUP_MGRS_FAILOVER_ONLY` is set to **TRUE**, if a message that started a local automatic action is sent to the primary server, and then the primary server goes down, the message agent sends the action response to backup manager. But this is discarded by the backup manager as the backup server does not have the original message that started the action.

Prerequisites:

- Trusted certificates of the backup servers must be installed on the Operations Agent node.
- Trusted certificate of the primary server must be installed on the backup server.
- The Operations Agent node must be added to the backup servers.

To set the values for the variables, follow these steps:

1. Log on to the node as root or administrator.
2. Run the following commands:
 - To set the backup server, run the following command:

```
ovconfchg -ns eaagt -set OPC_BACKUP_MGRS <servername>
```

<servername> is the name of the backup server. For example, *abc.xyz.example.com*.

To configure a list of backup servers, use either of the following commands:

- `ovconfchg -ns eaagt -set OPC_BACKUP_MGRS <servername1>,<servername2>,...,<servernameN>`
- `ovconfchg -ns eaagt -set OPC_BACKUP_MGRS <servername1>;<servername2>;...;<servernameN>`

The sequence of the backup servers depend on the sequence in which you specify the server names. In the preceding examples, <servername1> acts as the first backup server and <servername2> as the second backup server.

- To modify the value for the OPC_BACKUP_MGRS_FAILOVER_ONLY variable, run the following command:

```
ovconfchg -ns eaagt -set OPC_BACKUP_MGRS_FAILOVER_ONLY TRUE
```

The default value is **FALSE**.

Configuring the RTMA Component

The Real-Time Metric Access (RTMA) component provides you with real-time access to system performance metrics, locally or remotely. The **perfd** process, which is a part of the RTMA component, starts running on the node with the default configuration after you install the Operations Agent.

Checking the License for Perfd Process

*You can use this component only if you use the HP Ops OS Inst to Realtime Inst LTU, Glance Pak Software LTU, or Glance Software LTU. For more information, see the *Operations Agent License Guide*.*

The **perfd** process starts the data collection only after verifying that the license is enabled. If the license is not enabled, the process is idle. All the activities are recorded in the log files.

Modifying the Settings

You can modify the configuration settings of the **perfd** process from the **perfd.ini** file, which is available into the following directory on the node:

On Windows:

%ovdatadir%

On HP-UX/Linux/Solaris:

/var/opt/perf

Parameter	Description	Default Value
interval	The frequency of data collection in seconds. This value must be a multiple or factor of 60.	10
port	The port used by perfd .	5227
depth	The time duration for which global metric values are retained in the perfd cache. This data is used for data summarization.	30
maxrps	The maximum number of session requests per second accepted by perfd . If the number of requests exceeds the limit, perfd pauses for one second, and then logs the details of this event in the log file. The log file, <code>status-perfd.<port></code> , is located into the following directory on the node: On Windows: %ovdatadir% On HP-UX/Linux/Solaris: /var/opt/perf	20
maxtpc	The maximum number of sessions per client system accepted by perfd . After the available number of sessions reaches this limit, if an additional request arrives, perfd denies the additional request.	30
maxcps	The maximum number of simultaneous session requests accepted by perfd at a given instant. If the number of requests exceeds the limit, the server will pause for 3 seconds before establishing the sessions.	2
lightweight	If this is set to True , perfd stops collecting data for processes, application, NFS operations, logical systems, and ARM. In addition, the HBA and LVM data on HP-UX will not be collected.	False
localonly	If this is set to True , perfd can be configured	False

, continued

Parameter	Description	Default Value
	<p>only on the local machine.</p> <p>If this is set to True, perfd denies all connection requests except those coming from the host system (localhost) through the loopback interface. Details of the denied connection requests are logged in the status file.</p>	
IPv4	<p>This option enables perfd to accept only IPv6 connections. By default, if perfd cannot create an IPv6 socket, it automatically switches to the IPv4-only socket.</p> <p>Note: If you explicitly want perfd to accept only IPv4 connections, set IPv4 value to true in perfd.ini file.</p>	False
logsize	<p>This parameter specifies a file size above which perfd will rollover its log or trace files. If you specify a size lesser than 4096 bytes, it is ignored.</p>	1048576 bytes
add	<p>Use this parameter to specify a comma-separated list of metric classes for which data must be collected. This parameter is useful to add individual metric classes when lightweight parameter is set to True.</p>	
exclude	<p>Use this parameter to specify a comma-separated list of metric classes for which data must not be collected. This parameter is used to exclude individual metric classes, when lightweight parameter is False.</p>	

To change the default settings, follow these steps:

1. On the node, open the **perfd.ini** file with a text editor.
2. Modify the settings.
3. Save the file.
4. Restart the Operations Agent for the changes to take effect.

Monitoring Operations Agent on IPv6 connections

After the installation of Operations Agent 12.05 is complete, it determines the supported IP configuration and binds with the corresponding IP address. Additional configuration is not required for IPv6 connections. For a dual stack node, the node determines the supported server's IP address.

Server	Dual stack node
IPv4	uses IPv4
IPv6	uses IPv6
Dual stack	uses IPv6

Note: IPv6 communication is preferred if both server and node are dual stack.

Restricting Access

You can configure the Operations Agent to prevent other systems from accessing the real-time performance data of the local system by using the RTMA component.

The **cpsh**, **padv**, and **mpadv** utilities enable you to access the real-time performance data of the agent node remotely. You can use the **cpsh** utility to view the real-time performance data on any remote system where the **perfd** process is running.

You can use the **padv** or **mpadv** utility to run **adviser** scripts on any remote system where the **perfd** process is running.

These utilities depend on the **perfd** process to access the real-time performance data.

To prevent other systems from accessing the real-time performance data of the local system by revoking their access to the **perfd** process, follow these steps:

Note: This procedure does not prevent the Diagnostic View of Performance Manager from accessing the real-time performance data from the system.

1. Log on to the system as an administrator or root. Go to the following directory:

On Windows:

%OvDataDir%\

On HP-UX/Linux/Solaris:

```
/var/opt/perf/
```

2. All other systems in the environment (where the Operations Agent is available) now cannot access the real-time performance data from this system.

3. Create an empty file in this location and save the file as **authip**.

4. To enable a system to access the real-time performance data from this system, open the **authip** file with a text editor, add the FQDN or IP address of the system, and then save the file. You can specify multiple FQDNs or IP addresses (entries must be separated by new lines).

5. Restart **perfd** to enable the changes.

Systems that are specified in the **authip** file can now access the real-time performance data from this system.

6. To allow access for all systems, delete the **authip** file.

Configuring the Security Component for Asymmetric Key

RSA certificates and asymmetric encryption are used for secure communication during the SSL handshake between the nodes and, between the node and management server.

When you install the Operations Agent:

1. The Certificate Client component (OvSecCc), sets the configuration variable **ASYMMETRIC_KEY_LENGTH** to 2048 (default value)
2. The Certificate Management component (OvSecCm) creates 2048 bit RSA key pair for communication. This RSA key pair is used for secure communication.
3. CA certificates are created with the alias **<CA_ovcoreid_keylength>**.

Example: CA_8cd78962-ab51-755c-1279-85f5ba286e97_2048

2048 is the default key length.

To increase the strength of encryption, you must increase the length of the RSA Key. Use the **ASYMMETRIC_KEY_LENGTH** configuration variable to set the length of the RSA key in the **sec.cm** namespace.

Follow the steps to set the `ASYMMETRIC_KEY_LENGTH` configuration variable and to apply the configuration changes on the management server and managed nodes:

On the Management Server

Follow these steps to update the configuration variable on the management server:

1. Update the configuration variable `ASYMMETRIC_KEY_LENGTH` using the following command:

```
ovconfchg -ns sec.cm -set ASYMMETRIC_KEY_LENGTH <RSA Encryption algorithm supported key length>
```

2. Go to the following location on the management server:

On Windows:

```
%ovinstalldir%\bin\seccs\install\
```

On HP-UX/Linux/Solaris:

```
%ovinstalldir%/bin/seccs/install/
```

3. Run the `MigrateAsymKey` tool:

On Windows:

```
cscript MigrateAsymKey.vbs -createCAcert
```

On HP-UX/Linux/Solaris:

```
./MigrateAsymKey.sh -createCAcert
```

The `MigrateAsymKey`:

- Creates a new CA's key pair,
- Creates a new CA certificate according to the new key length value set in the configuration variable `ASYMMETRIC_KEY_LENGTH`.
- Updates trusted certificates for local agent and all other OV Resource Groups (ovrg's) on the management server,
- Creates a new certificate for the local agent and all ovrg's.

Note: During an upgrade of the Certificate Server component, running the migration tool with `-createCAcert` option is performed after installation. `-createCAcert` is used to create CA certificate corresponding to the new configuration value set to variable `ASYMMETRIC_KEY_LENGTH` and update the CA certificate created for agent on server and all ovrg's. So, do not run the migration tool with the `-createCAcert` option unless there is any configuration change made to the `ASYMMETRIC_KEY_LENGTH` configuration variable.

4. To update the trusted certificates on all the managed nodes, run the following command:

```
ovcert -updatetrusted
```

Note: You must run the `ovcert -updatetrusted` command on all the managed nodes before you run MigrateAsymKey tool with `--createNodecert` option.

5. To create a new node certificate for agent on the management server and all ovrg's on the server, run the command:

On Windows:

```
cscript MigrateAsymKey.vbs -createNodecert
```

On HP-UX/Linux/Solaris:

```
./MigrateAsymKey.sh -createNodecert
```

The command creates the new node certificates for Operations Agent on the management server and all ovrg's with the new RSA key pair.

Note: You can verify the updated key length by the running the command: `ovcert -certinfo <certificate_alias>`. The key length is updated with the `ASYMMETRIC_KEY_LENGTH`.

On the Agent node

To apply the configuration changes on the Agent node (managed node), follow these steps:

1. Update the configuration variable `ASYMMETRIC_KEY_LENGTH` using the following command:

```
ovconfchg -ns sec.cm -set ASYMMETRIC_KEY_LENGTH <RSA Encryption algorithm supported key length>
```

2. To remove the existing node certificate on the agent, run the following commands:

```
ovcert -remove<certificate alias>
```

```
ovcert -remove <CA certificate alias>
```

3. To request a new node certificate from the management server, run the following command:

```
ovcert -certreq
```

Configuring the Security Component for Symmetric Key

When you install Operations Agent, the security components OvSecore and OvJSecCore will continue to use the older default algorithm values for the encryption and decryption. Both OvSecCore and OvJSecCore are included in the Shared Component packages for Windows from version 11.10 onwards.

The supported symmetric algorithms are as follows:

- Blowfish
- DES
- DES3
- AES128
- AES192
- AES256

To change the default symmetric key algorithm, use the following configuration variables:

- **DEF_SYM_KEY_ALGO** - This variable is used to set the default symmetric key algorithm for encryption. Set this configuration variable in the `sec.core` namespace. The supported algorithm values are:
 - eBlowfish
 - eDES
 - eDES3
 - eAES128
 - eAES192
 - eAES256
 - eDefault – uses AES128 as the default algorithm.

Note: To enable FIPS compliance, use one of the following FIPS compliant symmetric algorithms:

- 3DES
 - AES128
 - AES198
 - AES256
- **ENABLE_DEF_SYM_KEY_ALGO** - Set this to TRUE to enable the use of the default symmetric key algorithm set in DEF_SYM_KEY_ALGO.

Note: The MigrateSymKey tool sets the ENABLE_DEF_SYM_KEY_ALGO configuration variable to TRUE if it is not set.

Follow the steps to update the configuration settings on the management server and on the nodes running with the Operations Agent:

1. Run the following command to set the configuration variable DEF_SYM_KEY_ALGO to any of the supported algorithms:

```
ovconfchg -ns sec.core -set DEF_SYM_KEY_ALGO <Supported Algorithm>
```

Note: If the DEF_SYM_KEY_ALGO variable is not set and the ENABLE_DEF_SYM_KEY_ALGO variable is set to TRUE, eAES128 is used as the default algorithm.

2. Go to the following location on the management server or on the node where Agent is installed:

On Windows:

```
%ovinstalldir%\bin\secco\
```

On HP-UX/Linux/Solaris:

```
%ovinstalldir%/lbin/secco/
```

3. Run the MigrateSymKey tool:

On Windows:

```
MigrateSymKey -sym_key_algo [eBlowfish | eDES | eDES3 | eAES128 | eAES192 | eAES256]
```

On HP-UX/Linux/Solaris:

```
./MigrateSymKey.sh -sym_key_algo [eBlowfish | eDES | eDES3 | eAES128 | eAES192 | eAES256]
```


The MigrateSymKey tool sets ENABLE_DEF_SYM_KEY_ALGO configuration variable to TRUE and migrates the KeyStore content based on the algorithm set to the configuration variable DEF_SYM_KEY_ALGO.

Usage:

MigrateSymKey

```
-sym_key_algo [eBlowfish | eDES | eDES3 | eAES128 | eAES192 | eAES256]  
-help
```

The -sym_key_algo:

- Updates "ENABLE_DEF_SYM_KEY_ALGO" to TRUE if not set.
- If symmetric key algorithm is specified, DEF_SYM_KEY_ALGO is updated with the specified value.
- If algorithm is not specified, eAES128 is used as symmetric key algorithm.

Configuring the Security Component with Hash Algorithm

Operations Agent 12.05 and above supports the configurable hash algorithm for hashing. **HASH_ALGO** is the configuration variable provided to set the hash algorithm. This configuration variable is available under the **sec.core** namespace.

HASH_ALGO supports the following hash algorithms:

- eMD5
- eSHA1
- eSHA224
- eSHA256
- eSHA384
- eSHA512
- eDefault

Note: To enable FIPS compliance, use one of the following FIPS complaint hash algorithms:

- SHA1
- SHA224
- SHA256
- SHA384
- SHA512

Set the configuration variable `HASH_ALGO` to any of the supported algorithms using the following command:

```
ovconfchg -ns sec.core -set HASH_ALGO <Supported Algorithm>
```

Example: `ovconfchg -ns sec.core -set HASH_ALGO=eSHA224`

If the `HASH_ALGO` variable is set to `eDefault` (or any other non-supported value), then `sec.core` uses `eSHA256` as the `HASH_ALGO` value.

Hashing is also used for securing keystore of Operations Agent. To enable the use of the algorithm specified in `HASH_ALGO` for securing keystore, you must use `MigrateSymKey` tool. `MigrateSymKey` tool sets the `HASH_ALGO_AS_SEED` configuration variable to `TRUE` in the `sec.core` namespace and migrates the keystore content.

Note:

- Before setting `HASH_ALGO_AS_SEED` configuration variable to `TRUE`, ensure that the version of `OvSecCore` and `OvJSecCore` components is 12.00 or above.
- After `HASH_ALGO_AS_SEED` is set to `TRUE`, you must only use `MigrateSymKey` tool to update the `HASH_ALGO` value. Do not use `ovconfchg` to set the `HASH_ALGO_AS_SEED`.
- The management server and the Agent node must use the same `HASH_ALGO` value for `ovcert -certreq` and `ovcm -grant` to work when `HASH_ALGO_AS_SEED` is set to `TRUE`.
- Do not revert `HASH_ALGO_AS_SEED` value after you set it.

Follow the steps to use the `MigrateSymKey` tool:

1. Go to the following location on the management server or on the node where Operations Agent is installed:

On Windows:

```
%ovinstalldir%\lbin\secco\
```

On HP-UX/Linux/Solaris:

```
/opt/OV/lbin/secco/
```

On AIX:

```
/usr/lpp/OV/lbin/secco/
```

2. Run the migration tool:

```
MigrateSymKey -hash_algo [eMD5 | eSHA1 | eSHA224 | eSHA256 | eSHA384 | eSHA512]
```

After running the tool, the KeyStore content is migrated based on the algorithm value specified and `HASH_ALGO_AS_SEED` is set to `TRUE`.

Usage:

```
MigrateSymKey
```

```
-hash_algo [eMD5 | eSHA1 | eSHA224 | eSHA256 | eSHA384 | eSHA512]
```

```
-help
```

- `-hash_algo` updates the following:
 - Updates `"HASH_ALGO_AS_SEED"` to `TRUE` if not set.
 - Updates `"HASH_ALGO"` configuration variable to the specified algorithm.
 - If algorithm is not specified, it is updated with `eSHA256`.

Verification

- If `MigrateSymKey` tool completes the action successfully, it prints the following messages:
"Migration successful".
- After running the `MigrateSymKey` tool, verify if `HASH_ALGO_AS_SEED` is set to `TRUE` and `HASH_ALGO` is set to the specified algorithm.

Configuring FIPS Compliant Operations Agent

Federal Information Processing Standard (FIPS) is a security standard used to endorse cryptographic modules that are used to protect the unclassified, valuable and sensitive information in hardware and software products. To enforce secure communication across a network environment and to manage all the requirements for cryptography modules, Operations Agent has been enhanced to make it FIPS compliant.

Prerequisites to make Operations Agent FIPS Compliant

1. Make sure that Operations Agent version 12.00 or above is installed.
2. Make sure the length of the RSA key is greater than or equal to 2048 bits.

To set or change the length of the RSA key, see [Configuring the Security Component for Asymmetric Key](#).

3. Make sure you use a FIPS complaint symmetric algorithm for encryption and decryption. By default, AES128 is used as the FIPS complaint symmetric algorithm. To change the default symmetric algorithm, see [Configuring the Security Component for Symmetric Key](#).
4. Make sure you use a FIPS complaint hash algorithm for encryption and decryption. To change the default hash algorithm, see [Configuring the Security Component with Hash Algorithm](#).

Enabling FIPS Compliance using FIPS_tool

Use the **FIPS_tool** provided by the OvSecCore component to enable FIPS compliance. The FIPS_tool is available in the following location:

On Windows

```
%ovinstalldir%\lbin\secco\
```

On Linux

```
/opt/OV/lbin/secco/
```

Usage:

```
FIPS_tool  
  -enable_FIPS [-Java_Home <jre_dir_path>]  
  -disable_FIPS [-Java_Home <jre_dir_path>]  
  -help
```

Follow the steps to enable FIPS mode using the FIPS_tool:

1. Go to the following location on the management server or on the node where Operations Agent is installed:

On Windows

```
%ovinstalldir%\bin\secco\
```

On UNIX

```
/opt/0V/lbin/secco/
```

On AIX

```
/usr/lpp/0V/lbin/secco
```

2. Run the FIPS_tool to enable the FIPS Mode:

```
FIPS_tool -enable_FIPS [-Java_Home <jre_dir_path>]
```

The ENABLE_FIPS_MODE configuration variable is set to TRUE.

In this instance, -Java_Home specifies the jre directory path.

For example:

```
FIPS_tool -enable_FIPS -Java_Home /opt/0V/non0V/jre/b
```

```
FIPS_tool -enable_FIPS -Java_Home /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.51-  
2.4.5.5.e17.x86_64/jre
```

Note: Run the FIPS_tool to disable the FIPS mode:

```
FIPS_tool -disable_FIPS [-Java_Home <jre_dir_path>]
```

For example:

```
FIPS_tool -disable_FIPS -Java_Home /opt/0V/non0V/jre/b
```

```
FIPS_tool -disable_FIPS -Java_Home /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.51-  
2.4.5.5.e17.x86_64/jre
```

If you run the FIPS_tool to disable the FIPS mode, FIPS mode is disabled but the configurations changes made in the FIPS mode are not reverted.

Configuration Settings in FIPS Mode

```
[sec.core]
DEF_SYM_KEY_ALGO=< FIPS compliant symmetric algorithm>
ENABLE_DEF_SYM_KEY_ALGO=TRUE
ENABLE_FIPS_MODE=TRUE
HASH_ALGO=< FIPS compliant hash algorithm>
HASH_ALGO_AS_SEED=TRUE
[sec.cm]
ASYMMETRIC_KEY_LENGTH=< FIPS compliant RSA key size>
[eaagt]
CRYPT_USING_LCORE=TRUE
```

For example:

```
[sec.core]
DEF_SYM_KEY_ALGO=eAES128
ENABLE_DEF_SYM_KEY_ALGO=TRUE
ENABLE_FIPS_MODE=TRUE
HASH_ALGO=eSHA256
HASH_ALGO_AS_SEED=TRUE
[sec.cm]
ASYMMETRIC_KEY_LENGTH=2048
[eaagt]
CRYPT_USING_LCORE=TRUE
```

Note:

If you enable FIPS mode on a Operations Agent nodes, you must enable FIPS mode on the management server and then redeploy all the policies from the management server.

Verifying if Operations Agent is running in FIPS mode

Run the following command:

```
ovbbccb -status
```

If Operations Agent is running in the FIPS mode, then in the command output FIPS Mode is ON.

```
# ovbbccb -status

Status: OK

(namespace, Port, Bind Address, Open Sockets)

<default> 18600 ANY 1

HP OpenView HTTP Communication Incoming Connections

BBC 12.00.078; ovbbccb 12.00.078
:::1.58782 :::1.18600

FIPS mode: ON
```

Troubleshooting

The SSL communication between the Java server and Java client fails in the FIPS mode.

The SSL communication between the Java server and Java client fails in the FIPS mode, because the Netscape Comment field is treated as Netscape CertType field.

To resolve this issue, Netscape Comment (OpenView Certificate) field is removed from OvSecCm in the current version (12.01) of the Operations Agent and certificates are reissued.

Note: Run the following command to check certificate details:

```
openssl x509 -in <cert.pem> -text -noout
```

Unrecognized SSL message, plaintext connection.

This issue occurs due to a delay caused by RSA Bsafe jars during SSL handshake when FIPS is enabled.

To resolve the issue, set the following configuration on the management server to increase the SSL handshake timeout:

```
[bbc.http]
```

```
SSL_HANDSHAKE_TIMEOUT=60000
```

IO exception: (sec.core-114) SSL connection error (unable to find ec dh parameters).

This issue occurs only when FIPS is enabled. The ECDHE algorithm is not supported in the FIPS mode. You will get this error after you enable FIPS if the ECDHE algorithm is picked by default during SSL communication.

To resolve this issue disable ECDHE algorithm in `java.security` file:

```
jdk.tls.disabledAlgorithms= ECDHE
```

Configuring the OPCMONA_POL_RUN_WITH_SHELL Variable

Opcmona Measurement Threshold policy can be used to specify command or program in the Program name or MONPROG field. The command execution fails when the Program name field is used while executing the switch user (`su`) command with environment variables. This is because the commands are executed with shell.

In the Operations Agent version 8.x, the command execution does not fail because the commands are executed without shell.

The OPCMONA_POL_RUN_WITH_SHELL variable provides backward compatibility with respect to opcmona command execution. If the variable is set to TRUE, the opcmona command execution is with shell and if set to FALSE, the opcmona command execution is without shell. You must set this variable based on your requirement (with shell or without shell).

To configure the OPCMONA_POL_RUN_WITH_SHELL variable, follow these steps:

1. Log on to the node with the necessary privileges.
2. In the command prompt, run the following command to set the OPCMONA_POL_RUN_WITH_SHELL variable:

```
ovconfchg -ns eaagt -set OPCMONA_POL_RUN_WITH_SHELL <Value>
```

In this instance, the *<Value>* is defined as one of the following:

TRUE- When set to TRUE, the opcmona command execution is with shell. The default value is TRUE.

FALSE- When set to FALSE, the opcmona command execution is without shell.

Configuring the Control Component Variable

If the `LOG_PROCESS_DISABLE` variable value is not set or set to zero (0), the control daemon logs the process details of the following in the `system.txt` file:

- The processes such as CORE OV processes (BBC, OVCONFD), Operations Agent processes (opcle, opcmona, opctrapi) and the non OV process (ovtomcatB) that is started or stopped.
- The processes such as CORE OV processes and Operations Agent processes that stops unexpectedly.

If the `LOG_PROCESS_DISABLE` variable value is set to 1, the control daemon does not log any process details (related to starting and stopping of the process) in the `system.txt` file.

You can set the `LOG_PROCESS_DISABLE` variable in the `ctrl.ovcd` namespace.

To configure the `LOG_PROCESS_DISABLE` variable, follow these steps:

1. Log on to the node with the necessary privileges.
2. Run the following command to set the `LOG_PROCESS_DISABLE` variable:

```
ovconfchg -ns ctrl.ovcd -set LOG_PROCESS_DISABLE <value>
```

In this instance, `<value>` specifies the value that you want to assign to the variable. The value is set to either one or zero (0).

Note:

- To set the `LOG_PROCESS_DISABLE` variable to 1, run the following command:

```
ovconfchg -ns ctrl.ovcd -set LOG_PROCESS_DISABLE 1
```

The default value of the `LOG_PROCESS_DISABLE` variable is zero (0).

- To set the `LOG_PROCESS_DISABLE` variable to zero (0), run the following command:

```
ovconfchg -ns ctrl.ovcd -set LOG_PROCESS_DISABLE 0
```

3. Run the following commands to restart the agent processes for the changes to take effect:
 - a. `ovc -kill`
 - b. `ovc -start`

Troubleshooting

The Control daemon does not log the process details in the `system.txt` file whenever a process is started or stopped.

When a process is started or stopped the Control daemon does not log the process details such as the process name, pid in the `system.txt` file.

To resolve this issue, the XPL traces of `ovc` and `ovcd` processes must be captured.

Registering the user-defined process with OvCtrl

The Operations Control (OvCtrl) maintains registration files for each component and the control registration files are written using XML syntax. The registration file contains information about the components.

After registration, the files are maintained by OvCtrl in the following directory:

```
<OvDataDir>/conf/ctrl
```

There is one separate registration file for each registered component. You can check the correctness of a registration file in the `<OvDataDir>/conf/ctrl` directory using the following command:

```
ovcreg -check
```

To register the components that are mentioned in the registration file and for the OvCtrl to replace its registration file, run the following command:

```
ovcreg -add
```

Note: The `ovcreg -add` command copies the component's registration file to `<OvDataDir>/conf/ctrl` directory and replaces the directory variables in the registration file with the values for the target platform.

Note: Only the process which runs continuously can be registered with OvCtrl.

Follow the steps to create an XML file for registering a process with OvCtrl:

1. Create an XML file using the below given sample XML template:

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

```
<ovc:OvCtrl
xmlns:ovc="http://openview.hp.com/xmlns/ctrl/registration/1.5">
  <ovc:Component>
    <ovc:Name>Name of the component</ovc:Name>
    <ovc:Label>
      <ovc:String>Display name when printing the status of a
      component</ovc:String>
    </ovc:Label>
    <ovc:Category>Name of the Category</ovc:Category>
    <ovc:Options>
      <ovc:AutoRestart>boolean value true or false</ovc:AutoRestart>
      <ovc:AutoRestartLimit>seconds</ovc:AutoRestartLimit>
      <ovc:AutoRestartMinRuntime>seconds</ovc:AutoRestartMinRuntime>
      <ovc:AutoRestartDelay>seconds</ovc:AutoRestartDelay>
    ></ovc:Options>
    ><ovc:ProcessDescription>String that uniquely identifies the component in
    the process table</ovc:ProcessDescription>
    <ovc:OnHook>
      <ovc:Name>START</ovc:Name>
      <ovc:Actions>
        <ovc:Start>
          <ovc:CommandLine>Path of the component(binary file or application) which
          specifies the start sequence of the component </ovc:CommandLine>
        </ovc:Start>
      </ovc:Actions>
    </ovc:OnHook>
    <ovc:OnHook>
      <ovc:Name>STOP</ovc:Name>
```

```
<ovc:Actions>
<ovc:WinEvent>
<ovc:Name>Specifies the stop sequence of the component</ovc:Name>
</ovc:WinEvent>
</ovc:Actions>
</ovc:OnHook>
</ovc:Component>
</ovc:OvCtrl>
```

For example:

A dummy process named runforever is created and the following example shows the structure of the **runforever.xml** file:

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<ovc:OvCtrl
xmlns:ovc="http://openview.hp.com/xmlns/ctrl/registration/1.5">
<ovc:Name>runforever</ovc:Name>
<ovc:Label>
<ovc:String>My runforever</ovc:String>
</ovc:Label>
<ovc:Category>AGENT</ovc:Category>
<ovc:Category>OA</ovc:Category>
<ovc:Options>
<ovc:AutoRestart>true</ovc:AutoRestart>
<ovc:AutoRestartLimit>10</ovc:AutoRestartLimit>
<ovc:AutoRestartMinRuntime>60</ovc:AutoRestartMinRuntime>
<ovc:AutoRestartDelay>5</ovc:AutoRestartDelay>
</ovc:Options>
<ovc:ProcessDescription>runforever</ovc:ProcessDescription>
```

```

<ovc:OnHook>

<ovc:Name>START</ovc:Name>

<ovc:Actions>

<ovc:Start>

<ovc:CommandLine>"/var/opt/OV/conf/ctrl/test.sh"</ovc:CommandLine>

</ovc:Start>

</ovc:Actions>

</ovc:OnHook>

<ovc:OnHook>

<ovc:Name>STOP</ovc:Name>

<ovc:Actions>

<ovc:UXSignal>

<ovc:Name>SIGTERM</ovc:Name>

</ovc:UXSignal>

</ovc:Actions>

</ovc:OnHook>

</ovc:Component>

</ovc:OvCtrl>

```

2. Specify the details in the XML file using the tag names as shown in the following table:

Tag Names	Description	Example
<ovc:OvCtrl> <ovc:Component><ovc: :Name>	Name of the component (required) Note: The component is generally a process.	<ovc:OvCtrl xmlns:ovc="http://openview.hp.com/ xmlns/ctrl/registration/1.5"> <ovc:Component><ovc:Name> runforever </ovc:Name>
<ovc:Label><ovc:Str	Label name	<ovc:Label><ovc:String> My runforever

Tag Names	Description	Example
ing>	(required) - The unique label of the component that is used when printing the status of the component.	</ovc:String></ovc:Label>
<ovc:Category>	Category name (optional) - The way to group components together to make the operation easier.	<ovc:Category>Agent</ovc:Category> <ovc:Category>OA</ovc:Category>
<ovc:Options> <ovc:AutoRestart> <ovc:AutoRestartLimit> <ovc:AutoRestartMinRuntime> <ovc:AutoRestartDelay>	Options (optional) - Set the boolean value for the process Autorestart to TRUE or FALSE. 1	<ovc:Options> <ovc:AutoRestart>true</ovc:AutoRestart> <ovc:AutoRestartLimit> 10</ovc:AutoRestartLimit> <ovc:AutoRestartMinRuntime>60</ovc:AutoRestartMinRuntime> <ovc:AutoRestartDelay> 5</ovc:AutoRestartDelay> </ovc:options>
<ovc:ProcessDescription>	Process description (required) - The string name that uniquely identifies the process in the process table.	<ovc:ProcessDescription>runforever </ovc:ProcessDescription>
<ovc:OnHook> <ovc:Name> <ovc:Actions> <ovc:Start> <ovc:CommandLine>	OnHook (Only START is required) A hook includes the following: <ul style="list-style-type: none">Name (required)	<ovc:OnHook> <ovc:Name>START</ovc:Name> <ovc:Actions> <ovc:Start> <ovc:CommandLine>

Tag Names	Description	Example
	<ul style="list-style-type: none"> • START_CHECK - Specify the sequence of actions that must complete successfully before starting the component. This can be used to conditionally start a component. • START - Specifies the start sequence of the component. • INITIALIZE - Specifies the additional actions that must be completed successfully before a component is considered as running. • STOP - Specifies the stop sequence 	<pre>"/var/opt/OV/conf/ctrl/test.sh" </ovc:CommandLine> </ovc:Start> </ovc:Actions></ovc:OnHook></pre>

Tag Names	Description	Example
	<p>of the component.</p> <ul style="list-style-type: none"> • CHECK_STATUS: Specifies the status check sequence of the contained component. • Action (required): Specify one or more actions to be executed for this hook. ◦ CommandLine (optional) 	
<pre><ovc:OnHook> <ovc:Name> <ovc:Actions> <ovc:UXSignal> <ovc:Component> <ovc:OvCtrl></pre>	<p>STOP - Specify the stop sequence of the component.</p> <p>UXSignal - It is UNIX specific signal sent to the component. The signal name can be specified.</p> <p>The other possible actions are:</p> <p>WinEvent - It is Windows specific and sends events only on Windows.</p> <p>Execute - Run a</p>	<pre><ovc:OnHook> <ovc:Name>STOP</ovc:Name> <ovc:Actions> <ovc:UXSignal> <ovc:Name>SIGTERM</ovc:Name> </ovc:UXSignal></ovc:Actions> </ovc:OnHook></ovc:Component> </ovc:OvCtrl></pre>

Tag Names	Description	Example
	command and wait for it complete. You can specify the environment.	

- Save the completed XML file in the following directory:

Note: The filename must have the `<process name.xml>` format.

On Windows:

```
%ovdatadir%\conf\ctrl
```

For example:

```
%ovdatadir%\conf\ctrl\runforever.xml
```

On UNIX/Linux:

```
/var/opt/OV/conf/ctrl/
```

For example:

```
/var/opt/OV/conf/ctrl/runforever.xml
```

- Run the following command to register the components mentioned in the registration file and for OvCtrl to replace its registration file:

```
ovcreg -add
```

- Run the command to start and stop the process:

- o `ovc -start <processname>`

For example: `ovc -start runforever`

- o `ovc -stop <processname>`

For example: `ovc -stop runforever`

Monitoring Windows Event Logs

The Logfile Encapsulator component of the Operations Agent supports the Event Forwarding Feature of Windows Event Log. That is, events which are forwarded from different machines can be read and it

also enables you to monitor Windows event logs. The Windows Event Log policies help you configure the agent to monitor Windows event logs of your choice.

The following versions of Windows provide a new category of event logs—Applications and Services logs:

- Windows Vista
- Windows Server 2008
- Windows Server 2008 R2
- Windows 7

You can monitor these Applications and Services logs with the Operations Agent with appropriately configured Windows Event Log policies.

The Operations Agent cannot monitor the following types of event logs:

- Events originating from a remote system (collected by using the Event Subscription feature of Windows)
- Saved event logs

The Operations Agent can monitor events with the following event levels:

- Error
- Information
- Warning
- LOG_ALWAYS
- VERBOSE
- Audit Failure
- Audit Success

The following table shows how eventlog fields are displayed in the message browser.

Table 1 Eventlog to Message Browser Field Correlation

Eventlog Field	Message Browser Field	Comments
Date	Date	The date the event was created on managed node.
Time	Time	The time the event was created on managed node.
Event ID	Message Text	The Event ID will be displayed before

Table 1 Eventlog to Message Browser Field Correlation , continued

Eventlog Field	Message Browser Field	Comments
		any additional message text.
Source	Application	None
Type <i>error</i> <i>error</i> <i>information</i> <i>warning</i> <i>log_always</i> <i>verbose</i> <i>audit failure</i> <i>audit success</i>	Severity <i>critical</i> <i>error</i> <i>normal</i> <i>warning</i> <i>normal</i> <i>normal</i> <i>warning</i> <i>normal</i>	The mapping of Event log type severity to the Operations Manager message severity.
Category	Object	None
Description	Message Text	All other message text (after the Event ID).
User	Not mapped	Not mapped
Computer	Node	The name of the node as it is known to the management server.
	Msg Group	Empty

Monitor Applications and Services Event Logs from OM for Windows

To create a Windows Event Log policy for monitoring Applications and Services log, follow these steps:

1. Log on to the Windows node where the Windows event log exists.
2. Open the Event Viewer window.
3. In the console tree, select the event. In the details pane, the name of the event log appears (next to the Log Name field).

Log Name:	Microsoft-Windows-Bits-Client/Operational
Source:	Bits-Client
Event ID:	306

Note down the name of the log file as it appears in the details pane.

4. Open the OM for Windows console.
5. In the console tree, under Agent Policies Grouped by Type, right-click **Windows Event Log**, and then click **New > policy**.

The policy editor for the Windows Event Log policy opens.

6. In the Source tab, type the name of the Windows event log (which you noted down in step 3) in the Event Log Name field.



Event log name* Microsoft-Windows-Bits-

7. Follow the instructions in the OM for Windows online help to specify other details in the policy.
8. Save the policy.
9. Deploy the policy on the Windows node.


Monitor Applications and Services Event Logs from OM on UNIX/Linux 9.xx

To create a Windows Event Log policy for monitoring an Applications and Services log, follow these steps:

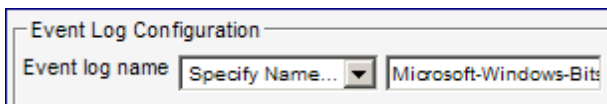
1. Log on to the Windows node where the Windows event log exists.
2. Open the Event Viewer window.
3. In the console tree, select the event. In the details pane, the name of the event log appears (next to the Log Name field).

Log Name:	Microsoft-Windows-Bits-Client/Operational
Source:	Bits-Client
Event ID:	306

Note down the name of the log file as it appears in the details pane.

4. Log on to the OM for UNIX Administration UI.
5. Click **OMU**.
6. Click **Browse > All Policy Types**.
7. Click Windows Event Log. The Policy Type **Windows_Event_Log** page opens.
8. Click  , and then click **New Policy**. The Add Windows_Event_Log Policy page opens.

In the Source tab, in the Event Log Name field, select Specify Name. A new text box appears. Type the name of the Windows event log (which you noted down in step 3) in the text box.



9. Follow the instructions in the OM for UNIX online help to specify other details in the policy.
10. Save the policy.
11. Deploy the policy on the Windows node.

1

Note: If AutoRestart option is set to TRUE, AutoRestartLimit, AutoRestartMinRuntime and AutoRestartDelay options are enabled.

<AutoRestart> This option restarts the process if it terminates unexpectedly. The default value is False.

<AutoRestartLimit> specifies the maximum number of times the process restarts. The default value is 5 seconds.

<AutoRestartMinRuntime> specifies how long in seconds the process has to run before it can be restarted automatically. The default value is 60 seconds.

<AutoRestartDelay> specifies how long in seconds OvCtrl waits before the automatic restart of the process. The default value is 5 seconds.

The other possible options include:

<AllowAttach> specifies the OvCtrl not to kill the component if it is already started but to attach it. The default value is false.

<MentionInStatus> When set to TRUE, mentions the status of the component in the status report. If set to FALSE, the status of the component will not be listed in the status report.

<Monitored> When a component terminates unexpectedly and if set to TRUE, it reports error, but when set to FALSE, it reports no error. The default value is TRUE.

<StartAtBootTime> Evaluate when -boot is specified. When set to FALSE the component is not started at boot time. The default value is TRUE.

<CoreProcess> specifies the OvCtrl not to stop the component unless -kill is used. The default value is FALSE.

<IsContainer> OvCtrl uses this component as a container for other components. The default value is FALSE.

<AutoShutdown> Used for container components. When set to TRUE, OvCtrl stops the container when all its contained components are stopped. The default value is FALSE.

<AutoShutdownTimer> used for container components. Specifies how long OvCtrl will wait before it will stop the container component when all of its contained components are stopped. The default value is 30 seconds.

<Container> used to denote a contained component. Defines a name of the container for this (contained) component.

<PollingInterval> used for contained components. Defines how often OvCtrl will poll the container to obtain the run status of the contained component. The default value is 30 seconds.

Chapter 2: Adviser for the RTMA Component

You can use the adviser feature only if you enable the HP Ops OS Inst to Realtime Inst LTU or Glance Pak Software LTU.

This topic focuses on the adviser feature that can be used with the RTMA component. The GlancePlus software offers additional features that can be used with the adviser utility. For information about using the adviser feature with the GlancePlus software, see *GlancePlus Online Help*.

The *Adviser* feature enables you to generate and view alarms when values of certain metrics, collected by the RTMA component, exceed (or fall below) the set threshold. The **adviser script** and **padv** utility build up the adviser feature. The adviser script helps you create the rules to generate alarms when the performance of the monitored system shows signs of degradation. The **padv** utility helps you run the adviser script on a system of your interest.

Alarms and Symptoms

Alarms enable you to highlight metric conditions. The adviser script enables you to define threshold values for metrics that are monitored by the RTMA component. When the metric value traverses beyond the set threshold, the RTMA component generates an alarm in the form of an alert message. This message is sent in the form of `stdout` to the **padv** utility.

An alarm can be triggered whenever conditions that you specify are met. Alarms are based on any period of time you specify, which can be one interval or longer.

A symptom is a combination of conditions that affects the performance of your system.

By observing different metrics with corresponding thresholds and adding values to the probability that these metrics contribute to a bottleneck, the adviser calculates one value that represents the combined probability that a bottleneck is present.

Working of the Adviser Script

When you run the **padv** command, the Operations Agent scans the script specified with the command and takes necessary actions. If you do not specify any script file with the **padv** command, the adviser utility retrieves necessary information from the following default script file:

On Windows:

```
%ovdatadir%\perfd
```

On HP-UX/Linux/Solaris:

```
/var/opt/perf/perfd
```

If you want to run the script that includes operating system-specific diagnosis and actions, use the following default scripts:

On Windows:

```
%ovdatadir%\perfd\os\<os_type>\adv
```

On HP-UX/Linux/Solaris:

```
/var/opt/perf/perfd/os/<os_type>/adv
```

In this instance, *<os_type>* specifies the operating system on the node where you want to run the script.

As a result of running the adviser script, you can achieve the following:

- Print the system status based on generated alarms into a text file
- View the real-time status of the system in the command console from where you ran the **padv** command.

Using Adviser

To use the adviser component to monitor the real-time health of the system, follow these steps:

1. Configure the adviser script according to your requirements. Sample scripts are available in the following directory:

On HP-UX/Linux/Solaris:

```
/opt/perf/examples/adviser
```

2. Identify the node where you want to run the script.
3. Make sure the `perfd` process runs on the identified system.
4. Run the following command:

```
padv -s <script_name> -n<system_name>
```

The adviser script starts running on the specified system and creates results based on the configuration of the script file.

Tip: While using the scripts on a remote system, make sure that the `perfd` process runs on the remote system. You can prevent other systems from running the adviser script on the local system. For more information, see [Restrict Access](#).

Running the Adviser Script on Multiple Systems

You can use the `mpadv` command to run the adviser script on multiple systems. To use the `mpadv` command, follow these steps:

1. Identify the nodes where you want to run the script.
2. Create a text file listing the names of all the identified systems.
3. Save the text file on the local system.
4. Configure the adviser script according to your requirements. Sample scripts are available in the following directory:

On HP-UX/Linux/Solaris:

```
/opt/perf/examples/adviser
```

5. Make sure the `perfd` process runs on the identified system.
6. Run the following command:

```
mpadv -l <system_list_text_file> -s <script_name>
```

The adviser script starts running on the systems specified in the `<system_list_text_file>` file and shows results based on the configuration of the script file.

Adviser Syntax

The Adviser syntax is a simple script language that enables you to set alarms and define symptom conditions.

A default syntax file— `adviser.syntax`— is provided in the following directory:

On Windows:

```
%ovdatadir%\perf
```

On HP-UX/Linux/Solaris:

```
/var/opt/perf
```

You can edit the syntax file to define your own alarms and symptoms.

Syntax Conventions

- Braces ({ }) indicate that one of the choices is required.
- Brackets ([]) indicate an optional item.
- Items separated by commas within brackets or braces are options. Choose only one.
- Italics indicate a variable name that you will replace.
- Adviser syntax keywords must always be written in the capital case.

Comments

Syntax:

```
# [any text or characters]
```

or

```
// [any text or characters]
```

You can precede comments either by double forward slashes (*//*) or the *#* sign. In both cases, the comment ends at the end of the line.

Conditions

A condition is defined as a comparison between two metric names, user variables, or numeric constants.

```
item1 {>, <, >=, <=, ==, !=} item2 [OR item3 \  
    {>, <, >=, <=, ==, !=} item4]
```

or:

```
item1 {>, <, >=, <=, ==, !=} item2 [AND item3 \  
    {>, <, >=, <=, ==, !=} item4]  
("==" means "equal", and "!=" means "not equal".)
```

Conditions are used in the ALARM statement and the IF statement. They can be used to compare two numeric metrics, variables or constants, and they can also be used between two string metric names, user variables or string constants. For string conditions, only == or != can be used as operators.

You can use compound conditions by specifying the OR or AND operator between subconditions.

Examples:

```
gbl_swap_space_reserved_util > 95
proc_proc_name == "test" OR proc_user_name == "tester"
proc_proc_name != "test" AND
    proc_cpu_sys_mode_util > highest_proc_so_far
```

Constants

Constants can be either alphanumeric or numeric. An alphanumeric constant must be enclosed in double quotes. There are two kinds of numeric constants: integer and real. Integer constants can contain only digits and an optional sign indicator. Real constants can include a decimal point.

Examples:

345	Numeric integer
345.2	Numeric real
"Time is"	Alphanumeric literal

Expressions

Use expressions to evaluate numerical values. An expression can be used in a condition or an action.

An expression can contain:

- Numeric constants
- Numeric metric names
- Numeric variables
- An arithmetic combination of the above
- A combination of the above grouped together using parentheses

Examples:

Iteration + 1

3.1416

```
gbl_cpu_total_util - gbl_cpu_user_mode_util
```

```
( 100 - gbl_cpu_total_util ) / 100.0
```

Metric Names in Adviser Syntax

You can directly reference metrics anywhere in the Adviser syntax. You can use the following types of metrics in the Adviser syntax:

- Global metrics (prefixed with gbl_ or tbl_)
- Application metrics (prefixed with app_)
- Process metrics (prefixed with proc_)
- Disk metrics (prefixed with bydisk_)
- By CPU metrics (prefixed with bycpu_)
- File system metrics (prefixed with fs_)
- Logical volume metrics (prefixed with lv_)
- Network interface metrics (prefixed with bynetif_)
- Swap metrics (prefixed with byswp_)
- ARM metrics (prefixed with tt_ or ttbin_)
- PRM metrics (prefixed with prm_)
- Locality Domain metrics (prefixed by ldom_)

You can only use process, logical volume, disk, file system, LAN, and swap metrics within the context of a LOOP statement.

Metric names can contain alphanumeric (for example, gbl_machine or app_name) or numeric data and can reflect several different kinds of measurement. For example, the metric ending of a metric name indicates what is being measured:

- a `_util` metric measures utilization in percentages
- a `_rate` metric measures units per second
- a `_queue` metric measures the number of processes or threads waiting for a resource

If you are unsure of the unit of measure for a specific metric, refer to the metric definition document.

You must associate an application metric with a specific application, except when using the LOOP statement. To do this, specify the application name followed by a colon, and then the metric name. For example, `other_apps:app_cpu_total_util` specifies the total CPU utilization for the application `other_apps`. Refer to the ALIAS statement description for more information on using application metrics in the syntax.

Application names, as defined by the parm file, may contain special characters and embedded blanks. To use these names in the syntax (where application names must match the form of a variable name), the names are made case-insensitive and embedded blanks are converted to underlines. This means that the application name defined as "Other Apps" may be referenced in the syntax as `other_apps`. For application names defined with special characters, you must use the ALIAS statement to specify an alternate name.

When explicitly qualified, application metrics may be referenced anywhere in the syntax. Unqualified application metrics may only be referenced within the context of the LOOP statement. This is an iterative statement which implicitly qualifies application or process metrics.

You can only reference process metrics within the context of a LOOP statement. There is no way to explicitly reference a process.

Printlist

The printlist is any combination of properly formatted expressions, Metric Names, user variables, or constants. See the examples for the proper formatting.

- Expression examples:

`expression [|width[|decimals|]`

Metric Names or User Variable examples:

`metric names [|width[|decimals|]`

or

`user variables [|width[|decimals|]`

The metric names or user variables must be alphanumeric.

- Constant examples:

No formatting is necessary for constants.

Formatted Examples:

`gb1_cpu_total_util|6|2` formats as '100.00'

```
(100.32 + 20)|6    formats as      ' 120'
gb1_machine|5      formats as      '7013/'
"User Label"       formats as      "User Label"
```

Variables

Variables must begin with a letter and can include letters, digits, and the underscore character.

Variables are not case-sensitive.

Define a variable by assigning something to it. The following example defines the numeric variable `highest_CPU_value` by assigning it a value of zero.

```
highest_CPU_value = 0
```

The following example defines the alphanumeric variable `my_name` by assigning it a null string value.

```
my_name = ""
```

ALARM Statement

Use the ALARM statement to notify you when certain events, which you define, occur on your system.

Using the ALARM statement, the adviser script can notify you through messages sent to the originating console of the `padv` command.

Syntax:

```
ALARM condition [FOR duration {SECONDS, MINUTES, INTERVALS}]
    [condition [FOR duration {SECONDS, MINUTES, INTERVALS}] ] ...
[START statement]
[REPEAT [EVERY duration [SECONDS, MINUTES, INTERVAL, INTERVALS]]
    statement]
[END statement]
```

The ALARM statement must be a top-level statement. It cannot be nested within any other statement.

However, you can include several ALARM conditions in a single ALARM statement, in which case all conditions must be true for the alarm to trigger. And you can also use a COMPOUND Statement, which is executed at the appropriate time during the alarm cycle.

START, REPEAT, and END are ALARM statement keywords. Each of these keywords specifies a statement. You must have a START, REPEAT, or END in an ALARM statement, and they must be listed in the correct order.

The alarm cycle begins on the first interval that all of the alarm conditions have been true for at least the specified duration. At that time, the adviser script executes the START statement, and on each subsequent interval checks the REPEAT condition. If enough time has transpired, the statement for the REPEAT clause is executed. This continues until one or more of the alarm conditions becomes false. This completes the alarm cycle and the END statement is executed.

If you omit the EVERY specification from the REPEAT statement, the adviser script executes the REPEAT statement at each interval.

ALERT Statement

The ALERT statement is used to place a message in padv command console. Whenever an ALARM detects a problem, it can run an ALERT statement to send a message with the specified severity to the padv command console.

You can use the ALERT statement in conjunction with an ALARM statement.

Syntax:

```
[(RED or CRITICAL), (YELLOW or WARNING), RESET] ALERT printlist
```

RED and YELLOW, are synonymous with CRITICAL and WARNING.

ALIAS Statement

Use the ALIAS statement to assign a variable to an application name that contains special characters or imbedded blanks.

Syntax:

```
ALIAS variable = "alias name"
```

ALIAS Example

Because you cannot use special characters or embedded blanks in the syntax, using the application name "other user root" in the PRINT statement below would have caused an error. Using ALIAS, you can still use "other user root" or other strings with blanks and special characters within the syntax.

```
ALIAS otherapp = "other user root"
```

```
PRINT "CPU for other root login processes is: ",  
      otherapp:app_cpu_total_util
```

ASSIGNMENT Statement

Use the ASSIGNMENT statement to assign a numeric or alphanumeric value, expression, to the user variable.

Syntax:

```
[VAR] variable = expression
```

```
[VAR] variable = alphaitem
```

```
[VAR] variable = alphaitem
```

COMPOUND Statement

Use the COMPOUND statement with the IF statement, the LOOP statement, and the START, REPEAT, and END clauses of the ALARM statement. By using a COMPOUND statement, a list of statements can be executed.

Syntax

```
{  
statement  
statement  
}
```

Construct compound statements by grouping a list of statements inside braces ({}). The compound statement can then be treated as a single statement within the syntax.

Compound statements cannot include ALARM and SYMPTOM statements. Compound is a type of statement and not a keyword.

EXEC Statement

Use the EXEC statement to execute a UNIX command from within your Adviser syntax. You could use the EXEC command, for example, if you wanted to send a mail message to the MIS staff each time a certain condition is met.

Syntax

```
EXEC printlist
```

The resulting printlist is submitted to your operating system for execution.

Because the EXEC command you specify may execute once every update interval, be careful when using the EXEC statement with operating system commands or scripts that have high overhead.

IF Statement

Use the IF statement to test conditions you define in the adviser script syntax.

Syntax:

```
IF condition THEN statement [ELSE statement]
```

The IF statement tests a condition. If **True**, the statement after the THEN is executed. If the condition is **False**, then the action depends on the optional ELSE clause.

If an ELSE clause has been specified, the statement following it is executed. Otherwise, the IF statement does nothing. The statement can be a COMPOUND statement which tells the adviser script to execute multiple statements.

LOOP Statement

Use the LOOP statements to find information about your system. For example, you can find the process that uses the highest percentage of CPU or the swap area that is being utilized most. You find this information with the LOOP statement and with corresponding statements that use metric names for the system conditions on which you are gathering information.

Syntax:

```
{APPLICATION, APP, CPU, DISK, DISK_DETAIL, FILESYSTEM, FS, FS_DETAIL, LAN,  
LOGICALVOLUME, LV, LV_DETAIL, NETIF, NFS, NFS_BYSYS_OPS, NFS_OP, PRM,  
PRM_BYVG, PROCESS, PROC, PROC_FILE, PROC_REGION, PROC_SYSCALL, SWAP,  
SYSTEMCALL, SC, THREAD, TRANSACTION, TT, TTBIN, TT_CLIENT, TT_INSTANCE,  
TT_UDM, TT_RESOURCE, TT_INSTANCE_CLIENT, TT_INSTANCE_UDM, TT_CLIENT_UDM,  
LDM, PROC_LDM}  
LOOP statement
```

A LOOP can be nested within other syntax statements, but you can only nest up to five levels. The statement may be a COMPOUND statement which contains a block of statements to be executed on each iteration of the loop. A BREAK statement ends the LOOP statement.

If you have a LOOP statement in your syntax for collecting specific data and there is no corresponding metric data on your system, the adviser script skips that LOOP and continues to the next syntax statement or instruction. For example, if you have defined a LOGICAL VOLUME LOOP, but have no logical volumes on your system, the adviser script skips that LOGICAL VOLUME LOOP and continues to the next syntax statement.

Loops that do not exist on your platform generate a syntax error.

As LOOP statement iterates through each interval, the values for the metric used in the statement change. For example, the following LOOP statement executes the PRINT statement once for each active application on the system, causing the name of each application to be printed.

PRINT Statement

Use the PRINT statement to print the data you are collecting to standard output (the padv command console). You can use the PRINT statement to log metrics or calculated variables.

Syntax:

```
PRINT printlist
```

```
PRINT Example
```

```
PRINT "The Application OTHER has a total CPU of ",
      other:app_cpu_total_util, "%"
```

When started, this statement prints a message to the padv command console as follows:

Note: The Application OTHER has a total CPU of 89%.

SYMPTOM Statement

Syntax:

```
SYMPTOM variable [TYPE = {CPU, DISK, MEMORY, NETWORK}]
```

```
RULE measurement {>, <, >=, <=, ==, !=} value PROB probability
```

```
[RULE measurement {>, <, >=, <=, ==, !=} value PROB probability]
```

.

-
-

The keywords SYMPTOM and RULE are exclusive for the SYMPTOM statement and cannot be used in other syntax statements. The SYMPTOM statement must be a top-level statement and cannot be nested within any other statement.

`variable` refers to the name of this symptom. Variable names defined in the SYMPTOM statement can be used in other syntax statements, but the variable value should not be changed in those statements.

RULE is an option of the SYMPTOM statement and cannot be used independently. You can use as many RULE options within the SYMPTOM statement as you need.

The SYMPTOM variable is evaluated according to the RULEs at each interval.

Measurement is the name of a variable or metric that is evaluated as part of the RULE

Value is a constant, variable, or metric that is compared to the measurement

Probability is a numeric constant, variable, or metric

The probabilities for all **True** SYMPTOM RULEs are added together to create a SYMPTOM value. The SYMPTOM value then appears in the message in the `padv` command console.

The sum of all probabilities where the condition between measurement and value is **True** is the probability that the symptom occurs.

Chapter 3: Performance Alarms

You can use the Performance Collection Component to define alarms. These alarms notify you when **oacore** or DSI metrics meet or exceed conditions that you have defined. To define alarms, you must specify conditions on each monitored system that, when met, trigger an alert or action. You can define alarms in the alarm definitions text file, `alarmdef`.

As data is logged by **oacore** or other collectors, it is compared with the alarm definitions in the `alarmdef` file. If **oacore** or DSI metrics meet or exceed the defined conditions, an alert or action is triggered.

With the real-time alarm generator, you can perform the following tasks:

- Send alert notifications to the OM console
- Create an SNMP trap when an alert notification is generated
- Forward the SNMP trap to an SNMP trap listener
- Perform local actions on the monitored systems

You can analyze historical data against the alarm definitions and report the results using the utility program's `analyze` command.

For more information on defining alarms for DSI metrics, see [Defining Alarms for DSI Metrics](#) in the *chapter Using the DSI Data Logged into the Metrics Datastore*.

Processing Alarms

As performance data is collected by the Performance Collection Component, the collected data is compared to the alarm conditions defined in the `alarmdef` file to determine whether the conditions were met. When a condition is met, an alarm is generated and the actions defined for alarms (ALERTs, PRINTs, and EXECs) are performed.

However, if data is not logged into the database files (for instance, when the threshold parameters are set to a high value), alarms are not generated even if the alarm conditions in the `alarmdef` file are met. For more information, see [Thresholds](#).

Actions defined in the alarm definition can include:

- local actions performed by using operating system commands
- messages sent to the Network Node Manager (NNM) and OM

Alarm Generator

The alarm generator component of the Performance Collection Component processes the `alarmdef` file and the available system performance data on the local system, and then generates alarms if necessary. The alarm generator consists of the following components:

- Alarm generator server (`perfalarm`)
- Alarm generator database (`agdb`)

The alarm generator server scans the information in the `alarmdef` file and sends alerts to the destinations based on the configuration information in the `alarmdef` file. The `agdb` database includes the list of target systems for the `perfalarm` component to forward SNMP traps against specific events. You can modify the default behavior of the `perfalarm` component and access the available data in the `agdb` database with the help of the `agsysdb` utility.

Run the following command to view the list of target systems where alert notifications are sent:

```
agsysdb -l
```

Enabling `perfalarm`

By default the alarm generator server (`perfalarm`) is disabled. Use one of the following methods to enable `perfalarm`:

- **Before Installing the Operations Agent:**
 - a. Set the variable `ENABLE_PERFALARM` to **True** in the profile file:

```
set nonXPL.config:ENABLE_PERFALARM=TRUE
```

- b. Run the following command to install the Operations Agent with the profile file:

On Windows:

```
cscript oainstall.vbs -i -a -agent_profile <path>\<profile_file> -s <server_ip_address>
```

On HP-UX/Linux/Solaris:

```
./oainstall.sh -i -a -agent_profile <path>/<profile_file> -s <server_ip_address>
```

In this instance:

<profile_file> is the name of the profile file.

<path> is the complete path to the profile file.

<server_ip_address> is the IP address or host name of the management server.

- **After Installing the Operations Agent:**

- a. Run the following commands to enable per-falarm:

On Windows:

```
copy "%ovinstalldir%newconfig\alarmdef.mwc" "%ovdatadir%"  
"%ovinstalldir%bin\ovpacmd" start alarm
```

On HP-UX/Linux/Solaris:

```
cp/opt/perf/newconfig/alarmdef /var/opt/perf/  
/opt/perf/bin/ovpa start alarm
```

On AIX:

```
cp/usr/lpp/perf/newconfig/alarmdef /var/opt/perf/  
/usr/lpp/perf/bin/ovpa start alarm
```

- b. Go to the following location on the node:

On Windows 64-bit:

```
%ovinstalldir%bin\win64\OpC\install
```

On other Windows:

```
%ovinstalldir%bin\OpC\install
```

On HP-UX/Linux/Solaris:

```
/opt/OV/bin/OpC/install
```

On AIX:

```
/usr/lpp/OV/bin/OpC/install
```

- c. Run the following command to reconfigure the Operations Agent:

On Windows:

```
cscript oainstall.vbs -a -configure -agent_profile<path>/<profile_file>
```

On HP-UX/Linux/Solaris:

```
./oainstall.sh -a -configure -agent_profile<path>/<profile_file>
```

In this instance:

<profile_file> is the name of the profile file.

<path> is the complete path to the profile file.

Note: If you upgrade from Operations Agent 11.xx to 12.xx, perfalarm continues to function as defined previously.

Sending SNMP Traps to Network Node Manager

To send SNMP traps to the Network Node Manager, you must add your system name to agdb in Performance Collection Component using the command:

```
agsysdb -add systemname
```

When an ALERT is generated, an SNMP trap is sent to the system you defined. The trap text will contain the same message as the ALERT.

To stop sending SNMP traps to a system, you must delete the system name from agdb using the command:

```
agsysdb -delete systemname
```

To send Performance Collection Component traps to another node, add the following entries to **/etc/services** file.

```
snmp-trap 162/tcp # SNMPTRAP
```

```
snmp-trap 162/udp # SNMPTRAP
```

In this instance, 162 specifies port number. If you want Performance Collection Component to send traps to another node, it checks the **/etc/services** file for the snmp-trap string. If this entry is not available, the traps will not be sent to another node.

Sending Messages to OM

By default, the alarm generator does *not* execute local actions that are defined in any alarm in the EXEC statement. Instead, it sends a message to OM's event browser.

You can change the default to stop sending information to OM using the following command:

```
agsysdb -ovo OFF
```

Table 9: Settings for sending information to OM and executing local actions

OM Flag	Operations Monitoring Component Running	Operations Monitoring Component Not Running
off	No alert notifications sent to OM.	No alert notifications sent to OM.
on	Alert notifications sent to OM.	No alert notifications sent to OM.

Executing Local Actions

By default, the Performance Collection Component does not run the local commands specified in the EXEC statements.

You can change the default to enable local actions as follows:

```
agsysdb -actions always
```

The following table lists the settings for sending information to Operations Manager (OM) and for executing local actions:

Table 10: Settings for sending information to OM and executing local actions

Local Actions Flag	Operations Monitoring Component Running	Operations Monitoring Component Not Running
off	No local actions executed.	No local actions executed.
always	Local actions executed even if the Operations Monitoring Component is running.	Local actions executed.
on	Local actions sent to OM.	Local actions executed.

Errors in Processing Alarms

The last error that occurred when sending an alarm is logged in agdb. To view the contents of agdb, type:

```
agsysdb -l
```

The following information appears:

PA alarming status:

OVO messages : on Last Error : none

Exec Actions : on

Analysis system: <hostname>, Key=<ip address>

PerfView : no Last Error : <error number>

SNMP : yes Last Error : <error number>

Analyzing Historical Data for Alarms

You can use the utility program's `analyze` command to find alarm conditions in datastore (see [Chapter 5, Utility Commands](#)). This is different from the processing of real-time alarms explained earlier because you are comparing historical data in the datastore to the alarm definitions in the `alarmdef` file to determine what alarm conditions would have been triggered.

Examples of Alarm Information in Historical Data

The following examples show what is reported when you analyze alarm conditions in historical data.

For the first example, `START`, `END`, and `REPEAT` statements have been defined in the alarm definition. An alarm-start event is listed every time an alarm has met all of its conditions for the specified duration. When these conditions are no longer satisfied, an alarm-end event is listed. A repeat event is listed, if an alarm condition is satisfied for a period long enough to generate another alarm before the first alarm comes to an end.

Each event listed shows the date and time, alarm number, and the alarm event. EXEC actions are *not* performed, but they are listed with any requested parameter substitutions in place.

```
05/10/99 11:15 ALARM [1] START  
CRITICAL: CPU test 99.97%
```

```
05/10/99 11:20 ALARM [1] REPEAT
WARNING: CPU test 99.997%
```

```
05/10/99 11:25 ALARM [1] END
RESET: CPU test 22.86%
EXEC: end.script
```

If you are using a color workstation, the following output is highlighted:

CRITICAL statements are RED

MAJOR statements are MAGENTA

MINOR statements are YELLOW

WARNING statements are CYAN

NORMAL statements are GREEN

The next example shows an alarm summary that is displayed after alarm events are listed. The first column lists the alarm number, the second column lists the number of times the alarm condition occurred, and the third column lists the total duration of the alarm condition.

Performance Alarm Summary:

Alarm	Count	Minutes
1	574	2865
2	0	0

Analysis coverage using "alarmdef":

Start: 05/04/99 08:00 Stop: 05/06/99 23:59

Total time analyzed: Days: 2 Hours: 15 Minutes: 59

Alarm Definition Components

An alarm occurs when one or more conditions you define continues over a specified duration. The alarm definition can include an action to be performed at the start or end of the alarm.

A condition is a comparison between two or more items. The compared items can be metric names, constants, or variables. For example:

```
ALARM gbl_cpu_total_util > 95 FOR 5 MINUTES
```

An action can be specified to be performed when the alarm starts, ends, or repeats. The action can be one of the following:

- ALERT - Sends a message to OM or an SNMP trap to NNM
- EXEC - Executes an operating system command
- PRINT - Sends a message to stdout when processed using the utility program.

For example:

```
ALARM gbl_swap_space_util > 95 FOR 5 MINUTES
  START
    RED ALERT "Global swap space is nearly full"
  END
  RESET ALERT "End of global swap space full condition"
```

You can create more complex actions using Boolean logic, loops through multiple-instance data such as applications and variables. (For more information, see [Alarm Syntax Reference](#)).

You can also use the INCLUDE statement to identify additional alarm definition files that you want to use. For example, you may want to break up your alarm definitions into smaller files.

Alarm Syntax Reference

This section describes the statements that are available in the alarm syntax. You may want to look at the `alarmdef` file for examples of how the syntax is used to create useful alarm definitions.

Alarm Syntax

```
ALARM condition [[AND,OR]condition]
  FOR duration [SECONDS, MINUTES]

  [TYPE="string"]
  [SERVICE="string"]
  [SEVERITY=integer]
  [START action]
  [REPEAT EVERY duration [SECONDS, MINUTES] action]
  [END action]

[RED, CRITICAL, ORANGE, MAJOR, YELLOW, MINOR, CYAN, WARNING,
  GREEN, NORMAL, RESET] ALERT message

EXEC "UNIX command"

PRINT message
IF condition
  THEN action
  [ELSE action]
```

```
{APPLICATION, PROCESS, DISK, LVOLUME, TRANSACTION, NETIF, CPU,  
  FILESYSTEM} LOOP action  
  
INCLUDE "filename"  
  
USE "data source name"  
  
[VAR] name = value  
  
ALIAS name = "replaced-name"  
  
SYMPTOM variable [ TYPE = {CPU, DISK, MEMORY, NETWORK}]  
  RULE condition PROB probability  
  [RULE condition PROB probability]  
  .  
  .
```

Syntax Conventions

- Braces ({ }) indicate that one of the choices is required.
- Brackets ([]) indicate an optional item.
- Items separated by commas within brackets or braces are options. Choose only one.
- Italics indicate a variable name that you replace.
- All syntax keywords are in uppercase.

Common Elements

The following elements are used in several statements in the alarm syntax and are described below.

- [Comments](#)
- [Compound Statement](#)
- [Conditions](#)
- [Constants](#)
- [Expressions](#)
- [Metric names](#)
- [Messages](#)

Metric Names

When you specify a metric name in an alarm definition, the current value of the metric is substituted. Metric names must be typed exactly as they appear in the metric definition, except for case sensitivity. Metrics definitions can be found in the *Performance Collection Component Dictionary of Operating Systems Performance Metrics*.

It is recommended that you use fully-qualified metric names if the metrics are from a data source other than the SCOPE data source (such as DSI metrics).

The format for specifying a fully qualified metric is:

```
data_source:instance(class):metric_name
```

A global metric in the SCOPE data source requires no qualification. For example:

```
metric_1
```

An application metric, which is available for each application defined in the SCOPE data source, requires the application name. For example,

```
application_1:metric_1
```

For multi-instance data types such as application, process, disk, netif, transaction, lvolume, cpu, and filesystem, you must associate the metric with the data type name, except when using the LOOP statement. To do this, specify the data type name followed by a colon, and then the metric name. For example, other_apps:app_cpu_total_util specifies the total CPU utilization for the application other_apps.

Note: When specifying fully qualified multi-instance metrics and using aliases within aliases, if one of the aliases has a class identifier, we recommend you use the syntax shown in this example:

```
alias my_fs="/dev/vg01/lvol1(LVOLUME)"alarm my_fs:LV_SPACE_UTIL > 50 for 5
minutes
```

If you use an application name that has an embedded space, you must replace the space with an underscore (_). For example, application 1 must be changed to application_1. For more information on using names that contain special characters, or names where case is significant, see [ALIAS Statement](#).

If you had a disk named "other" and an application named "other", you would need to specify the class as well as the instance:

```
other (disk):metric_1
```

A global metric in an extracted log file (where `scope_extract` is the data source name) would be specified this way:

```
scope_extract:application_1:metric_1
```

A DSI metric would be specified this way:

```
dsi_data_source:dsi_class:metric_name
```

Note: Any metric names containing special characters (such as asterisks) must be aliased before they are specified.

Messages

A message is the information sent by a PRINT or ALERT statement. It can consist of any combination of quoted alphanumeric strings, numeric constants, expressions, and variables. The elements in the message are separated by commas. For example:

```
RED ALERT "cpu utilization=", gbl_cpu_total_util
```

Numeric constants, metrics, and expressions can be formatted for width and number of decimals.

Width specifies how wide the field should be formatted; *decimals* specifies how many decimal places to use. Numeric values are right-justified. The - (minus sign) specifies left-justification. Alphanumeric strings are always left-justified. For example:

```
metric names [|[-]width[|decimals]]

gbl_cpu_total_util|6|2   formats as '100.00'
(100.32 + 20)|6         formats as '  120'
gbl_cpu_total_util|-6|0  formats as '100  '
gbl_cpu_total_util|10|2  formats as '  99.13'
gbl_cpu_total_util|10|4  formats as ' 99.1300'
```

ALARM Statement

The ALARM statement defines a condition or set of conditions and a duration for the conditions to be true. Within the ALARM statement, you can define actions to be performed when the alarm condition starts, repeats, and ends. Conditions or events that you can define as alarms include:

- Global swap space has been nearly full for 5 minutes
- Memory paging rate has been too high for 1 interval
- CPU has been running at 75 percent utilization for the last ten minutes

Syntax

```
ALARM condition [[AND,OR]condition]
  FOR duration{SECONDS, MINUTES}
  [TYPE="string"]
  [SERVICE="string"]
  [SEVERITY=integer]
  [START action]
  [REPEAT EVERY duration {SECONDS, MINUTES} action]
  [END action]
```

In this instance:

- The ALARM statement must be a top-level statement. It cannot be nested within any other statement. However, you can include several ALARM conditions in a single ALARM statement. If the conditions are linked by AND, all conditions must be true to trigger the alarm. If they are linked by OR, any one condition will trigger the alarm.
- TYPE is a quoted string of up to 38 characters. If you are sending alarms, you can use TYPE to categorize alarms and to specify the name of a graph template to use.
- SERVICE is a quoted string of up to 200 characters. If you are using ServiceNavigator, you can link your Performance Collection Component alarms with the services you defined in ServiceNavigator. For more information see, the *Operations ServiceNavigator Concepts and Configuration Guide*.

```
SERVICE="Service_id"
```

- SEVERITY is an integer from 0 to 32767.
- START, REPEAT, and END are *keywords* used to specify what action to take when alarm conditions are met, met again, or stop. You should always have at least one of START, REPEAT, or END in an ALARM statement. Each of these keywords is followed by an *action*.
- action is most often used with an ALARM START, REPEAT, or END is the ALERT statement. However, you can also use the EXEC statement to mail a message or run a batch file, or a PRINT statement if you are analyzing historical log files with the utility program. Any syntax statement is legal except another ALARM.

START, REPEAT, and END actions can be compound statements. For example, you can use compound statements to provide both an ALERT and an EXEC.

- Conditions is defined as a comparison between two items.

```
item1 {>, <, >=, <=, ==, !=}item2
```

```
[AND, OR[item3 {>, <, >=, <=, ==, !=}item4]]
```

where "==" means "equal", and "!=" means "not equal"

An item can be a metric name, a numeric constant, an alphanumeric string enclosed in quotes, an alias, or a variable. When comparing alphanumeric items, only == or != can be used as operators.

You can use compound conditions by specifying the “OR” and “AND” operator between subconditions.

For example:

```
ALARM gbl_cpu_total_util > 90 AND
gbl_pri_queue > 1 for 5 minutes
```

- You also can use compound conditions *without* specifying the “OR” and “AND” operator between subconditions. For example:

```
ALARM gbl_cpu_total_util > 90
gbl_cpu_sys_mode_util > 50 for 5 minutes
```

An alarm is generated when both conditions are **True**.

FOR specifies the time duration in SECONDS or MINUTES during which the condition must remain **True** to trigger an alarm.

Use caution when specifying durations of less than one minute, particularly when there are multiple data sources on the system. Performance can be seriously degraded if each data source must be polled for data at very small intervals. The duration must be a multiple of the longest collection interval of the metrics mentioned in the alarm condition.

Do not set a duration that is shorter than the collection interval. Since the metric value does not change until the next collection cycle is complete, setting a duration shorter than the collection interval means unnecessary processing for generating duplicate alarms.

In the parm file, the following parameters under the `collectioninterval` parameter control the collection interval:

- `global`: This parameter indicates the collection interval (in seconds) for all metrics that are logged into the `oacore` and `dsi` database files except for the process metrics. By default, this is set to 5 minutes.
- `process`: This parameter indicates the collection interval (in seconds) for process metrics.

For more information about the `collectioninterval` parameter, see [Configure Data Logging Intervals](#).

- `REPEAT EVERYnSECONDS, MINUTES` specifies the time period before the alarm is repeated.

In this instance, **n** refers to duration. For example, `REPEAT EVERY5SECONDS, MINUTES`

How it is Used

The alarm cycle begins on the first interval when all of the **AND**, or one of the **OR** alarm conditions have been **True** for at least the specified duration. At that time, the alarm generator executes the **START action**, and on each subsequent interval checks the **REPEAT** condition. If enough time has transpired, the **action** for the **REPEAT** clause is executed. This continues until one or more of the alarm conditions becomes *False*. This completes the alarm cycle and the **END** statement is executed if there is one.

To be notified of the alarm, use the **ALERT** statement within the **START** and **END** statements. If you do not specify an **END ALERT**, the alarm generator automatically sends an alarm to OM and sends an SNMP trap to NNM. [VAR Statement](#)

Examples

The following **ALARM** example sends a red alert when the swap utilization is high for 5 minutes. It is similar to an alarm condition in the default `alarmdef` file. Do not add this example to your `alarmdef` file without removing the default alarm condition, or your subsequent alert messages may be confusing.

```
ALARM gbl_swap_space_util > 90 FOR 5 MINUTES
  START
    RED ALERT "swap utilization is very high "
  REPEAT EVERY 15 MINUTES
    RED ALERT "swap utilization is still very high "
  END
  RESET ALERT "End of swap utilization condition"
```

This **ALARM** example tests the metric `gbl_swap_space_util` to see if it is greater than 90. Depending on how you configured the alarm generator, the **ALERT** can be sent to NNM via an SNMP trap or as a message to Operations Manager.

The **REPEAT** statement checks for the `gbl_swap_space_util` condition every 15 minutes. As long as the metric remains greater than 90, the **REPEAT** statement will send the message “swap utilization is still very high” every 15 minutes.

When the `gbl_swap_space_util` condition goes below 90, the **RESET ALERT** statement with the “End of swap utilization condition” message is sent.

The following example defines a compound action within the **ALARM** statement. This example shows you how to mail a message when an event occurs.

```
ALARM gbl_cpu_total_util > 90 FOR 5 MINUTES
  START
  {
    RED ALERT "Your CPU is busy."
```

```

EXEC "echo 'cpu is too high' | mailx root"
}
END
  RESET ALERT "CPU no longer busy."

```

The ALERT can trigger an SNMP trap to be sent to NNM or a message to be sent to OM. The EXEC can trigger a mail message to be sent as a local action on your node, depending on how you configured your alarm generator.

The following two examples show the use of multiple conditions. You can have more than one test condition in the ALARM statement. In this case, each statement must be true for the ALERT to be sent.

The following ALARM example tests the metric `gbl_cpu_total_util` and the metric `gbl_cpu_sys_mode_util`. If both conditions are true, the RED ALERT statement sends a red alert. When either test condition becomes false, the RESET alert is sent.

```

ALARM gbl_cpu_total_util > 85
  AND gbl_cpu_sys_mode_util > 50 FOR 5 MINUTES
START
  RED ALERT "CPU busy and Sys Mode CPU util is high."
END
  RESET ALERT "The CPU alert is now over."

```

The next ALARM example tests the metric `gbl_cpu_total_util` and the metric `gbl_cpu_sys_mode_util`. If either condition is true, the RED ALERT statement sends a red alert.

```

ALARM gbl_cpu_total_util > 85
  OR
  gbl_cpu_sys_mode_util > 50 FOR 10 MINUTES
START
  RED ALERT "Either total CPU util or sys mode CPU high"

```

Do not use metrics that are logged at different intervals in the same alarm. For example, you should not loop on a process (logged at 1-minute intervals) based on the value of a global metric (logged at 5-minute intervals) in a statement like this:

```
IF gbl_cpu_total_util > 85 THEN  PROCESS LOOP...
```

The different intervals cannot be synchronized as you might expect, so results will not be valid.

Note: For GlancePlus, use the process metrics inside a process loop to send alarm for all the processes.

ALERT Statement

The ALERT statement allows a message to be sent to the Network Node Manager or Operations Manager. The ALERT statement is most often used as an action within an ALARM. It could also be used within an IF statement to send a message as soon as a condition is detected instead of waiting till the duration is complete. If an ALERT is used outside of an ALARM or IF statement, the message will be sent at every interval.

Syntax

```
[RED, CRITICAL, ORANGE, MAJOR, YELLOW, MINOR, CYAN, WARNING, GREEN, NORMAL, RESET]  
ALERT message
```

- RED is synonymous with CRITICAL, ORANGE is synonymous with MAJOR, YELLOW is synonymous with MINOR, CYAN is synonymous with WARNING, and GREEN is synonymous with NORMAL. These keywords turn the alarm symbol to the color associated with the alarm condition.
- RESET — Sends a RESET ALERT with a message when the ALARM condition ends. If you have not defined a reset in the alarm definition, sends a RESET ALERT without a message when the ALARM condition ends.
- *message* — A combination of strings and numeric values used to create a message. Numeric values can be formatted with the parameters [| [-] *width* [| *decimals*]]. *Width* specifies how wide the field should be formatted; *decimals* specifies how many decimal places to use. Numeric values are right-justified. The - (minus sign) specifies left-justification. Alphanumeric strings are always left-justified.

How it is Used

The ALERT can also trigger an SNMP trap to be sent to NNM or a message to be sent to OM, depending on how you configured your alarm generator. For alert messages sent to OM, the WARNINGS appear in blue in the message browser.

Example

A typical ALERT statement is:

```
RED ALERT "CPU utilization = ", gbl_cpu_total_util
```

If you have the Network Node Manager, this statement creates a critical severity alarm in the Alarm Browser window in Network Node Manager.

EXEC Statement

The EXEC statement allows you to specify a system (UNIX or Windows) command to be executed on the local system. For example, you could use the EXEC statement to send mail to an IT administrator each time a certain condition is met.

EXEC should be used within an ALARM or IF statement so the command is executed only when specified conditions are met. If an EXEC statement is used outside of an ALARM or IF statement, the action will be performed at unpredictable intervals.

Syntax

```
EXEC "system command"
```

system command — Is a command to be executed on the local system.

Do not use embedded double quotes (") in EXEC statements. Doing so causes `per-falarm` to fail to send the alarm to OM. Use embedded single (') quotes instead. For example:

```
EXEC "echo 'performance problem detected' "
```

```
EXEC "mkdir c:\\directory\\filename"
```

The syntax of the EXEC statement requires the path name of the file to be enclosed within double quotes. However, if a path name contains spaces, the path name must be enclosed within single quotes, which must be again enclosed within double quotes.

Example:

```
EXEC "'C:\\Program Files\\Mail Program\\SendMail.exe'"
```

If any arguments to the system command of the EXEC statement contains single quotes, the program must be enclosed within single quotes as the first pair of single quotes (') are converted into double quotes (") while you run the command with the EXEC statement.

Example:

```
EXEC "'echo' 'test execution'"
```

In the above example, `echo` is the program enclosed within single quotes as it contains an argument (in this case, `test execution`) with single quotes. Furthermore, as per the syntax of the EXEC statement, the entire string of the command must be enclosed in double quotes.

Do not use embedded double quotes (") in EXEC statements; `perfalarm` will fail to send the alarm to the OM. Use embedded single quotes (') instead.

For example:

```
EXEC "'echo' 'dialog performance problem'"
```

In the above example, `echo` is the program enclosed within single quotes as it contains an argument (in this case, `dialog performance problem`) with single quotes. Further, as per the syntax of the EXEC statement, the entire string of the command must be enclosed in double quotes.

How it is Used

The EXEC can trigger a local action on your local system, depending on how you configured your alarm generator. For example, you can turn local actions on or off. If you configured your alarm generator to send information to OM local actions will not usually be performed.

Examples

In the following example, the EXEC statement performs the UNIX `mailx` command when the `gbl_disk_util_peak` metric exceeds 20.

```
IF gbl_disk_util_peak > 20 THEN
    EXEC "echo 'high disk utilization detected' | mailx root"
```

The next example shows the EXEC statement sending mail to the system administrator when the network packet rate exceeds 1000 per second average for 15 minutes.

```
ALARM gbl_net_packet_rate > 1000 for 15 minutes
    TYPE = "net busy"
    SEVERITY = 5
    START
    {
        RED ALERT "network is busy"
        EXEC "echo 'network busy condition detected' | mailx root"
    }
    END
    RESET ALERT "NETWORK OK"
```

Note: Be careful when using the EXEC statement with commands or scripts that have high

overhead if it is performed often.

The alarm generator executes the command and waits until it completes before continuing. We recommend that you not specify commands that take a long time to complete.

PRINT Statement

The PRINT statement allows you to print a message from the utility program using its `analyze` function. The alarm generator ignores the PRINT statement.

For more information, see ["PRINT Statement" on page 98](#)

IF Statement

Use the IF statement to define a condition using IF-THEN logic. The IF statement should be used within the ALARM statement. However, it can be used by itself or any place in the `alarmdef` file where IF-THEN logic is needed.

If you specify an IF statement outside of an ALARM statement, you do not have control over how frequently it gets executed.

Syntax

```
IF condition THEN action [ELSE action]
```

Condition — A condition is defined as a comparison between two items.

```
item1 {>, <, >=, <=, ==, !=}item2  
  [AND, OR[item3 {>, <, >=, <=, ==, !=}item4]]
```

where "==" means "equal", and "!=" means "not equal".

An item can be a metric name, a numeric constant, an alphanumeric string enclosed in quotes, an alias, or a variable. When comparing alphanumeric strings, only == or != can be used as operators.

action — Any action, or set a variable. (ALARM is not valid in this case.)

How it is Used

The IF statement tests the *condition*. If the *condition* is true, the *action* after the THEN is executed. If the *condition* is false, the *action* depends on the optional ELSE clause. If an ELSE clause has been specified, the *action* following it is executed; otherwise the IF statement does nothing.

Example

In this example, a CPU bottleneck symptom is calculated and the resulting bottleneck probability is used to define cyan or red ALERTs. Depending on how you configured your alarm generator, the ALERT triggers an SNMP trap to NNM or the message "End of CPU Bottleneck Alert" to Operations Manager along with the percentage of CPU used.

```

SYMPTOM CPU_Bottleneck > type=CPU
  RULE gbl_cpu_total_util > 75 prob 25
  RULE gbl_cpu_total_util > 85 prob 25
  RULE gbl_cpu_total_util > 90 prob 25
  RULE gbl_cpu_total_util > 4 prob 25

ALARM CPU_Bottleneck > 50 for 5 minutes
  TYPE="CPU"
  START
    IF CPU_Bottleneck > 90 then
      RED ALERT "CPU Bottleneck probability= ",
        CPU_Bottleneck, "%"
    ELSE
      CYAN ALERT "CPU Bottleneck probability= ",
        CPU_Bottleneck, "%"
  REPEAT every 10 minutes
    IF CPU_Bottleneck > 90 then
      RED ALERT "CPU Bottleneck probability= ",
        CPU_Bottleneck, "%"
    ELSE
      CYAN ALERT "CPU Bottleneck probability= ",
        CPU_Bottleneck, "%"
  END
  RESET ALERT "End of CPU Bottleneck Alert"

```

Do not use metrics that are logged at different intervals in the same statement. For instance, you should not loop on a process (logged at 1-minute intervals) based on the value of a global metric (logged at 5-minute intervals) in a statement like this:

```
IF gbl_cpu_total_util > 85 THEN PROCESS LOOP ...
```

The different intervals cannot be synchronized as you might expect, so results will not be valid.

LOOP Statement

The LOOP statement goes through multiple-instance data types and performs the *action* defined for each instance.

Syntax

```
{APPLICATION, PROCESS, LVOLUME, DISK, CPU, FILESYSTEM, TRANSACTION, NETIF, LOGICAL}  
LOOP
```

action

- APPLICATION, PROCESS, LVOLUME, DISK, CPU, FILESYSTEM, TRANSACTION, NETIF, LOGICAL — Performance Collection Component data types that contain multi-instance data.
- *action* — PRINT, EXEC, ALERT, set variables.

How it is Used

As LOOP statements iterate through each instance of the data type, metric values change. For instance, the following LOOP statement prints the name of each application to `stdout` if you are using the utility program's `analyze` command.

```
APPLICATION LOOP  
  
PRINT app_name
```

A LOOP can be nested within another LOOP statement up to a maximum of five levels.

In order for the LOOP to execute, the LOOP statement must refer to one or more metrics of the same data type as the type defined in the LOOP statement.

Example

You can use the LOOP statement to cycle through all active applications.

The following example shows how to determine which application has the highest CPU at each interval.

```
highest_cpu = 0  
APPLICATION loop
```



```
IF app_cpu_total_util > highest_cpu THEN
{
  highest_cpu = app_cpu_total_util
  big_app = app_name
}

ALERT "Application ", app_name, " has the highest cpu util at ",highest_cpu_
util|5|2, "%"

ALARM highest_cpu > 50
START
  RED ALERT big_app, " is the highest CPU user at ", highest_cpu, "%"
  REPEAT EVERY 15 minutes
  CYAN ALERT big_app, " is the highest CPU user at ", highest_cpu, "%"
END
RESET ALERT "No applications using excessive cpu"
```

INCLUDE Statement

Use the INCLUDE statement to include another alarm definitions file along with the default alarmdef file.

Syntax

```
INCLUDE "filename"
```

where *filename* is the name of another alarm definitions file. The file name must always be fully qualified.

How it is Used

The INCLUDE statement could be used to separate logically distinct sets of alarm definitions into separate files.

Example

For example, if you have some alarm definitions in a separate file for your transaction metrics and it is named

```
trans_alarmdef1
```

You can include it by adding the following line to the alarm definitions in your alarmdef file:

```
INCLUDE "/var/opt/perf/trans_alarmdef1"
```

USE Statement

You can add the `USE` statement to simplify the use of metric names in the `alarmdef` file when data sources other than the default `SCOPE` data source are referenced. This allows you to specify a metric name without having to include the data source name.

The data source name must be defined in the `datasources` file. The `alarmdef` file will fail its syntax check if an invalid or unavailable data source name is encountered.

Note: The appearance of a `USE` statement in the `alarmdef` file does not imply that all metric names that follow will be from the specified data source.

Syntax

```
USE "datasourcename"
```

How it is Used

As the alarm generator checks the `alarmdef` file for valid syntax, it builds an ordered search list of all data sources that are referenced in the file. `perfalarm` sequentially adds entries to this data source search list as it encounters fully-qualified metric names or `USE` statements. This list is subsequently used to match metric names that are not fully qualified with the appropriate data source name. The `USE` statement provides a convenient way to add data sources to `perfalarm`'s search list, which then allows for shortened metric names in the `alarmdef` file. For a discussion of metric name syntax, see [Metric Names](#) earlier in this chapter.

The default behavior of `perfalarm` for matching metric names to a data source is to look first in the `SCOPE` data source for the metric name. This implied `USE "SCOPE"` statement is executed when `perfalarm` encounters the first metric name in the `alarmdef` file. This feature enables a default search path to the `SCOPE` data source so that `SCOPE` metrics can be referenced in the `alarmdef` file without the need to fully qualify them. This is shown in the following example on the next page.

```
ALARM gbl_cpu_total_util > 80 FOR 10 MINUTES
    START RED ALERT "CPU utilization too high"

USE "ORACLE7"

ALARM ActiveTransactions >= 95 FOR 5 MINUTES
    START RED ALERT "Nearing limit of transactions for ORACLE7"
```

When `perfalarm` checks the syntax of the `alarmdef` file containing the above statements, it encounters the metric `"gbl_cpu_total_util"` and then tries to find its data source. `Perfalarm` does not yet have any data sources in its search list of data sources, so it executes an implied `USE "SCOPE"` statement and then searches the `SCOPE` data source to find the metric name. A match is found and `perfalarm` continues checking the rest of the `alarmdef` file.

When `perfalarm` encounters the `USE "ORACLE7"` statement, it adds the `ORACLE7` data source to the search list of data sources. When the `"ActiveTransactions"` metric name is encountered, `perfalarm` sequentially searches the list of data sources starting with the `SCOPE` data source. `SCOPE` does not contain that metric name, so the `ORACLE7` data source is searched next and a match is found.

If `perfalarm` does not find a match in any data source for a metric name, an error message is printed and `perfalarm` terminates.

To change the default search behavior, a `USE` statement can be added to the beginning of the `alarmdef` file before any references to metric names. This will cause the data source specified in the `USE` statement to be added to the search list of data sources before the `SCOPE` data source. The data source (s) in the `USE` statement(s) will be searched before the `SCOPE` data source to find matches to the metrics names. This is shown in the following example.

Once a data source has been referenced with a `USE` statement, there is no way to change its order or to remove it from the search list.

```
USE "ORACLE7"

ALARM gbl_cpu_total_util > 80 FOR 10 MINUTES
      START RED ALERT "CPU utilization too high"

ALARM ActiveTransactions >= 95 FOR 5 MINUTES
      START RED ALERT "Nearing limit of
      transactions for ORACLE7"
```

In the example above, the order of the statements in the `alarmdef` file has changed. The `USE "ORACLE7"` statement is defined before any metric names are referenced, therefore the `ORACLE7` data source is added as the first data source in the search list of data sources. The implied `USE "SCOPE"` statement is executed when `perfalarm` encounters the first metric name `"gbl_cpu_total_util."` Because the `"gbl_cpu_total_util"` metric name is not fully-qualified, `perfalarm` sequentially searches through the list of data sources starting with `ORACLE7`. `ORACLE7` does not contain that metric name so the `SCOPE` data source is searched next and a match is found.

`perfalarm` continues checking the rest of the `alarmdef` file. When `perfalarm` encounters the `"ActiveTransactions"` metric, it sequentially searches the list of data sources starting with `ORACLE7`. A match is found and `perfalarm` continues searching the rest of the `alarmdef` file. If `perfalarm` does not find a match in any data source for a metric name (that is not fully-qualified), an error message will be printed and `perfalarm` terminates.

Be careful how you use the `USE` statement when multiple data sources contain the same metric names. `perfalarm` sequentially searches the list of data sources. If you are defining alarm conditions from different data sources that use the same metric names, you must qualify the metric names with their data source names to guarantee that the metric value is retrieved from the correct data source. This is shown in the following example where the metric names in the alarm statements include their data sources.

```
ALARM ORACLE7:ActiveTransactions >= 95 FOR 5 MINUTES
  START RED ALERT "Nearing limit of transactions for ORACLE7"
```

```
ALARM FINANCE:ActiveTransactions >= 95 FOR 5 MINUTES
  START RED ALERT "Nearing limit of transactions for FINANCE"
```

VAR Statement

The `VAR` statement allows you to define a variable and assign a value to it.

Syntax

```
[VAR] name = value
```

name - Variable names must begin with a letter and can include letters, digits, and the underscore character. Variable names are not case-sensitive.

value - If the value is an alphanumeric string, it must be enclosed in quotes.

How it is Used

`VAR` assigns a value to the user variable. If the variable did not previously exist, it is created.

Once defined, variables can be used anywhere in the `alarmdef` file.

Examples

You can define a variable by assigning something to it. The following example defines the numeric variable `highest_CPU_value` by assigning it a value of zero.

```
highest_CPU_value = 0
```

The next example defines the alphanumeric variable `my_name` by assigning it an empty string value.

```
my_name = ""
```

ALIAS Statement

The ALIAS statement allows you to substitute an alias if any part of a metric name (class, instance, or metric) has a case-sensitive name or a name that includes special characters. These are the only circumstances where the ALIAS statement should be used.

Syntax

```
ALIAS name = "replaced-name"
```

- *name* — The name must begin with a letter and can include letters, digits, and the underscore character.
- *replaced-name* — The name that must be replaced by the ALIAS statement to make it uniquely recognizable to the alarm generator.

How it is Used

Because of the way the `alarmdef` file is processed, if any part of a metric name (class, instance, or metric name) can be identified uniquely only by recognizing uppercase and lowercase, you will need to create an alias. You will also need to create an alias for any name that includes special characters. For example, if you have applications called "BIG" and "big," you'll need to alias "big" to ensure that they are viewed as different applications. You must define the alias somewhere in the `alarmdef` file before the *first* instance of the name you want substituted.

Examples

Because you cannot use special characters or upper and lower case in the syntax, using the application name "AppA" and "appa" could cause errors because the processing would be unable to distinguish between the two. You would alias "AppA" to give it a uniquely recognizable name. For example:

```
ALIAS appa_uc = "AppA"  
ALERT "CPU alert for AppA.util is", appa_uc: app_cpu_total_util
```

If you are using an alias for an instance with a class identifier, include both the instance name and the class name in the alias. The following example shows the alias for the instance name 'other' and the class name 'APPLICATION.'

```
ALIAS my_app="other(APPLICATION)"
ALERT my_app:app_cpu_total_util > 50 for 5 minutes
```

SYMPTOM Statement

A SYMPTOM provides a way to set a single variable value based on a set of conditions. Whenever any of the conditions is **True**, its probability value is added to the value of the SYMPTOM variable.

For more information see, ["SYMPTOM Statement " on page 98](#)

Alarm Definition Examples

The following examples show typical uses of alarm definitions.

Example of a CPU Problem

Depending on how you configured the alarm generator, this example triggers an SNMP trap to Network Node Manager or a message to Operations Manager whenever CPU utilization exceeds 90 percent for 5 minutes and the CPU run queue exceeds 3 for 5 minutes.

```
ALARM gbl_cpu_total_util > 90 AND
  gbl_run_queue > 3 FOR 5 MINUTES
START
  CYAN ALERT "CPU too high at", gbl_cpu_total_util, "%"
REPEAT EVERY 20 MINUTES
{
  RED ALERT "CPU still to high at ", gbl_cpu_total_util, "%"
  EXEC "/usr/bin/pager -n 555-3456"
}
END
  RESET ALERT "CPU at ", gbl_cpu_total_util, "% - RELAX"
```

If both conditions continue to hold true after 20 minutes, a critical severity alarm can be created in NNM. A program is then run to page the system administrator.

When either one of the alarm conditions fails to be true, the alarm symbol is deleted and a message is sent showing the global CPU utilization, the time the alert ended, and a note to RELAX.

Example of Swap Utilization

In this example, depending on how you configured the alarm generator, the ALERT can trigger an SNMP trap to be sent to NNM or a message to be sent to OM, whenever swap space utilization exceeds 95 percent for 5 minutes.

```
ALARM gbl_swap_space_util > 95 FOR 5 MINUTES
START
  RED ALERT "GLOBAL SWAP space is nearly full "
END
  RESET ALERT "End of GLOBAL SWAP full condition"
```

Example of Time-Based Alarms

You can specify a time interval during which alarm conditions can be active. For example, if you are running system maintenance jobs that are scheduled to run at regular intervals, you can specify alarm conditions for normal operating hours and a different set of alarm conditions for system maintenance hours.

In this example, the alarm will only be triggered during the day from 8:00AM to 5:00PM.

```
start_shift = "08:00"
end_shift = "17:00"

ALARM gbl_cpu_total_util > 80
  TIME > start_shift
  TIME < end_shift for 10 minutes
  TYPE = "cpu"
START
  CYAN ALERT "cpu too high at ", gbl_cpu_total_util, "%"
REPEAT EVERY 10 minutes
  RED ALERT "cpu still too high at ", gbl_cpu_total_util, "%"
END
  IF time == end_shift then
  {
  IF gbl_cpu_total_util > 80 then
    RESET ALERT "cpu still too high, but at the end of shift"
  ELSE
    RESET ALERT "cpu back to normal"
  ELSE
```

Example of Disk Instance Alarms

Alarms can be generated for a particular disk by identifying the specific disk instance name and corresponding metric name.

The following example of alarm syntax generates alarms for a specific disk instance. Aliasing is required when special characters are used in the disk instance.

```
ALIAS diskname=""
ALARM diskname:bydisk_phys_read > 1000 for 5 minutes
TYPE="Disk"
START
  RED ALERT "Disk  "
REPEAT EVERY 10 MINUTES
  CYAN ALERT "Disk cyan alert"
END
  RESET ALERT "Disk reset alert"
```

Customizing Alarm Definitions

You must specify the conditions that generate alarms in the alarm definitions file `alarmdef`. When Performance Collection Component is first installed, the `alarmdef` file contains a set of default alarm definitions. You can use these default alarm definitions or customize them to suit your needs.

You can customize your `alarmdef` file as follows:

1. Revise your alarm definition(s) as necessary. You can look at examples of the alarm definition syntax elsewhere in this chapter.
2. Save the file.
3. Validate the alarm definitions using the Performance Collection Component utility program:
 - a. Type `utility`
 - b. At the prompt, type

```
checkdef
```

This checks the alarm syntax and displays errors or warnings if there any problems with the file.

4. In order for the new alarm definitions to take effect, type:

```
ovpa restart alarm
```

This causes the alarm generator to stop, restart, and read the customized `alarmdef` file.

You can use a unique set of alarm definitions for each Performance Collection Component system, or you can choose to standardize monitoring of a group of systems by using the same set of alarm definitions across the group.

If the `alarmdef` file is very large, the Performance Collection Component alarm generator and utility programs may not work as expected. This problem may or may not occur based on the availability of system resources.

The best way to learn about performance alarms is to experiment with adding new alarm definitions or changing the default alarm definitions.

Chapter 4: Operations Agent in a Secure Environment

The OM, along with the Operations Agent, helps you monitor and manage systems and applications deployed in your network environment from a central console. In an OM-based management environment, you can start monitoring the systems of your interest after installing the Operations Agent on them. With the help of policies deployed from the OM console on the agent node, you can enable different monitoring capabilities of the agent.

Primary responsibilities of the Operations Agent in a distributed environment are:

- **Monitoring data**

The Operations Agent can compare the value of a specific metric with a preset value and take necessary actions based on its configuration. Policies, deployed from the OM console to the node, play a major role in facilitating the monitoring capability of the Operations Agent.

- **Collecting and storing data**

You can program the performance data collector of the Operations Agent to collect and log the data of your interest on the monitored system. You can add additional collection capabilities by installing SPIs and log the data collected by SPIs into agent datastore.

Policies

To work with the agent, you must deploy collections of configuration details and specifications called policies on the managed nodes from the OM console. Depending on the types of policies deployed, different components of the Operations Agent are enabled. A policy can provide the following details to the agent:

- **Monitoring source details**

- Objects to monitor
- Polling interval for monitoring an object
- Threshold value for the monitored object
- Rules and conditions for analysis of data against the set threshold

- **Event details**

You can configure the Operations Agent using policies to generate events with messages, instructions, and severity flags when a monitored object violates the threshold rule. The events are forwarded to the OM console in the form of messages. You can set the agent to perform certain action against these events.

- **Data collection details**

If you want to monitor the data collected by an external program, you can program the Operations Agent to log the data into its datastore.

HTTPS Mode of Communication

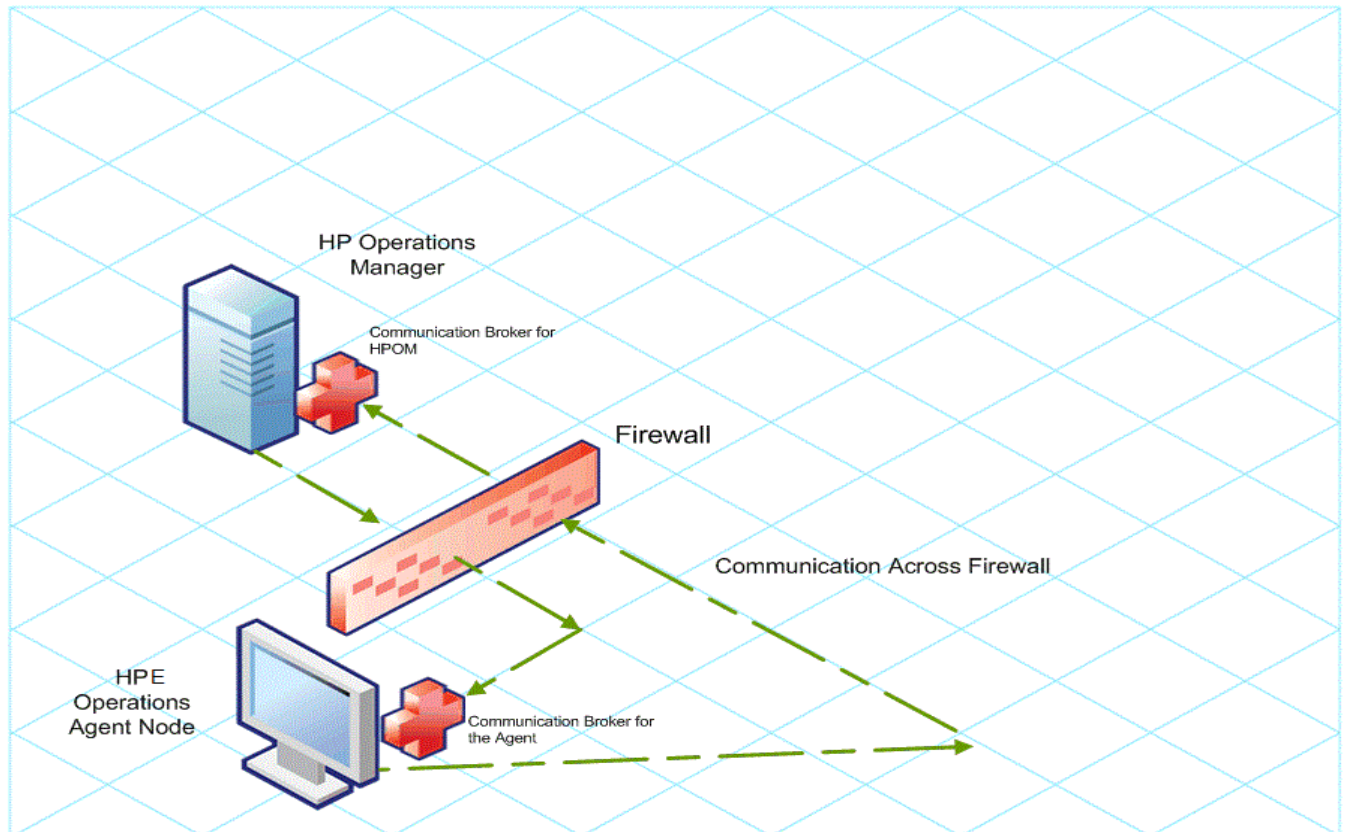
The Operations Agent nodes, by using the HTTPS mode of communication, can easily communicate with each other, as well as with other industry-standard products.

Benefits of the HTTPS Communication

- **Communication Over Firewalls**

With the help of the HTTPS protocol, the Operations Agent nodes can communicate with other systems available across firewalls. You can deploy the Operations Agent in a secure environment built with HTTP proxies and firewalls.

The following figure illustrates how to cross a firewall using HTTPS-communication.



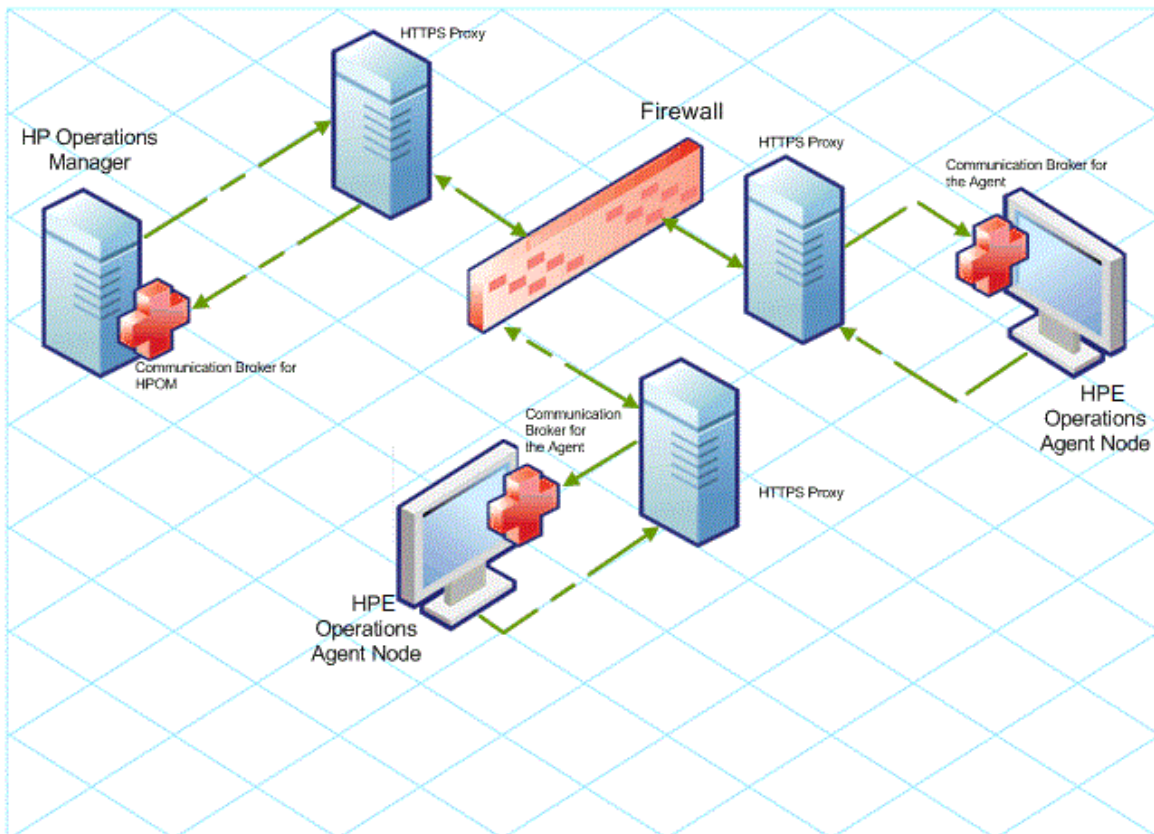
- **Advanced Security**

The Operations Agent product uses the Secure Socket Layer (SSL) to restrict and control user access. With the help of SSL, the Operations Agent product compresses and encrypts all the data involved in its communication with other systems.

In addition, all remote messages arrive through the **Communication Broker** component, providing a single port entry to the Operations Agent node.

From an Operations Agent node, if you want to send messages, files, or objects, you can configure one or more standard HTTP proxies to cross a firewall or reach a remote system.

The following figure illustrates how to cross a firewall using external HTTPS proxies:



- **Open Standards**

Operations Agent's HTTPS communication is built on the industry standard HTTP 1.1 protocol and SSL sockets. Operations Agent's adherence to open standards, such as HTTP, SSL, and SOAP, enables you to maximize the use of your current HTTP infrastructure.

- **Scalability**

Operations Agent's HTTPS communication is designed to perform well, independent of the size of the environment and the amount of data sent and received. Operations Agent's HTTPS communication can be configured to suit the requirement of your organization.

Communication Broker

The Communication Broker component provides a single-port solution for an Operations Agent node. In a typical deployment scenario, multiple servers can be registered with the Operations Agent node for data communication. The Operations Agent product directs the requests for all registered servers on the node through the Communication Broker. The Communication Broker transparently forwards the request to the registered server in the same way as an HTTP proxy forwards an HTTP request. The

default port for the Communication Broker is 383. You can configure the Operations Agent product to use a different port for the Communication Broker.

For higher security on UNIX systems, the Communication Broker starts up with `chroot`. `chroot` restricts the part of the file system visible to the Communication Broker process by making the specified path act as the root directory, therefore reducing exposure to unauthorized access.

The Communication Broker runs as a daemon on a UNIX system and as a service on a Windows system.

The Communication Broker uses a minimum of one port for accepting incoming data to a node. The port is associated with a unique node identifier (`OVCOREID`) to identify the node. You can configure the Communication Broker to use multiple ports for high availability nodes.

Firewall Scenarios

Firewalls can protect systems in a network from external attacks. They usually separate the Internet from a private Intranet. You can implement multiple levels of firewalls to restrict access to the more trusted environments from those of lower sensitivity.

A firewall separates a networked environment into two basic zones: the **trusted zone** and the **Demilitarized zone (DMZ)** (for example, the Internet). A firewall configuration ensures that data transmission from DMZ to the trusted zone is restricted or controlled. Based on the configuration, the firewall can allow a **two-way communication** or an **outbound-only communication**.

If you configure the firewall in your environment to allow a two-way communication, the network allows the HTTPS communication across the firewall in both directions with certain restrictions. You can configure the firewall settings in this environment to use the following configuration options:

- *Proxies*: If your network allows only certain proxy systems to connect through the firewall, you can redirect Operations Agent communication through these proxies.
- *Local ports*: If your network allows outbound connections from only certain local ports, you can configure Operations Agent to use specific local ports.
- *Communication broker ports*: If your network allows inbound connections to only certain destination ports, but not to port 383, you can configure alternate communication broker ports.

When the firewall in your environment allows outbound-only communication, you can configure a **Reverse Channel Proxy (RCP)** with the Operations Agent product. The RCP configured with the Operations Agent node works like an HTTP proxy and enables you to transfer data from DMZ to the trusted (secure) zone. Instead of directly communicating to HPE Software systems, RCPs establish a communication channel to the Communication Broker. The Communication Broker verifies and

authenticates the information originating from DMZ, and then transfers the validated information to the Operations Agent node present in the trusted (secure) zone.

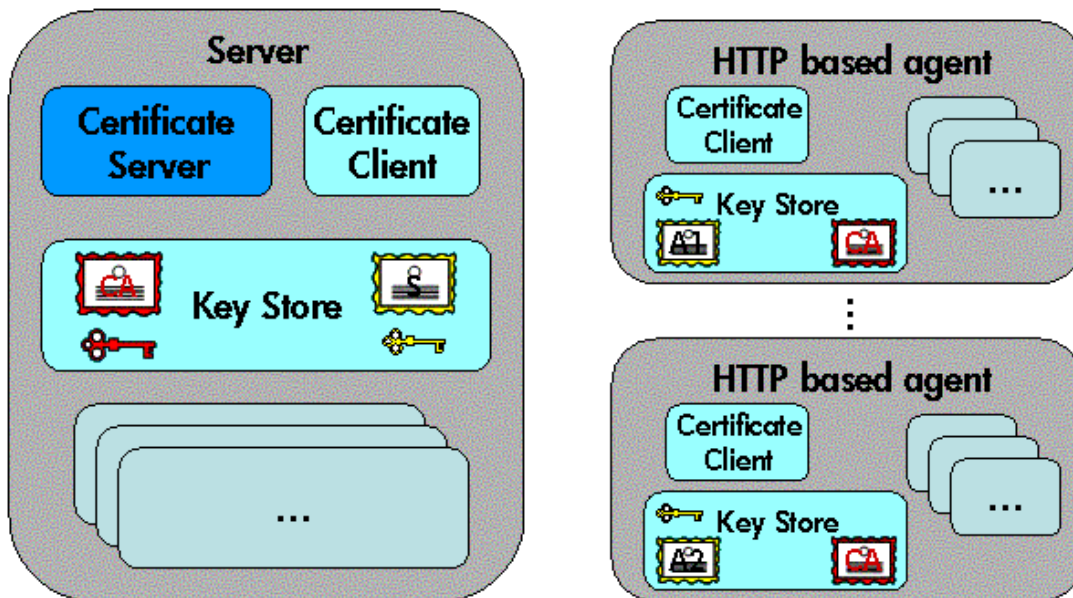
HTTPS-Based Security Components

To communicate with other Operations Agent nodes or the OM server, an Operations Agent node must have a valid, industry standard, X509 certificate. The nodes communicate with one another after exchanging certificates signed by 1024-bit keys. The exchange of certificates helps a node identify another node or server in the managed environment.

The major components responsible for creating and managing certificates are:

- Certificate Server (resides on the OM server)
- Operations Agent Key Store
- Operations Agent Certificate Client

The following figure illustrates these components:

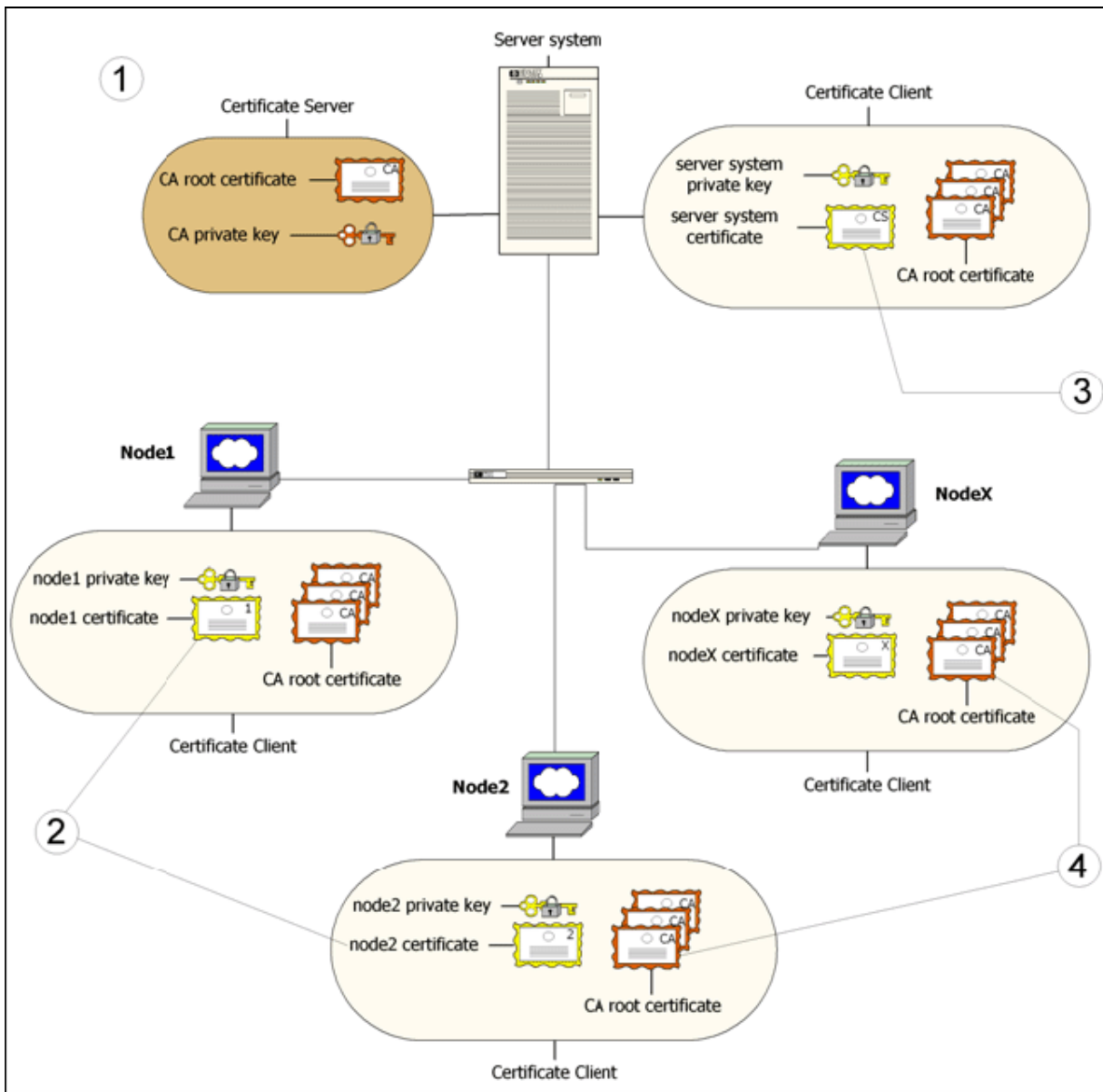


Each system hosting an Operations Agent is allocated a unique identifier value for the parameter `OvCoreId`, which is created during installation of the Operations Agent on that system.

Note: The `OvCoreId` parameter does not change for an agent node even after changing the hostname or IP address of the system.

For each agent node, `0vCoreId` is used as a unique identifier and contained in the corresponding node certificate. `0vCoreId` is allocated its value during installation.

The following figure illustrates an environment for authenticated communication in an Operations Agent deployment:



1. A server system hosts the Certificate Server, which contains the required certification authority (CA) functionality.
2. Every system has a certificate that was signed by the Certificate Server with the certification authority private key.

3. The server system also needs a certificate to manage its identity.
4. Every system has a list of trusted root certificates, which must contain at least one certificate. The trusted root (CA) certificates are used to verify the identity of the communication partners; a communication partner is trusted if the presented certificate can be validated using the list of trusted certificates.

A list of trusted root certificates is required when the certificate client is being managed by more than one OM server.

Certificates

The Operations Agent uses the following two types of certificates:

- Root certificates
- Node certificates

A root certificate is a self-signed certificate, which contains the identity of the certification authority of the certificate server. The private key belonging to the root certificate is stored on the certificate server system and protected from unauthorized access. The certification authority uses its root certificate to digitally sign all certificates.

Every agent node in the managed environment receives a node certificate issued by the certificate server. While issuing the certificate, the certificate client running on the agent node stores a corresponding private key in the file system.

Note: A node certificate contains a unique node identifier—OvCoreId. The following is an example of OvCoreId:

```
d498f286-aa97-4a31-b5c3-806e384fcf6e
```

Each node can be securely authenticated through its node certificate. The node certificate can be verified by all other nodes in the environment using the root certificate(s) to verify the signature. Node certificates are used to establish SSL-based connections between two HTTPS nodes that use client and server authentication, and can be configured to encrypt all communication.

The `ovcert` tool provided by the certificate client lists the contents of the Key Store or shows information about an installed certificate.

Operations Agent Certificate Server

The certificate server is responsible for the following:

- Creating and installing self-signed root certificates.
- Importing self-signed root certificates from the file system.
- Storing the private keys of root certificates.
- Granting or denying certification requests.
- Creating a new certificate and a corresponding private key or creating an installation key for manual certificate installation.
- Offering a service for clients to automatically retrieve trusted root certificates.

Certification Authority

Note: Every OM server is automatically configured as a Certificate Authority. The default certificate server for every agent node is the OM server associated with the node.

The certification authority is a part of the certificate server and is the center of trust in certificate management. Certificates signed by this certification authority will be regarded as valid certificates and therefore be trustworthy. The certification authority must be hosted in a highly secure location. By default, it is installed on the system hosting OM.

Since the certification authority is the root of trust, it operates with a self-signed root certificate. This root certificate and the corresponding private key are created and stored on the file system with the level of protection to allow the certification authority to operate. After initialization, the certificate authority signs granted certificate requests using its root certificate.

Certificate Client

The certificate client runs on every agent system.

The certificate client operates as follows:

- The certificate client checks whether the node has a valid certificate.
- If the node has no certificate, the certificate client generates a new public and private key pair and creates a certificate request based on the unique identity (`OvCoreId` value) of the node. The certificate client sends the certificate request to the certificate server with additional node properties, and then the certificate client waits for a response.
- The additional node properties, for example DNS name and IP address of the node helps the certificate server identify the origin of the request.
- When the certificate server issues a new certificate, the certificate client installs the certificate on the node. The certificate client can ensure that all HTTPS-based communication uses this certificate.
- If the request is not successfully processed, a descriptive error is logged and the associated status is set.

In addition, the certificate client performs the following tasks:

- The certificate client contacts a certificate server to update the server's trusted root certificates.
- It supports the import of a node certificate and the corresponding private key from the file system.
- It supports the import of trusted root certificates.
- It provides status information. Status includes `OK`, `valid certificate`, `no certificate`, `certificate requested`, and `certificate request denied`.

Root Certificate Update and Deployment

It may be necessary to update the trusted root certificates of one or more nodes, for example, in environments hosting several certificate servers.

It is possible to supply all currently trusted root certificates to certificate clients in a secure way. It is usually sufficient to supply the root certificate of the certification authority. However, it may be necessary to deploy one or more additional root certificates to selected certificate clients, for example when there is more than one certification authority in the environment.

The certificate client enables you to request the certificate server to update the trusted root certificates through the command line tool `ovcert`.

Part III: Using the Operations Agent Performance Collection Component

The Operations Agent helps you to monitor a system by collecting metrics that indicate the health, performance, resource utilization and availability of essential elements of the system.

The data collection mechanism of the **oacore** data collector is controlled by the settings in the **parm** file located in the **%ovdatadir%** directory for Windows and `/var/opt/perf` directory for UNIX or Linux. Based on the classes defined in the **parm** file, the **oacore** data collector, collects system-wide resource utilization information such as process data, performance data for different devices, transaction data and logical systems data.

The data collected by **oacore** is stored in the **Metrics Datastore**. The Metrics Datastore is a Relational Database Management System (RDBMS) based datastore. For every class of data that is logged into the Metrics Datastore, one database file is created. The database files are available at `datadir/databases/oa`.

You can use tools such as **Utility** and **Extract** to view specific information stored in the Metrics Datastore. You can also analyze historical data against the alarm definitions and report the results using the utility program's `analyze` command. The **scan report** lists the configuration settings of the **parm** file, name and definition of each application, addition/deletion or change in applications, and information about disk space availability.

You can use the process of **baselining** to compute and provide reference values based on the historic data stored in the Metrics Datastore. The baseline data provides reference values to analyze performance trends and dynamically set optimal threshold values to analyze the pattern of resource utilization.

Chapter 5: Managing Data Collection

The Operations Agent, with its embedded data collector - **oacore**, continuously collects performance and health data across your system and stores the collected data in the Metrics Datastore. You can view and analyze the logged data using tools such as **extract** and **utility**. You can also integrate the Operations Agent with data analysis tools such as Performance Manager or Reporter to analyze the data with the help of graphs and reports.

The configuration parameter file, **parm file**, enables you to configure the default data logging mechanism of the **oacore** data collector. You can use the **parm** file to control the data logging interval and the type of data logged.

Using the Metrics Datastore

With the Operations Agent version 12.05, the Metrics Datastore replaces the log file based datastore. Multiple datastores such as CODA, *Scope*, and DSI log files have been consolidated into a Relational Database Management System (RDBMS) based datastore.

After you upgrade Operations Agent 11.xx to 12.00, data from the Operations Agent 11.xx (stored in the CODA database files, *Scope* log files and DSI log files) is retained in the read-only mode and is saved at:

```
/var/opt/perf/datafiles/
```

You can access the old data through utilities such as `ovcodutil`, `extract`, `utility` or through reporting tools such as the Performance Manager and the Reporter.

Note:

- Old data from the Operations Agent 11.xx is not migrated to the Metrics Datastore.
- On a HP-UX AR system, if you upgrade the Operations Agent 11.xx to 12.xx, data from the Operations Agent 11.xx is saved at:

```
/var/opt/perf/datafiles/
```

You can access this data using tools such as `ovcodutil`, `extract`, `utility` or through reporting tools such as the Performance Manager and the Reporter.

- On a HP-UX IA system, if you upgrade the Operations Agent 11.xx to 12.xx, data from the

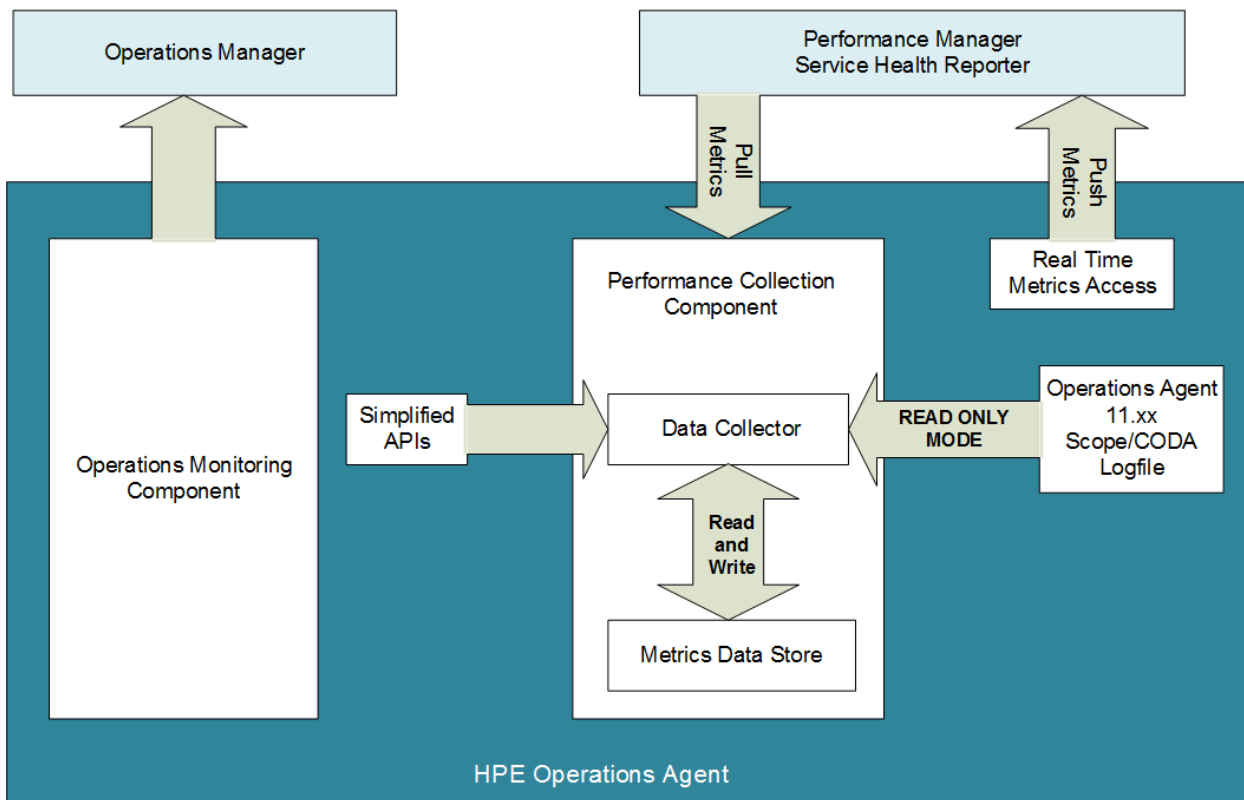
Operations Agent 11.xx is saved at:

```
/var/opt/OV/tmp/BackUp/
```

You *cannot* access this data using tools such as `ovcodauti1`, `extract`, `utility` or through reporting tools such as Performance Manager and Reporter.

- After you upgrade Operations Agent 11.xx to 12.xx, logical system data from the Operations Agent 11.xx (saved in the logfiles) cannot be accessed using tools such as the Performance Manager or the Reporter.

The following figure provides an overview of the new architecture of the Operations Agent:



Despite the change in data storing and data collection mechanism, threshold comparison process through policies remains the same.

Collecting Data

With the current version of the Operations Agent, the **CODA** and **Scope** processes (**scopeux** on UNIX and Linux nodes and **scopent** on Windows nodes) are consolidated into a single process called

oacore.

The **oacore** data collector collects a large set of system performance metrics, which presents a wide view of the health and performance of the system. The data collector captures the following information:

- System-wide resource utilization information
- Process data
- Performance data for different devices
- Transaction data
- Logical systems data

The **oacore** data collector collects data based on the data classes specified in the **parm** file. The default performance metric classes that you can define in the **parm** file are global, application, process, disk device, lvolume, transaction, configuration, netif, CPU, filesystem, and host bus adapter.

Data collected by the **oacore** data collector is stored in class specific database files. For every class of data that is logged into the Metrics datastore, one database file is created. The database files are available at **datadir/databases/oa**.

Note: Operations Agent might take up to 5 minutes to provide collected data after a fresh installation or an upgrade.

Verifying the Status of the oacore Process

If the Operations Agent is deployed from OM, the **oacore** process starts automatically. If Operations Agent is installed on a node, then the **oacore** process starts only if appropriate licenses are available on the node. For more information about licenses, see the *Operations Agent License Guide*.

The status of the **oacore** process is logged in the `System.txt` file located at:

On Windows:

```
%ovdatadir%\log\System.txt
```

On Linux:

```
/var/opt/OV/log
```

New information is appended to this file each time the **oacore** data collector starts, stops, or generates a warning or error.

After the Operations Agent is installed, follow the steps to check the status of the **oacore** process:

1. Log on to the Operations Agent node.
2. Run the command:

On Windows X64 system:

```
"%ovinstalldir%bin\win64\ovc"
```

On Windows X86 system:

```
"%ovinstalldir%bin\ovc"
```

On AIX:

```
/usr/lpp/OV/bin/ovc
```

On HP-UX/Linux/Solaris/:

```
/opt/OV/bin/ovc
```

If the **oacore** process is running, you will get the following output:

oacore	Operations Agent Core	AGENT,OA	(25357)	Running
--------	-----------------------	----------	---------	---------

Starting the oacore Process

If the **oacore** process is not running, follow the steps to start the **oacore** process:

1. Log on to the node.
2. Go to the following directory:

On Windows X64 system:

```
%ovinstalldir%bin\win64\
```

On Windows X86 system:

```
%ovinstalldir%bin
```

On AIX:

```
/usr/lpp/OV/bin
```

On HP-UX/Linux/Solaris/:

```
/opt/OV/bin
```


3. Run the following command to start the process:

```
ovc -start oacore
```

Note: To stop the **oacore** process, run the following command:

```
ovc -stop oacore
```

Note: The **oacore** completes the data access requests much before the time-out occurs. Data access requests are completed in the order in which they arrive.

If you request a large data (for example: 2 million process records), **oacore** might take longer than usual to complete the request. While **oacore** is processing a large data access request, other requests (such as `dsilog`, `extract`) may be timed out. If you get a time-out error, you must request again.

Verifying Data Logging

Metrics data collected by the **oacore** data collector is logged in the Metrics Datastore. Follow the steps to verify data logging:

1. Log on to the node.
2. Run the command:

On Windows X64 system:

```
%ovinstalldir%bin\win64\ovcodautl -obj
```

On Windows X86 system:

```
%ovinstalldir%bin\ovcodautl -obj
```

On AIX:

```
/usr/lpp/OV/bin/ovcodautl -obj
```

On HP-UX/Linux/Solaris:

```
/opt/OV/bin/ovcodautl -obj
```

If data is being logged, a list of all logged metrics is shown.

Controlling Disk Space Used by Database Files

Performance Collection Component provides for automatic management of the database files.

Controlling Disk Space Used by Database Files that Store the Default Performance Metric Classes

The size of the database files that store the *default performance metrics class*, depends on the maximum size specified in the **parm** file. If the size specification in the **parm** file is changed, **oacore** detects it *only* during startup.

The database files for a class are rolled over, when the maximum size specified in the **parm** file is reached. During rollover, 20 percent of oldest data is removed. If *size* is not specified in the **parm** file, database files are rolled over when the default maximum size of 1 GB is reached.

Note: The size of the database files cannot be set below 1 MB.

Changing the Maximum Size Specified in the parm File

For every class of data that is logged into the datastore, 5 database files are created. The size of these database files, depend on the maximum size specified in the **parm** file.

Note: For every class of data logged into the datastore, one database file is created initially. As the first database file reaches a size approximately equal to 20% of the maximum size specified in the **parm** file, the second database file is created. This process continues till 5 database files are created.

If the size specification in the **parm** file is changed, **oacore** detects it only during startup. The new size is effective only for the new database files. Eventually as older database files are removed during rollover, new database files having a size approximately equal to 20% of the maximum size specified in the **parm** file are created.

Note: The total size might differ from (may be greater or lesser than) the configured size, as the new size is effective only for the new database files. Gradually as the older database files are removed during rollover, total size would match the size specified in the **parm** file.

For example:

If you specify 10 MB as the maximum size for a class of data in the **parm** file, then 5 files each of the size 2 MB is created.

During rollover, when the maximum size (10 MB) specified in the **parm** file is reached, oldest database file (2 MB in size) is removed.

Scenario 1 - You decrease the maximum size from 10 MB to 5 MB:

You must restart the Performance Collection Component for the changes to take effect.

All the existing database files continue to be of the size 2 MB.

Eventually as older database files are removed during rollover, new database files with a maximum size of 1 MB are created.

After all old files (that existed before the restart) are removed, 5 database files each with a maximum size of 1 MB will be created. Thus the maximum size of 5 MB specified in the **parm** file will be achieved.

Note: The total size might exceed the configured size as the new size is effective only for the new database files. Gradually as the older database files are removed (during rollover), total size would match the size specified in **parm** file.

Scenario 2 - You increase the maximum size from 10 MB to 15 MB:

You must restart the Performance Collection Component for the changes to take effect.

All the existing database files continue to be of the size 2 MB.

Eventually as older database files are removed during rollover, new database files with a maximum size of 3 MB are created.

After all old files (that existed before the restart) are removed, 5 database files each with a maximum size of 3 MB will be created. Thus the maximum size of 15 MB specified in the **parm** file will be achieved.

Note: The total size might be lesser than the configured size as the new size is effective only for the new files. Gradually as the older files are removed (during rollover), total size would match the size specified in **parm** file.

Controlling Disk Space Used by Database Files that Store the Custom Data

The maximum size of the database files that store the *custom data* is set to 1 GB by default. You can configure the maximum size by using the **parm** file. If the size specification in the **parm** file is changed, **oacore** detects it *only* during startup.

Syntax to configure maximum size:

```
customsize "Datasourcename.Classname1"=10, "Datasourcename.Classname2"=10,  
"Datasourcename.*"=2, "*.Classname2"=10, "*.*"=1
```

For example: customsize "MySQL.GLOBAL"=2, "NET-APP.*"=10, "*.*"=50

In this instance,

The maximum size of class GLOBAL in MySQL datasource is set to 2 MB, for all the classes under NET-APP datasource is set to 10 MB and for all others classes it is set to 50 MB.

The following parameters control the size of custom data logged:

- Limit the maximum size of DB file (in MB) for custom class specified.
- If custom size is not specified then default maximum size of 1024 MB is set.
- If custom size less than 1 MB is specified, it will default to 1 MB.
- Maximum size supported is 2048 MB.

For every class of data logged into the datastore, one database file is created initially. As the first database file reaches a size approximately equal to 20% of maximum size, the second database file is created. This process continues till 5 database files are created. The database files are rolled over when the maximum size is reached. During roll over, 20 percent of oldest data is removed (oldest database file is deleted permanently).

Note:

- On UNIX systems, the **oacore** process requires adequate resources like open file descriptor. In most of the UNIX systems this limit is configured to 256 (ulimit -n). For the **oacore** process, this limit is sufficient to operate in normal scenarios. On systems configured to log additional custom classes, open file descriptor must be configured proportionately.

For example:

If you have about 20 managed classes, it is recommended to configure open file descriptor to 512.

If you have about 50 managed classes, it is recommended to configure open file descriptor to 1024.

After configuring open file descriptor you must restart the **oacore** process.

- On Solaris platforms, the command `ovc -start oacore` starts the `/opt/perf/bin/runoacore` script. This script sets the limit of the open file descriptor to 4096 before starting the **oacore** process.

Stopping and Restarting Data Collection

The **oacore** process and the other associated processes are designed to run continuously. The only time you should stop them are when any of the following occurs:

- You are updating the Performance Collection Component software to a new release.
- You are adding or deleting transactions in the transaction configuration file, **ttd.conf**. (For more information, see [What is Transaction Tracking?](#))
- You are modifying distribution ranges or service level objectives (SLOs) in the transaction configuration file, **ttd.conf**. (For more information on transaction tracking, see [What is Transaction Tracking?](#))
- You are changing the **parm** file and want the changes to take effect. Changes made to the **parm** file take effect only after the **oacore** process is restarted.
- You are shutting down the system.
- You are adding the hardware or modifying the configuration changes. Changes made take effect only after the **oacore** process is restarted.

Stopping Data Collection

The **ovpa** script's stop option ensures that no data is lost when **oacore** and other Performance Collection Component processes are stopped.

To manually stop data collection, run the following command:

On Windows:

```
%ovinstalldir%bin\ovpacmd stop
```

On HP-UX/Linux/Solaris:

```
/opt/perf/bin/ovpa -stop
```

On AIX:

```
/usr/lpp/perf/bin/ovpa -stop
```

Note: **oacore** does not log NFS data but you can view the NFS data through GlancePlus on the local file system.

Restarting Data Collection

You have different options for restarting data collection after the Performance Collection Component processes have stopped or configuration files have been changed and you want these changes to take effect.

To start **oacore** and the other Performance Collection Component processes after the system has been down, or after you have stopped them, use `<InstallDir>/ovpa start`. Here, `InstallDir` is the directory where Performance Collection Component is installed.

To restart **oacore** and the other processes while they are running, use `<InstallDir>/ovpa restart`. Here, `InstallDir` is the directory where Performance Collection Component is installed. This stops the currently running processes and starts them again.

When you restart **oacore**, the Performance Collection Component continues to use the same database files that were used before stopping the **oacore** process. New records are appended to the end of the existing files.

Note: The `SEM_KEY_PATH` entry in the `ttd.conf` configuration file is used for generating IPC keys for the semaphores used in `ttd` and the `midaemon` process on UNIX platforms. The default value used is `/var/opt/perf/datafiles`. You can change the value of `SEM_KEY_PATH` if `midaemon` or `ttd` does not respond because of sem id collisions.

Daylight Savings

During daylight saving if the system time advances by an hour, data logging continues without any change. If the system time moves behind by an hour, then no data is logged for an hour until the system time synchronizes with the timestamp of the last logged data.

When daylight savings is turned off, the system time advances by one hour, and therefore, the timestamp of the next logged record advances by an hour. This introduces a one-hour gap after the last logged record even though data collection does not stop.

Changing System Time Manually

If you advance the system time by an hour, data logging continues without any change. If you move the system time behind by an hour, **oacore** stops logging for an hour until the system time synchronizes with the timestamp of the last logged data.

Note: The system time must synchronize with the data collection time to avoid any gaps in the collection.

Using the parm File

The **parm** file is a text file containing instructions for the **oacore** data collector to log specific performance data.

When you install Operations Agent 12.05 on a node where previous version of Operations Agent is not installed, the **parm** file is placed in two different directories. During an upgrade from earlier versions of the Operations Agent to the current version, the **parm** file placed in one of the directories is upgraded. For more information see, availability of **parm** file in the following scenarios:

- [Installing Operations Agent 12.xx](#)
- [Upgrading to Operations Agent 12.xx](#)

Installing the Operations Agent 12.xx

When you install the Operations Agent 12.xx on a node where previous version of Operations Agent is not installed, the default **parm** file is placed in two different directories:

On Windows:

```
%ovinstalldir%\newconfig
```

```
%ovdatadir%
```

Note: On Windows, the **parm** file exists with the extension **.mwc** (**parm.mwc**).

On HP-UX/Linux/Solaris:

```
/opt/perf/newconfig
```

```
/var/opt/perf
```

On AIX:

```
/usr/lpp/perf/newconfig
```

```
/var/opt/perf
```

The data collection mechanism of the **oacore** data collector is controlled by the settings in the **parm** file located in the `%ovdatadir%` for Windows and `/var/opt/perf` for UNIX or Linux directory.

To modify the default collection mechanism, you must modify the settings in the **parm** file that is located in the `%ovdatadir%` for Windows and `/var/opt/perf` for UNIX or Linux directory.

Upgrading to the Operations Agent 12.xx

On Windows

When you upgrade Operations Agent on a node, the upgrade process updates the copy of the **parm** file available in the **newconfig** directory. The **parm** file that resides in the `%ovdatadir%` directory remains unaffected and continues to govern the data collection mechanism on the node. This method, enables you to retain the configured data collection mechanism after an upgrade.

After the product upgrade, you can compare the existing configuration settings of the **parm** file with the new version of the **parm** file available in the **newconfig** directory, and then make necessary changes. For more information on modifying the **parm** file, see ["Modifying the parm File" on the next page](#)

On Unix/Linux

On all UNIX/Linux systems, the m4 macro processor utility is used for preprocessing the **parm** file. The Performance Collection Component preprocesses the **parm** file located in the `/var/opt/perf` directory to create a run-time **parm** file which is a text file.

You can use the following command to generate a run-time **parm** file:

```
# m4 -DPARMOS=<Operatingsystem>/var/opt/perf/parm > parm.m4
```

Set the appropriate value for `<Operatingsystem>`.

Note: The value of `<Operatingsystem>` is case sensitive. To get an appropriate value for the operating system, run the following command:

```
uname
```

For example:

If you run the `uname` command on a Linux operating system, the following output is generated:

Linux

The pre-existing **parm** files, without the `m4` utility, works for the earlier versions of Operations Agent.

The system checks for `m4` at the time of installation. If `m4` is not present, make sure that you install `m4` on the system.

For more information, see the section *Prerequisites for Installing Operations Agent* in the *Operations Agent and Operations Smart Plug-in for Infrastructure Installation Guide*.

Note: You can add custom application definitions to the external file and include it in the **parm** file by using the command `#include(var/opt/perf/parm.apps)`. For more information on how to define an application, see "[Application Definition Parameters](#)" on page 174

Modifying the parm File

You can modify the **parm** file using any word processor or editor that can save a file in the ASCII format. When you modify the **parm** file, or create a new one, the following rules and conventions apply:

- Any parameter you specify overrides the default values. See the **parm** file available in the **newconfig** directory for the default values.
- The order in which the parameters are specified in the **parm** file is not important.
- If you specify a parameter more than once, the last instance of the parameter takes effect.
- The `file`, `user`, `group`, `cmd`, `argv1`, and or parameters must follow the application statement that they define.
- Application parameters must be listed in order so that a process is aggregated into an application when it is first matched.
- You can use uppercase letters, lowercase letters, or a combination of both for all commands and parameter statements.
- You can use blank spaces or commas to separate keywords in each statement.
- You can comment parameters in the **parm** file. Any line starting with a comment code (`/*`) or pound sign (`#`) is ignored.

After modifying the **parm** file, you must restart the Performance Collection Component for the changes to take effect. To restart the Performance Collection Component, run the following command:

On Windows:

```
%ovinstalldir%bin\ovpacmd REFRESH COL
```

On HP-UX/Linux/Solaris:

```
/opt/perf/bin/ovpa -restart
```

Note: You can use the command `/opt/perf/bin/ovpa -restart scope` to restart Performance Collection Component. This command is retained only for backward compatibility after you upgrade from earlier versions to the Operations Agent 12.xx.

On AIX:

```
/usr/lpp/perf/bin/ovpa -restart
```

Note: You can use the command `/usr/lpp/perf/bin/ovpa -restart scope` to restart Performance Collection Component. This command is retained only for backward compatibility after you upgrade from earlier versions to the Operations Agent 12.xx.

If you want to use the Real-Time Metric Access (RTMA) component, you must also restart the **perfd** process:

On Windows:

```
%ovinstalldir%bin\ovpacmd stop RTMA  
%ovinstalldir%bin\ovpacmd start RTMA
```

On HP-UX/Linux/Solaris:

```
/opt/perf/bin/pctl restart
```

On AIX:

```
/usr/lpp/perf/bin/pctl restart
```

parm File Parameters

The data collector - **oacore** is controlled by specific parameters in the collection parameters (**parm**) file that do the following:

- Specify data types to be logged.
- Specify the interval at which data should be logged.
- Specify attributes of processes and metrics to be logged.
- Define types of performance data to be collected and logged.

- Specify the user-definable sets of applications that should be monitored. An application can be one or more programs that are monitored as a group.

You can modify these parameters to configure **oacore** to log performance data that match the requirements of the monitored system (see [Modifying the parm File](#)).

Note: Collection of system configuration parameters happen only when you start, add or remove devices from the system.

The **parm** file parameters listed in the table are used by **oacore**. Some of these parameters are for specific systems as indicated in the table. For detailed descriptions of these parameters, see [Parameter Descriptions](#) and [Application Definition Parameters](#).

parm File Parameters Used by oacore

Parameter	Values or Options
Parameter Descriptions	
<code>id</code>	system ID
<code>log</code>	<p>global</p> <p>application [=all]</p> <p>- enables oacore to log all applications into the datastore at every interval, regardless of whether the applications are active or not.</p> <p>application [=prm] (on HP-UX only)</p> <p>- enables oacore to record active Process Resource Manager (PRM) groups to the datastore.</p> <p>process</p> <p>device=disk,lvm,cpu,filesystem,all (lvm on HP-UX and Linux only)</p> <p>transaction=correlator, resource (resource on HP-UX only)</p> <p>Note: Only high level transaction class metrics are logged after you upgrade from earlier versions to Operations Agent 12.00.</p> <p>logicalsystem (For Solaris, logical system is supported on Solaris 10 operating environment or later)</p> <p>On AIX, logical system is supported on LPAR on AIX 5L V5.3 ML3 and later and WPAR on AIX 6.1 TL2 Global environment only.</p>

parm File Parameters Used by oacore, continued

Parameter	Values or Options
	<p>For enabling lpar logging, logicalsystems=lpar logicalsystems</p> <p>For enabling wpar logging, logicalsystems=wpar</p> <p>For enabling both lpar and wpar logging, logicalsystems=lpar,wpar logicalsystems=wpar,lpar logicalsystems=all</p>
subprocinterval	value in seconds (not on HP-UX)
javaarg	True False
procthreshold (same as threshold)	<p>cpu=<i>percent</i></p> <p>disk=<i>rate</i> (not on Linux or Windows)</p> <p>memory=<i>nn</i> (values in MBs)</p> <p>IO=<i>rate</i> (values in KBs/sec)</p> <p>nonew</p> <p>nokilled</p> <p>shortlived</p>
appthreshold	cpu= <i>percent</i>
diskthreshold	util= <i>rate</i>
bynetifthreshold	iorate= <i>rate</i>
fsthreshold	util= <i>rate</i>
lvthreshold	iorate= <i>rate</i>
bycputhreshold	cpu= <i>percent</i>
fstypes	<p>To include specific file systems for data logging, use the syntax <file_system1>, <file_system2>, <file_system3>, ...</p> <p>To exclude specific file systems, use the syntax !<file_system1>, !<file_system2>, !<file_system3>, ... or !<file_system1>, <file_system2>, <file_system3>, ...</p>
wait	<p>cpu=<i>percent</i> (HP-UX only)</p> <p>disk=<i>percent</i> (HP-UX only)</p>

parm File Parameters Used by oacore, continued

Parameter	Values or Options
	mem= <i>percent</i> (HP-UX only) sem= <i>percent</i> (HP-UX only) lan= <i>percent</i> (HP-UX only)
<code>size</code>	size (values are in MBs) process=nn (the maximum value is 4096) The maximum value for the following classes is 2048: global=nn application=nn device=nn transaction=nn logicalsystem=nn
<code>collectioninterval</code>	process=ss (values in seconds) global=ss
<code>gapapp</code>	blank unassignedprocesses existingapplicationname other
<code>Flush</code>	ss(values in seconds) 0 (disables data flush)
<code>project_app</code>	true
NOTE: Only on Solaris	false (only on Solaris 10 and above)
<code>proccmd</code>	0 (disables logging of process commands) 1 (enables logging of process commands) The maximum length of the process command logged is always 4096. However, real-time tools like cpsh will show full command irrespective of the length.
<code>proclist</code>	all (the Performance Collection Component in the global zone monitors all the processes that are running in the global and non-global zones) local (the Performance Collection Component in the global zone monitors only the processes that are running in the global zone) This parameter has no effect in non-global zones.
<code>appproc</code>	all (configures the Performance Collection Component to calculate APP_)

parm File Parameters Used by oacore, continued

Parameter	Values or Options
NOTE: Only on Solaris	<p>metrics with processes for applications that belong to the global and non-global zones)</p> <p>local (configures the Performance Collection Component to calculate APP_metrics with processes for applications that belong to the global zone only)</p> <p>This parameter has no effect in non-global zones.</p>
<code>ignore_mt</code>	<p>true(CPU metrics of global class report values normalized against the active number of cores in the system)</p> <p>false(CPU metrics of global class report values normalized against active number of CPU threads in the system)</p> <p>ineffective(multithreading is turned off)</p> <p>NOTE: This parameter has no effect on HP-UX. You must run the midaemon -ignore_mt command on HP-UX to switch between the above modes. For more information, see Logging Metrics Calculated with the Core-Based Normalization.</p>
<code>cachemem</code>	<p>f or free (The Operations Agent does not include the buffer cache size when calculating the value of the GBL_MEM_UTIL metric)</p> <p>u or user (The Operations Agent includes the buffer cache size when calculating the value of the GBL_MEM_UTIL metric)</p>
Application Definition Parameters	
<code>application</code>	<i>application name</i>
<code>file</code>	<i>file name [, ...]</i>
<code>argv1</code>	first command argument [,]
<code>cmd</code>	command line regular expression
<code>user</code>	<i>user login name [,]</i>
<code>group</code>	<i>groupname [,]</i>
<code>or</code>	
<code>priority</code>	low value-high value (range varies by platform)

Parameter Descriptions

Following are descriptions of each of the **parm** file parameters:

ID

The system ID value is a string of characters that identifies your system. The default ID assigned is the system's hostname. If you want to modify the default ID assigned, make sure all the systems have unique ID strings. This identifier is included in the datastore to identify the system on which the data was collected. You can specify a maximum of 39 characters.

Log

The log parameter specifies the data types to be collected by **oacore**.

- **log global** enables **oacore** to record global records into the datastore. You must have global data records to view and analyze performance data on your system. Global metrics are not affected by logging options or values of application or process data.
- **log application** enables **oacore** to record active application records into the datastore. By default, **oacore** logs only applications that have active processes during an interval.

The parameter, `log application=all` in the **parm** file enables **oacore** to log all applications into the datastore at every interval, regardless of whether the applications are active or not. The `application=all` option may be desirable in specific circumstances in relation to the use of application alarms. For example, you can generate alarm when an application becomes inactive (`APP_ALIVE_PROC`).

Only on HP-UX, you can specify the parameter `log application=prm` to enable **oacore** to record active Process Resource Manager (PRM) groups to the datastore. If you specify this parameter, **oacore** will not record user-defined application sets listed in the **parm** file. In addition, all application metrics collected will reflect a PRM context and will be grouped by the `APP_NAME_PRM_GROUPNAME` metric.

Application logging options do not affect global or process data.

- **log process** enables **oacore** to record information about interesting processes into the datastore. A process may become interesting when it is first created, when it ends, and when it exceeds a threshold specified in the **parm** file for an application. Process threshold logging options have no effect on global or application data.
- **log device=disk, lvm, cpu, filesystem** enables **oacore** to record information about individual disks, logical volumes (HP-UX and Linux only), CPUs, and file systems into the datastore.

Note: Use `lvm` only on systems that run with HP-UX or Linux operating system.

By default, only logical volumes and interfaces that had I/O generated through them during an interval are logged. `netif` (logical LAN device) records are always logged regardless of the selected log device options.

Note: By default, `netif` records are logged even if it is not specified in the `parm` file.

For Example:

To request logging of records for individual disks, logical volumes, CPUs, network interfaces, but *not* individual file systems, use the following setting:

```
log device=disk, lvm, cpu, netif
```

When filesystem is specified, all mounted local file systems are logged at every interval, regardless of the activity.

`log device=all` in the `parm` file enables `oacore` to log all disk, logical volume, CPU, and network interface devices to the datastore at every interval, regardless of whether the devices are active or not.

- **log transaction** enables `oacore` to record ARM transactions into the datastore. To enable `oacore` to collect data, a process that is instrumented with the Application Response Measurement (ARM) API must be running on your system.

The default values for the log transaction parameter are no resource and no correlator.

To enable resource data collection (HP-UX only) or correlator data collection, specify **log transaction=resource** or **log transaction=correlator**. Both can be logged by specifying **log transaction=resource, correlator**.

Note: When you upgrade to Operations Agent 12.xx log transaction enables `oacore` to record only high level transaction class metrics.

- **log logicalsystems** enables `oacore` to record information about the logical systems into the datastore. Data for logical systems is summarized periodically at intervals specified in the `parm` file.

Note: BYLS metrics is not supported on Windows and Linux platforms.

Operations Agent 12.xx does not collect BYLS metrics from Xen, KVM, VMware vSphere, Hyper-V and other virtualization domains.

On AIX 6.1 TL2, BYLS logging for LPAR and WPAR can be configured by using the `logicalsystems` parameter in the `parm` file. For more information, ["parm File Parameters Used by oacore" on page 155](#).

Note: If you specify the log parameter without options, **oacore** logs only the global and process data.

Thresholds

Threshold parameters enable **oacore** to log only critical information in the datastore and filter out unnecessary, non-critical details of the system.

When the threshold value specified exceeds for a particular instance of a class of data, a record for that instance is logged by **oacore**. You can specify lower values for the threshold, to enable **oacore** to log more data or you can specify higher values for the threshold, to enable **oacore** to log lesser data.

The following parameters specify the thresholds for different classes of metrics:

- [Procthreshold](#)
- [apptreshold](#)
- [diskthreshold](#)
- [bynetifthreshold](#)
- [fsthreshold](#)
- [lvthreshold](#)
- [bycputhreshold](#)

Procthreshold

The `procthreshold` parameter is used to set activity levels to specify criteria for interesting processes. To use this parameter, you must enable process logging. `procthreshold` affects only processes that are logged and does not affect other classes of data.

You must specify threshold options on the same parameter line (separated by commas).

procthreshold Options for Process Data

cpu	<p>Sets the percentage of CPU utilization that a process must exceed to become “interesting” and be logged.</p> <p>The value percent is a real number indicating overall CPU use.</p> <p>For example:</p> <p>cpu=7.5 indicates that a process is logged if it exceeds 7.5 percent of CPU utilization in a 1-minute sample.</p>
disk	<p>Sets the rate of physical disk I/O per second that a process must exceed to become 'interesting' and be logged.</p>

procthreshold Options for Process Data , continued

	<p>The value is a real number.</p> <p>Note: This option is not available on Windows.</p> <p>For example:</p> <p>disk=8.0 indicates that a process will be logged if the average physical disk I/O rate exceeds 8 KBs per second.</p>
memory	<p>Sets the memory threshold that a process must exceed to become “interesting” and be logged.</p> <p>The value is in megabyte units and is accurate to the nearest 100 KB. If set, the memory threshold is compared with the value of the PROC_MEM_VIRT metric.</p> <p>Each process that exceeds the memory threshold will be logged, similarly to the disk and CPU process logging thresholds.</p>
IO	<p>Sets the IO threshold that a process must exceed to become "interesting" and be logged.</p> <p>The value is in kilobyte units.</p> <p>For example:</p> <p>IO=100 indicates that a process will be logged if the process I/O rate (PROC_IO_BYTE_RATE) exceeds 100 KBs per second.</p>
nonew	<p>Disables logging of new processes if they have not exceeded any threshold.</p> <p>If not specified, all new processes are logged.</p> <p>On HP-UX, if <code>shortlived</code> is <i>not</i> specified, then only new processes that lasted more than one second are logged.</p>
nokilled	<p>Disables logging of exited processes if they did not exceed any threshold. If not specified, all killed (exited) processes are logged.</p> <p>On HP-UX, if <code>shortlived</code> is <i>not</i> specified, then only killed processes greater than one second are logged.</p>
shortlived	<p>Enables logging of processes that ran for less than one second in an interval. This often significantly increases the number of processes logged.</p> <p>If <code>oacore</code> finds the threshold <code>shortlived</code> in the <code>parm</code> file, it logs shortlived processes, regardless of the CPU or disk threshold, as long as the <code>nonew</code> and <code>nokilled</code> options are removed.</p> <p>By default no <code>shortlived</code> processes are logged.</p>

`procthreshold` specifies the thresholds for the PROCESS class. The default values are as follows:

- Processes that used more than 10% of a processor's worth of CPU during the last interval.
- Processes that had a virtual memory set size over 900 MB.
- Processes that had an average physical disk I/O rate greater than 5 KB per second.

apptreshold

The `apptreshold` parameter is used to specify threshold values for the APPLICATION data class (APP_CPU_TOTAL_UTIL metric). The threshold criteria is based on the percentage of CPU utilization that an application must exceed for the application to be recorded in the datastore.

The default setting in the `parm` file enables `oacore` to record applications that use more than 0% of CPU.

diskthreshold

The `diskthreshold` parameter is used to specify the threshold values for DISK class. The threshold criteria for DISK class is based on the percentage of time duration, a disk performs I/Os (BYDSK_UTIL metric).

The default setting in the `parm` file enables `oacore` to record the details of disks that are busy performing I/Os for more than 10% of the time duration.

bynetifthreshold

The `bynetifthreshold` parameter specifies the thresholds for the NETIF class. Netif data class threshold criteria is based on the number of packets transferred by the network interface per second (BYNETIF_PACKET_RATE metric).

The default setting in the `parm` file enables `oacore` to record the details of network interfaces that transfer more than 60 packets per second. If the value for this parameter is not specified or if the parameter is commented out, `oacore` logs the details of all the network interfaces that are not idle.

fsthreshold

The `fsthreshold` parameter specifies the thresholds for the FILESYSTEM class. The file system data class threshold criteria is based on the percentage of disk space utilized by the filesystems (FS_SPACE_UTIL metric).

The default setting in the **parm** file enables **oacore** to record the details of filesystems that utilize more than 70% of disk space.

lvthreshold

The `lvthreshold` specifies the thresholds for the LOGICALVOLUME class. Logical volume data class threshold values are based on I/Os per second (LV_READ_RATE + LV_WRITE_RATE).

The default setting in the **parm** file enables **oacore** to record the details of logical volumes that have more than 35 I/Os per second.

bycputhreshold

The `bycputhreshold` parameter specifies the thresholds for CPU class. CPU data class threshold criteria is based on percentage of time the CPU was busy (BYCPU_CPU_TOTAL_UTIL).

The default setting in the **parm** file enables **oacore** to record the details of CPUs that are busy for more than 90% of the time.

subprocinterval

The `subprocinterval` parameter, if specified, overrides the default that **oacore** uses to sample process data.

Process data and global data are logged periodically at intervals specified in the **parm** file. However, **oacore** probes its instrumentation every few seconds to catch short-term activities. This instrumentation sampling interval is 5 seconds by default.

The process data logging interval must be an even multiple of the `subprocinterval`. For more information, see ["Configuring Data Logging Intervals" on page 181](#).

On some systems with thousands of active threads or processes, the `subprocinterval` should be made longer to reduce overall **oacore** overhead. On other systems with many short-lived processes that you may want to log, setting the `subprocinterval` lower could be considered, although the effect on **oacore** overhead should be monitored closely in this case. This setting must take values that are factors of the process logging interval as specified in the **parm** file.

Note: Lower values for the `subprocinterval` will decrease the gap between global metrics and the sum of applications on all operating systems other than HP-UX.

gapapp

`gapapp` parameter in the **parm** file controls the modification of the application class of data to account for any difference between the global (system-wide) data and the sum of application data.

Application data is derived from process-level instrumentation. Typically there is difference between the global metrics and the sum of applications. In systems which have high process creation rates the difference will be significant. You can choose from the following options:

- If `gapapp` is blank, an application named `gapapp` is added to the application list.
- If `gapapp = UnassignedProcesses`, an application by the name `UnassignedProcesses` is added to the application list.
- If `gapapp = ExistingApplicationName` (or) `gapapp = other`, the difference to the global values is added to the specified application instead of being logged separately and a new entry is added to the application list.

fstypes

The `fstypes` parameter enables you to monitor specific file systems on the system. By default, the **oacore** collector logs data for all file systems that are attached to the system. With the `fstypes` parameter, you can enable data collection for only specific file systems.

In the **parm** file, you must set the `fstypes` parameter to the name of the file system that you want to monitor. You can specify multiple file system names separated by commas.

The syntax for this parameter is:

- To include specific file systems for data logging: `fstypes= <file_system1>, <file_system2>, ...`
- To exclude specific file systems: `fstypes= !<file_system1>, !<file_system2>, ...` or `fstypes= !<file_system1>, <file_system2>, ...`

Here, `<file_system1>` and `<file_system2>` are the file system types.

For HP-UX, Linux, and AIX, all available file systems along with file system types are listed in the **/etc/fstab** file.

For AIX, all available file systems along with file system types are listed in the **/etc/filesystems** file.

For Solaris, all available file systems along with file system types are listed in the **/etc/vfstab** file.

To find out the file system type on Windows, right-click the disk drive in the Windows Explorer, and then click **Properties**. The value displayed for File system is the file system type that you can specify with the `fstypes` parameter.

You can set this parameter to an inclusion list, which is a comma-separated list of file system types indicates an inclusion list and enables the agent to monitor data only for the specified file systems. You can also set this parameter to an exclusion list, which is a comma-separated list of file system types that begins with the character `!` in front of every file system type to be excluded. Specifying the `!` character in the beginning of the list ensures that the agent does not monitor any file system that belongs to the list.

While configuring the `fstypes` parameter, use the file system names returned by operating system commands.

Example 1:

```
fstypes = tmpfs, mvfs, nfs
```

In this instance, data for `tmpfs`, `mvfs`, and `nfs` file system types will be included.

Example 2:

```
fstypes = !tmpfs, !mvfs, !nfs
```

Or

```
fstypes = !tmpfs, mvfs, nfs
```

In this instance, Operations Agent will monitor all the file systems that are available except for `tmpfs`, `mvfs` and `nfs` file system types.

Example 3:

```
fstypes = or fstypes = *
```

Note: Specifying `*` or blank character ensures that Operations Agent monitors all the file systems that are available.

By default, **oacore** collector logs data for the following file system types if `fstypes` is not specified in the **parm** file:

Operating System	Default <code>fstypes</code> Logged
HP-UX	All file system types except "memfs", "nfs*", "mvfs", "cifs", "lofs".
Linux	All file system types except "nfs*", "autofs", "tmpfs", "mvfs", "cdfs".
AIX	All file system types with <code>fstypes</code> "jfs*", "aix".

Solaris	All file system types with fstypes "ufs", "zfs", "tmpfs", "vxfs", "lofs".
Windows	Data for all file system types is logged by default.

wait

You can use the `wait` parameter (on HP-UX only) to capture details of processes which wait for system resources. You can specify the value of the `wait` parameter in percentage.

When a process waits for system resources: `cpu`, `disk`, `mem`, `sem`, and `lan`, for a percentage of interval greater than the value specified for the `wait` parameter then the details of that process are logged in the datastore. See ["parm File Parameters Used by oacore" on page 155](#) for values and options.

For example:

If process logging interval is defined as 60 seconds and the `wait` parameter for the CPU is set to 50%, any process waiting for CPU for more than or equal to 30 seconds is captured in the datastore.

Size

The `size` parameter is used to set the maximum size (in megabytes) of the database files. The **oacore** collector reads these specifications when it is initiated.

Database files that store the *default performance metric classes* are rolled over when the maximum size specified in the **parm** file is reached. If `size` is not specified in the **parm** file, database files are rolled over when the maximum size of 1 GB is reached.

The maximum size of the database files that store the *custom data* is set to 1 GB by default. These database files are rolled over when the default maximum size of 1 GB is reached.

For more information about controlling disk space used by database files, see [Controlling Disk Space Used by Database Files](#).

javaarg

The `javaarg` parameter is a flag that only affects the value of the **proc_proc_argv1** metric. This parameter is set to true by default.

When `javaarg` is true, `proc_proc_argv1` metric is logged with the class or jar name. This is very useful to define applications specific to Java. When there is class name in the command string for a process, you can use the `argv1=` application qualifier to define your application by class name.

When `javaarg` is set to false or if it is not defined in the **parm** file, `proc_proc_argv1` metric is logged with the value of the first argument in the command string for the process.

For example:

When the following process is executed:

```
java -XX:MaxPermSize=128m -XX:+CMSClassUnloadingEnabled -Xms8000m -Xmx8000m -  
Dserver_port=1099 -jar ./ApacheJMeter.jar
```

- If `javaarg` is set to **True**, then value of `proc_proc_argv1` will be `-jar ./ApacheJMeter.jar`
- If `javaarg` is set to **False**, then value of `proc_proc_argv1` will be `-XX:MaxPermSize=128m`

Flush

The `flush` parameter specifies the data logging intervals (in seconds) at which all instances of application and device data will be logged. The flush intervals must be in the range 300-32700 and be an even multiple of 300.

The default value of `flush` interval is 3600 seconds for all instances of application and device data.

You can disable the `flush` parameter by specifying the value as 0 (zero). If the `flush` parameter is set to 0, **oacore** will not log application and device data which does not meet the thresholds specified in the **parm** file.

project_app

If you set the `project_app` parameter to `True`, the Performance Collection Component deems each Solaris project as an application (and the project ID as the application ID). To ignore Solaris projects, set this parameter to `False`.

Note: `project_app` parameter is supported only for Solaris 10 and above.

proclist

You can use this parameter only in Solaris global zones; it has no effect on the Operations Agent that is running in a non-global zone.

In a global zone, if you set this parameter to `all`, the Performance Collection Component monitors all global and non-global zone processes. To monitor only the processes that belong to the global zone, set this parameter to `local`.

appproc

appproc parameter is available only on Solaris. You can use this parameter only in a global zone; it has no effect on the Operations Agent that is running in a non-global zone.

In a global zone, if you set this parameter to `a11`, the Performance Collection Component includes the processes for global and non-global zone applications while calculating values of all APP_ metrics. To include only the global zone applications for the calculation of APP_ metrics, set this parameter to **local**.

proccmd

The `proccmd` parameter enables logging of process commands to the datastore. By default, the value of this process is set to 0 and the logging of process commands is disabled. To enable logging of process commands, set the value of this parameter to 1.

Note: `proccmd` parameter logging is turned on when the value of this parameter is greater or equal to 1. The maximum length of the process command logged is always 4096 irrespective of the value specified in this parameter. However, real-time tools like `cpsh` will show full command irrespective of the length.

ignore_mt

If you set this parameter to **True**, the Performance Collection Component logs all the CPU-related metrics of the Global class after normalizing the metric values against the number of active cores on the monitored system.

When this parameter is set to **False**, the Performance Collection Component logs all the CPU-related metrics of the Global class after normalizing the metric values against the number of threads on the monitored system.

On Linux machines, this parameter is set to false by default.

This parameter has no effect on HP-UX. You must run the `midaemon -ignore_mt` command on HP-UX to switch between the modes. For more information, ["Logging Metrics Calculated with the Core-Based Normalization" on page 189](#).

When `ignore_mt` flag is set as a command-line parameter to `midaemon` on HP-UX systems, few of the metrics are affected. For more information on the list of metrics which get affected, See ["Metrics affected while setting the ignore_mt flag to midaemon on HP-UX systems" on page 1](#).

The Performance Collection Component ignores this parameter if the multi-threading property is disabled on the system. As a result, the value of the GBL_IGNORE_MT metric is logged as **True**.

Note: If you enable or disable Simultaneous Multi-Threading (SMT) on a Windows, Linux, or Solaris system, you must restart the system.

Metrics Affected while Setting the ignore_mt flag to midaemon on HP-UX Systems

Global Metric Class	
GBL_CPU_TOTAL_UTIL	GBL_CPU_CSWITCH_UTIL_CUM
GBL_CPU_TOTAL_UTIL_CUM	GBL_CPU_CSWITCH_UTIL_HIGH
GBL_CPU_TOTAL_UTIL_HIGH	GBL_CPU_CSWITCH_TIME
GBL_CPU_TOTAL_TIME	GBL_CPU_CSWITCH_TIME_CUM
GBL_CPU_TOTAL_TIME_CUM	GBL_CPU_INTERRUPT_UTIL
GBL_CPU_SYS_MODE_UTIL	GBL_CPU_INTERRUPT_UTIL_CUM
GBL_CPU_SYS_MODE_UTIL_CUM	GBL_CPU_INTERRUPT_UTIL_HIGH
GBL_CPU_SYS_MODE_TIME	GBL_CPU_INTERRUPT_TIME
GBL_CPU_SYS_MODE_TIME_CUM	GBL_CPU_INTERRUPT_TIME_CUM
GBL_CPU_TRAP_TIME	GBL_CPU_VFAULT_TIME
GBL_CPU_TRAP_TIME_CUM	GBL_CPU_VFAULT_TIME_CUM
GBL_CPU_TRAP_UTIL	GBL_CPU_VFAULT_UTIL
GBL_CPU_TRAP_UTIL_CUM	GBL_CPU_VFAULT_UTIL_CUM
GBL_CPU_TRAP_UTIL_HIGH	GBL_CPU_VFAULT_UTIL_HIGH
GBL_CPU_USER_MODE_TIME	GBL_CPU_IDLE_UTIL
GBL_CPU_USER_MODE_TIME_CUM	GBL_CPU_IDLE_UTIL_CUM
GBL_CPU_USER_MODE_UTIL	GBL_CPU_IDLE_UTIL_HIGH
GBL_CPU_USER_MODE_UTIL_CUM	GBL_CPU_IDLE_TIME
GBL_CPU_NICE_UTIL	GBL_CPU_IDLE_TIME_CUM
GBL_CPU_NICE_UTIL_CUM	GBL_CPU_NORMAL_UTIL
GBL_CPU_NICE_UTIL_HIGH	GBL_CPU_NORMAL_UTIL_CUM
GBL_CPU_NICE_TIME	GBL_CPU_NORMAL_UTIL_HIGH

GBL_CPU_NICE_TIME_CUM	GBL_CPU_NORMAL_TIME
GBL_CPU_NNICE_UTIL	GBL_CPU_NORMAL_TIME_CUM
GBL_CPU_NNICE_UTIL_CUM	GBL_CPU_SYSCALL_UTIL
GBL_CPU_NNICE_UTIL_HIGH	GBL_CPU_SYSCALL_UTIL_CUM
GBL_CPU_NNICE_TIME	GBL_CPU_SYSCALL_UTIL_HIGH
GBL_CPU_NNICE_TIME_CUM	GBL_CPU_SYSCALL_TIME
GBL_CPU_REALTIME_UTIL	GBL_CPU_SYSCALL_TIME_CUM
GBL_CPU_REALTIME_UTIL_CUM	GBL_CPU_WAIT_UTIL
GBL_CPU_REALTIME_UTIL_HIGH	GBL_CPU_WAIT_TIME
GBL_CPU_REALTIME_TIME	GBL_CPU_WAIT_TIME_CUM
GBL_CPU_REALTIME_TIME_CUM	GBL_CPU_WAIT_UTIL_CUM
GBL_CPU_CSWITCH_UTIL	GBL_CPU_WAIT_UTIL_HIGH
Application Metric Class	
APP_PRM_CPU_TOTAL_UTIL_CUM	APP_CPU_NORMAL_UTIL
APP_CPU_NNICE_UTIL	APP_CPU_USER_MODE_UTIL
APP_CPU_NNICE_TIME	APP_CPU_TOTAL_TIME
APP_CPU_TOTAL_UTIL	APP_CPU_SYS_MODE_TIME
APP_CPU_TOTAL_UTIL_CUM	APP_CPU_NICE_TIME
APP_CPU_SYS_MODE_UTIL	APP_CPU_REALTIME_TIME
APP_CPU_NICE_UTIL	APP_CPU_NORMAL_TIME
APP_CPU_REALTIME_UTIL	APP_CPU_USER_MODE_TIME
PROC Metric Class	
PROC_CPU_TOTAL_UTIL	PROC_CPU_REALTIME_UTIL
PROC_CPU_TOTAL_UTIL_CUM	PROC_CPU_REALTIME_UTIL_CUM
PROC_CPU_TOTAL_TIME	PROC_CPU_REALTIME_TIME
PROC_CPU_TOTAL_TIME_CUM	PROC_CPU_REALTIME_TIME_CUM
PROC_CPU_SYS_MODE_UTIL	PROC_CPU_CSWITCH_UTIL
PROC_CPU_SYS_MODE_UTIL_CUM	PROC_CPU_CSWITCH_UTIL_CUM

PROC_CPU_SYS_MODE_TIME	PROC_CPU_CSWITCH_TIME
PROC_CPU_SYS_MODE_TIME_CUM	PROC_CPU_CSWITCH_TIME_CUM
PROC_CPU_USER_MODE_UTIL	PROC_CPU_INTERRUPT_UTIL
PROC_CPU_USER_MODE_UTIL_CUM	PROC_CPU_INTERRUPT_UTIL_CUM
PROC_CPU_USER_MODE_TIME	PROC_CPU_INTERRUPT_TIME
PROC_CPU_USER_MODE_TIME_CUM	PROC_CPU_INTERRUPT_TIME_CUM
PROC_CPU_NICE_UTIL	PROC_CPU_NORMAL_UTIL
PROC_CPU_NICE_UTIL_CUM	PROC_CPU_NORMAL_UTIL_CUM
PROC_CPU_NICE_TIME	PROC_CPU_NORMAL_TIME
PROC_CPU_NICE_TIME_CUM	PROC_CPU_NORMAL_TIME_CUM
PROC_CPU_NNICE_UTIL	PROC_CPU_SYSCALL_UTIL
PROC_CPU_NNICE_UTIL_CUM	PROC_CPU_SYSCALL_UTIL_CUM
PROC_CPU_NNICE_TIME	PROC_CPU_SYSCALL_TIME
PROC_CPU_NNICE_TIME_CUM	PROC_CPU_SYSCALL_TIME_CUM
PROC_CPU_ALIVE_TOTAL_UTIL	
PROC_CPU_ALIVE_USER_MODE_UTIL	
PROC_CPU_ALIVE_SYS_MODE_UTIL	
BYCPU Metric Class	
BYCPU_CPU_TOTAL_UTIL	BYCPU_CPU_INTERRUPT_TIME
BYCPU_CPU_TOTAL_UTIL_CUM	BYCPU_CPU_INTERRUPT_TIME_CUM
BYCPU_CPU_TRAP_TIME	BYCPU_CPU_INTERRUPT_UTIL
BYCPU_CPU_TRAP_TIME_CUM	BYCPU_CPU_INTERRUPT_UTIL_CUM
BYCPU_CPU_TRAP_UTIL	BYCPU_CPU_CSWITCH_TIME
BYCPU_CPU_TRAP_UTIL_CUM	BYCPU_CPU_CSWITCH_TIME_CUM
BYCPU_CPU_USER_MODE_TIME	BYCPU_CPU_CSWITCH_UTIL
BYCPU_CPU_USER_MODE_TIME_CUM	BYCPU_CPU_CSWITCH_UTIL_CUM
BYCPU_CPU_USER_MODE_UTIL	BYCPU_CPU_VFAULT_TIME
BYCPU_CPU_USER_MODE_UTIL_CUM	BYCPU_CPU_VFAULT_TIME_CUM

BYCPU_CPU_NICE_TIME	BYCPU_CPU_VFAULT_UTIL
BYCPU_CPU_NICE_TIME_CUM	BYCPU_CPU_VFAULT_UTIL_CUM
BYCPU_CPU_NICE_UTIL	BYCPU_CPU_REALTIME_TIME
BYCPU_CPU_NICE_UTIL_CUM	BYCPU_CPU_REALTIME_TIME_CUM
BYCPU_CPU_NNICE_TIME	BYCPU_CPU_REALTIME_UTIL
BYCPU_CPU_NNICE_TIME_CUM	BYCPU_CPU_REALTIME_UTIL_CUM
BYCPU_CPU_NNICE_UTIL	BYCPU_CPU_NORMAL_TIME
BYCPU_CPU_NNICE_UTIL_CUM	BYCPU_CPU_NORMAL_TIME_CUM
BYCPU_CPU_TOTAL_TIME	BYCPU_CPU_NORMAL_UTIL
BYCPU_CPU_TOTAL_TIME_CUM	BYCPU_CPU_NORMAL_UTIL_CUM
BYCPU_CPU_SYS_MODE_TIME	BYCPU_CPU_SYSCALL_TIME
BYCPU_CPU_SYS_MODE_TIME_CUM	BYCPU_CPU_SYSCALL_TIME_CUM
BYCPU_CPU_SYS_MODE_UTIL	BYCPU_CPU_SYSCALL_UTIL
BYCPU_CPU_SYS_MODE_UTIL_CUM	BYCPU_CPU_SYSCALL_UTIL_CUM
SYSCALL Metric Class	
SYSCALL_CPU_TOTAL_TIME	SYSCALL_CPU_TOTAL_TIME_CUM

cachemem

The `cachemem` parameter in the `parm` file enables you to configure the agent to include the buffer cache size while reporting the total memory utilization data.

You can set the `cachemem` parameter to one of the following:

1. **Set the `cachemem` parameter to free (f).**

By default, the parameter is set to **free (f)**. The following metric values are affected:

GBL_MEM_UTIL- Does not include the buffer cache size.

GBL_MEM_FREE_UTIL - Includes the buffer cache size.

2. **Set the `cachemem` parameter to user (u).**

The following metric values are affected:

GBL_MEM_UTIL- Includes the buffer cache size.

GBL_MEM_FREE_UTIL - Does not include the buffer cache size.

Note:

On AIX machines, you should set the `cachemem` parameter to **user (u)** to match with `topas` commands.

On Solaris machines, `cachemem` parameter is applicable only for Adaptive Replacement Cache (ARC) for ZFS.

`cachemem` parameter is set to **free (f)** by default. `GBL_MEM_UTIL` excludes ZFS ARC cache when `cachemem` parameter in the `parm` file is set to **free**.

Application Definition Parameters

The following parameters pertain to application definitions: `application`, `file`, `user`, `group`, `cmd`, `argv1`, and `or`.

The Performance Collection Component groups logically related processes together into an application to log the combined effect of the processes on computing resources such as memory and CPU.

Note: In PRM mode (for HP-UX only), active PRM groups are logged and the user-defined application sets, listed in the `parm` file are ignored.

An application can be a list of files (base program names), a list of commands, or a combination of these qualified by user names, group names, or argument selections. All these application qualifiers can be used individually or in conjunction with each other.

For example:

If both `cmd` and `user` qualifiers are used then a process must meet the specification of both the command string and the user name to belong to that application.

Each qualifier is discussed in detail below.

Note: Any process on the system belongs to only one application. No process is counted into two or more applications.

Application Name

The application name defines an application or class that groups together multiple processes and reports on their combined activities.

When you specify an application the following rules or conventions apply:

- The application name is a string of up to 19 characters used to identify the application.
- Application names can be in lowercase or uppercase and can contain letters, numbers, underscores, and embedded blanks.
- Do not use the same application name more than once in the **parm** file.
- An equal sign (=) is optional between the application keyword and the application name.
- The application name must precede any combination of `file`, `user`, `group`, `cmd`, `argv1`, and or parameters that refer to it, with all such parameters applying against the last application workload definition.
- Each parameter can be up to 170 characters long, including the carriage return character, with no continuation characters permitted. If your list of files is longer than 170 characters, continue the list on the next line after another `file`, `user`, `group`, `cmd` or `argv1` statement.
- You can define up to 998 applications.
- You can have a maximum of 4096 `file`, `user`, `group`, `argv1`, and `cmd` specifications for all applications combined.
- Performance Collection Component predefines an application named `other`. The `other` application collects all processes not captured by `application` statements in the **parm** file.

For example:

```
application Prog_Dev  
file vi,cc,ccom,pc,pascomp,dbx,xdb
```

```
application xyz  
file xyz*,startxyz
```

Where:

`xyz*` counts as only one specification even though it can match more than one program file.

Note: If a program file is included in more than one application, it is logged in the first application that contains it.

The default **parm** file contains some sample applications that you can modify. The **examples** directory also contains other samples (in a file called **parm_apps**) that you can copy in your **parm** file and modify as needed.

File

The **file** parameter specifies the program files that belong to an application. All interactive or background executions of these programs are included. It applies to the last application statement issued. An error is generated if no **application** statement is found.

The file name can be any of the following:

- **On UNIX/Linux:**
 - A single UNIX program file such as `vi`.
 - A group of UNIX program files (indicated with a wild card) such as `xyz*`. In this case, any program name that starts with the letters `xyz` is included. A file specification with wild cards counts as only one specification toward the maximum allowed.
- **On Windows:**
 - A single program file such as `winword`.
 - A group of program files (indicated with a wild card) such as `xyz*`. In this case, any program name that starts with the letters `xyz` is included. A file specification with wild cards counts as only one specification toward the maximum of 1000 for all file specifications.

Note: For Windows, when you define executable files for an application in the **parm** file, no file extensions are required. For example, you can define `winword` in the **parm** file without its `.exe` extension.

The name in the **file** parameter is limited to 15 characters in length. An equal sign (=) is optional between the **file** parameter and the file name.

You can enter multiple file names on the same parameter line (separated by commas) or in separate file statements. File names cannot be qualified by a path name. The file specifications are compared to the specific metric `PROC_PROC_NAME`, which is set to a process's `argv[0]` value (typically its base name).

For Example:

On UNIX/Linux:

```
application = prog_dev
file = vi,vim,gvim,make,gmake,lint*,cc*,gcc,ccom*,cfront
file = cpp*,CC,cpass*,c++*
file = xdb*,adb,pxdb*,dbx,xlC,ld,as,gprof,lex,yacc,are,nm,gencat
```



```
file = javac,java,jre,aCC,ctcom*,awk,gawk
```

```
application Mail
```

```
file = sendmail,mail*,*mail,elm,xmh
```

On Windows:

```
application payroll
```

```
file account1,basepay,endreport
```

```
application Office
```

```
file winword* excel*
```

```
file 123* msaccess*
```

If you do not specify a file parameter, all programs that satisfy the other parameters qualify.

Note: The asterisk (*) is the only wild card character supported for **parm** file application qualifiers except for the cmd qualifier.

argv1

The argv1 parameter specifies the processes that are selected for the application based on the value of the PROC_PROC_ARGV1 metric. This is normally the first argument of the command line, except when javaarg is **True** and when this is the class or jar name for Java processes.

argv1 parameter uses the same pattern matching syntax used by **parm** parameters such as file= and user=. Each selection criteria can have asterisks as a wild card match character, and you can have more than one selection on one line separated by commas.

For example:

The following application definition buckets all processes whose first argument in the command line is either -title, -fn, or -display:

```
application = xapps
argv1 = -title,-fn,-display
```

The following application definition buckets a specific Java application (when javaarg=true):

```
application = JavaCollector
argv1 = com.*Collector
```

The following example shows how the argv1 parameter can be combined with the file parameter:

```
application = sun-java
file = java
argv1 = com.sun*
```

cmd

The `cmd` parameter specifies processes for inclusion in an application by their command strings. The command string consists of the program executed and its arguments (parameters). This parameter allows extensive use of wild card characters besides the use of the asterisk character.

Similar to regular expressions, extensive pattern matching is allowed. For a complete description of the pattern criteria, see the UNIX man page for `fnmatch`. You can have only one selection per line, however you can have multiple lines.

The following example shows the use of `cmd` parameter:

```
application = newbie  
cmd = *java *[Hh]ello[Ww]orld*
```

User

The `user` parameter specifies the users (login names) who belong to the application.

The format is:

```
application <application_name>  
file <file_name>  
user [<Domain_Name>]\<User_Name>
```

The domain name in the `user` parameter is optional. To specify the user names of a non-local system, you must specify the domain name.

For example:

```
application test_app  
file test  
user TestDomain\TestUser
```

If you specify the user name without the domain name, all user names from the local system will be considered.

For example:

```
application Prog_Dev  
file vi,xb,abb,ld,lint  
user ted,rebecca,test*
```

You can only use the wild card asterisk (*) to ensure the user names with a similar string of characters prefixed before the asterisk (*) and suffixed after the asterisk (*) belong to the application.

If you do not specify a user parameter, all programs that satisfy the other parameters qualify.

The name in the user parameter is limited to 15 characters in length.

Group

The `group` parameter specifies user group names that belong to an application.

For example:

```
application Prog_Dev_Group2
file vi,xb,abb,ld,lint
user ted,rebecca,test*
group lab, test
```

If you do not specify a group parameter, all programs that satisfy the other parameters qualify.

The name in the `group` parameter is limited to 15 characters in length.

Or

Use the `or` parameter to allow more than one application definition to apply to the same application. Within a single application definition, a process must match at least one of each category of parameters. Parameters separated by the `or` parameter are treated as independent definitions. If a process matches the conditions for any definition, it will belong to the application.

For example:

```
application = Prog_Dev_Group2
user julie
or
user mark
file vi, store, dmp
```

This defines the application (`Prog_Dev_Group2`) that consists of any programs run by the user Julie along with other programs (`vi, store, dmp`) if they are executed by the user Mark.

Priority

You can restrict processes in an application to those belonging to a specified range by specifying values in the priority parameter.

For example:

```
application = swapping
priority 128-131
```

Processes can range in priority from -511 to 255, depending on which platform the Operations Agent is running. The priority can be changed over the life of a process. The scheduler adjusts the priority of time-share processes. You can also change priorities programmatically or while executing.

Note: The **parm** file is processed in the order entered and the first match of the qualifier will define the application to which a particular process belongs. Therefore, it is normal to have more specific application definitions prior to more general definitions.

Application Definition Examples

The following examples show application definitions.

```
application firstthreesvrs
cmd = *appserver* *-option[123]*
```

```
application oursvrs
cmd = *appserver*
user = xyz,abc
```

```
application othersvrs
cmd = *appserver*
cmd = *appsvr*
or
argv1 = -xyz
```

The following is an example of how several programs would be logged using the preceding **parm** file.

Command String	User Login	Application
/opt/local/bin/appserver -xyz -option1	xyz	firstthreesvrs
./appserver -option5	root	othersvrs
./appserver -xyz -option2 -abc	root	firstthreesvrs
./appsvr -xyz -option2 -abc	xyz	othersvrs
./appclient -abc	root	other
./appserver -mno -option4	xyz	oursvrs
appserver -option3 -jkl	xyz	firstthreesvrs
/tmp/bleh -xyz -option1	xyz	othersvrs

Configuring Data Logging Intervals

The default collection intervals used by **oacore** are 60 seconds for process data and 300 seconds for global and all other classes of data. You can override this using the collection interval parameter in the **parm** file.

The values must satisfy the following conditions:

- The collection intervals for process data can be configured between 5 to 60 seconds in steps of 5 seconds. The collection intervals for process data must be a multiple of the subproc interval (see [subprocinterval](#)) and it must divide evenly into the global collection interval.
- The collection interval for global data can be configured to one of the following values: 15, 30, 60 and 300 seconds. The global collection interval must be greater than or equal to process interval, and a multiple of the process collection interval. The global collection interval applies to the global metrics and all non-process metric classes such as filesystem and application.

Note: To log the current PROC_INTEREST for a process, it is suggested to reduce the process collection interval below the default value of 60 seconds. Reducing the process collection interval below the default value is *not* recommended.

PROC_INTEREST is a string containing the reason(s) why the process or thread is of interest based on the thresholds specified in the **parm** file.

Configuring Data Collection for Frames

You can collect performance data from all AIX LPARs available on a single frame by installing the Operations Agent on only one LPAR node. You can also configure the agent to collect performance data from the AIX frame where all the monitored LPARs reside.

You can have the following advantages if you enable the monitoring of frames:

- Collect configuration information:
 - Name and UUID of the frame where monitored LPARs reside
 - Model, serial number, and type of the frame
 - CPU configuration and memory capacity of the frame
- With the additional information, you can analyze resource utilization of the frame

- Use data analysis tools (like Performance Manager) to analyze the CPU consumption ratio of the frame and all the LPARs

Task 1: Configure Passwordless SSH Access

On the Hardware Management Console (HMC) system, you can configure passwordless SSH access between the LPAR node (node where you installed the agent) and the HMC system. After you configure the access to a passwordless SSH for a user, you can run commands using the SSH protocol on the HMC system remotely from the LPAR node without having to provide a password for authentication.

Follow the steps:

Configuring passwordless SSH Access

On the LPAR node

1. Log on to the LPAR node as root.
2. Run the command to create a directory:

```
# mkdir .ssh
```

Note: You can run the following command to remove a .ssh file # `rm -rf .ssh`

3. Run the command to generate a public/private rsa key pair:

```
# ssh-keygen -t rsa
```

4. Specify a file to save the key:

```
./.ssh/<filename>_rsa
```

Your private key is saved in `./.ssh/ <filename>_rsa` and your public key is saved in `./.ssh/ <filename>_rsa.pub`.

5. Run the following commands to view the key:

```
# cat ./ssh/<filename>_rsa
```

```
# cat .ssh/<filename>_rsa.pub
```

The key generated is displayed.

On HMC Machine

1. Log on to the HMC machine with your user credentials.
2. Run the following command to set the passwordless access:

```
mkauthkeys -a 'ssh-rsa<key>'
```

In this instance <key> is the key generated in [step 3](#).

Note: The entire key must be mentioned in a single line.

Verifying passwordless SSH Access

Follow the steps to verify if passwordless SSH access is created for the user:

1. Log on to the LPAR node as root.
2. Run the following command:

```
ssh hmcuser@hmchost.example.domain.com lssyscfg r sys
```

If the command shows the list of frames without prompting for password, then the user is correctly configured and can monitor the frame utilization.

Task 2: Enable Frame Utilization Monitoring on the HMC System

1. Log on to the HMC system as root.
2. Run the following command:

```
chlparrutil -r config -s 60
```

Task 3: Configure the Operations Agent

1. Log on to the LPAR node.
2. Go to the `/var/opt/perf` directory.
3. Create a new file.
4. Open the file with a text editor, and then add the following content to the file:

`<hmcusername>@<hmc_fqdn>`

In this instance:

`<hmcusername>` is the user that was granted passwordless SSH access to the HMC system in ["Task 1: Configure Passwordless SSH Access" on page 182](#).

`<hmc_fqdn>` is the fully qualified domain name of the HMC host.

For example:

```
hmcusername@hmchost.example.domain.com
```

5. Save the file as `hmc` in the `/var/opt/perf` directory.
6. Configure the `logicalsystem` parameter in the `parm` file and start the collection process. For more information about configuring the `parm` file, see ["Using the parm File" on page 151](#).
7. Restart the Performance Collection Component.

The Operations Agent collects the frame-specific performance data at the interval of five minutes.

Enabling the Global and Process System Call Metrics for GlancePlus on Linux

On a Linux machine, you can enable the collection of `GBL_SYSCALL`, `SYSCALL`, and `PROCSYSCALL` performance metric values for GlancePlus. Before you start configuring the metric collection, make sure to stop all the processes that use `FTRACE` on your Linux machine.

Note: The metrics are supported for RHEL/OEL/CentOS 6.1 and above, SLES 11 SP1 and above, Ubuntu 11.10 and above, Debian 6.0 and above.

You can enable the metric collection for GlancePlus by *one* of the following methods:

- [Configuring the Metric Collection using `init_ftrace.sh`](#)
- [Configuring the Metric Collection using manual steps](#)

Configuring the Metric Collection using **`init_ftrace.sh`**

To configure the metric collection on your Linux machine using `init_ftrace.sh`, follow these steps:

1. Log on to the node with the necessary privileges.
2. To stop all the Performance Collection Component processes on your Linux machine, run the following command:

```
/opt/perf/bin/ovpa stop all
```

3. Stop all midaemon processes on your Linux machine. To check the midaemon process, run the following command:

```
ps -ef | grep midaemon
```

If midaemon process is running on your Linux machine, run the following command:

```
killall midaemon
```

This stops all midaemon processes.

4. To configure the metric collection, run the following command:

```
/opt/perf/bin/init_ftrace.sh
```

The command mounts debugfs and enables FTRACE on your Linux machine. If FTRACE is already running on your machine with any other application, message appears as:

Do you want to reset the FTRACE interface for use with midaemon? (Y/N).

You can select *one* of the following:

Y – Resets the FTRACE interface on your Linux machine.

N – Does not configure the metric collection. You cannot view the metrics in GlancePlus.

5. To start all the Performance Collection Component processes along with the midaemon process and metric collection, run the following command:

```
/opt/perf/bin/ovpa start all
```

Use the following screens in GlancePlus to check the performance metrics:

Y - to check the SYSCALL metrics

L - to check the PROCSYSCALL metrics

Note: When GlancePlus is not at System Call screen, the performance of the system is not affected even if debugfs is mounted and FTRACE is enabled. To unmount debugfs from **/sys/kernel/debug**, you can run the command `umount /sys/kernel/debug`.

Configuring the Metric Collection using Manual Steps

To configure the metric collection manually on your Linux machine, follow these steps:

1. Log on to the node with the necessary privileges.
2. To stop all the Performance Collection Component processes on your Linux machine, run the following command:

```
/opt/perf/bin/ovpa stop all
```

3. Stop all midaemon processes on your Linux machine. To check the midaemon process, run the following command:

```
ps -ef | grep midaemon
```

If midaemon process is running on your Linux machine, run the following command:

```
killall midaemon
```

This stops all midaemon processes.

4. Mount debugfs manually on your Linux machine

Note: Skip this step if debugfs is already mounted on your Linux machine. To check if debugfs is already mounted, run the following command:

```
cat /proc/mounts |grep debugfs
```

The command generates *one* of the following outputs:

nodev /sys/kernel/debug debugfs rw,relatime 0 0 - debugfs is already mounted on your machine.

nil – debugfs is not mounted on your machine.

To mount debugfs manually, follow these steps:

- i. Run the following command:

```
mount -t debugfs nodev /sys/kernel/debug
```

The output appears as

nil - debugfs is successfully mounted on your machine.

If debugfs is not successfully mounted on your machine, a failure message appears.

5. Enable FTRACE manually on your Linux machine

Note: Skip this step if FTRACE is already enabled on your Linux machine. To check if FTRACE is already enabled, run the following command:

```
cat /proc/sys/kernel/ftrace_enabled
```

The command generates *one* of the following outputs:

1 - FTRACE is already enabled on your machine.

0 - FTRACE is not enabled on your machine.

To enable FTRACE manually, follow the step:

- i. Run the following command:

```
echo "1" >/proc/sys/kernel/ftrace_enabled
```

The output appears as

nil - FTRACE is successfully enabled on your machine

If FTRACE is not successfully enabled on your machine, a failure message appears.

6. To start all the Performance Collection Component processes along with the `mi.daemon` process and metric collection, run the following command:

```
/opt/perf/bin/ovpa start all
```

Use the following screens in GlancePlus to check the performance metrics:

Y - to check SYSCALL metrics

L - to check the PROCSYSCALL metrics

Note: When GlancePlus is not at System Call screen, the performance of the system is not affected even if `debugfs` is mounted and FTRACE is enabled. To unmount `debugfs` from `/sys/kernel/debug`, you can run the command `umount /sys/kernel/debug`.

Troubleshooting

This section describes the solution or workaround for the common problem encountered while configuring the metric collection.

Problem:

If a `midaemon` process error occurs, see the `/var/opt/perf/status.mi` for more information. You may experience `midaemon` process error if the `midaemon` has run out of room in its shared memory segment.

Solution:

To avoid this problem, you have to clear the unwanted shared memory. To clear the shared memory, follow these steps:

1. Run the following command:

```
ipcs -m | grep 0x0c6629c9
```

The command will generate an output with a variable value in the second data field. To clear the shared memory, run the following command:

```
ipcrm -m <field_value>
```

Example

The following example shows how to clear the shared memory:

```
ipcs -m | grep 0x0c6629c9
```

```
output= 0x0c6629c9 18841617 root 640 8704448 7
```

```
ipcrm -m 18841617
```

2. To restart `midaemon`, run the following command:

```
/opt/perf/bin/ovpa restart
```

Normalizing CPU Metrics on Hyper-Threading or Simultaneous Multi-Threading-Enabled Systems

On a system where hyper-threading/simultaneous multi-threading (HT/SMT) is enabled, the physical CPU supports two or more hardware threads. As a result, multiple software processes or threads can run on the hardware threads simultaneously. On a system with a multi-core processor, multiple threads can run simultaneously on individual cores.

The Performance Collection Component provides you with several CPU-related metrics, which help you analyze and understand the CPU utilization of the monitored system. By default, on all HT/SMT-enabled systems, the Performance Collection Component calculates the values of all CPU-related metrics by normalizing the gathered data against the number of available threads on the monitored

system. When a single thread completely utilizes the entire CPU core, values calculated using the **thread-based normalization** do not always represent the true picture of the CPU utilization.

This version of the Operations Agent introduces a new configuration parameter, `ignore_mt`, which enables you to configure the Performance Collection Component to log the CPU-related data that has been calculated using the **core-based normalization**. Metric values that are calculated with the core-based normalization present a more accurate status of the CPU utilization, and therefore, help you make more effective decisions while analyzing the system's performance.

Logging Metrics Calculated with the Core-Based Normalization

On HP-UX, you can configure the Performance Collection Component to log all CPU-related metrics with core-based normalization. On other platforms, you can configure the Performance Collection Component to calculate the CPU-related metrics of the GLOBAL class using the core-based normalization before logging.

To configure the Performance Collection Component to use the core-based normalization for CPU-related metrics, follow these steps:

On HP-UX

1. Log on to the system with the root privileges.
2. Configure the **parm** file based on your requirement. Do not set the `ignore_mt` flag in the **parm** file.

Note: The value of the `ignore_mt` flag in the **parm** file on HP-UX has no effect on the operation of the Performance Collection Component.

3. Define alarm rules as necessary.
4. Run the following command:

```
/opt/perf/bin/midaemon -ignore_mt
```
5. Start the Operations Agent by running the following command:

```
/opt/OV/bin/opcagt -start
```

The Performance Collection Component starts logging all CPU-related metrics (for all classes) using the core-based normalization.

If you restart the Operations Agent, the Performance Collection Component starts logging the CPU data with the thread-based normalization again and you must configure the Performance Collection

Component once again by using the above steps. To enable the agent to always use the core-based normalization, follow these steps:

1. On the agent node, go to the following location:

`/var/opt/perf`

2. Open the following file with a text editor:

`vppa.env`

3. Set the MIPARMS parameter to `ignore_mt`.
4. Save the file.
5. Restart the agent by running the following command:

```
/opt/OV/bin/opcagt -start
```

On other platforms:

1. Log on to the system with the root or administrative privileges.
2. Configure the **parm** file based on your requirement. Set the `ignore_mt` flag in the **parm** file to **True**.
3. Define alarm rules as necessary.
4. Start the Operations Agent using the following command:

On Windows:

```
%ovinstalldir%bin\opcagt -start
```

On Linux and Solaris:

```
/opt/OV/bin/opcagt -start
```

On AIX:

```
/usr/lpp/OV/bin/opcagt -start
```

The Performance Collection Component starts logging CPU-related metrics for the GLOBAL class using the core-based normalization.

Chapter 6: Using the Utility Program

Utility program is a tool for managing and reporting information on the collection parameters (`parm`) file and the alarm definitions (`alarmdef`) file. You can use the `utility` tool to perform the following tasks:

- Scan the Operations Agent datastore and generate a report showing:
 - Dates and time covered
 - Effects of application and process settings in the collection parameters (**parm**) file
- Check the `parm` file for syntax warnings or errors
- Check the `alarmdef` file for syntax warnings or errors
- Process data stored in the datastore against alarm definitions to detect alarm conditions in historical data

Running the Utility Program

Operations Agent 12.xx supports only command line mode to run the `utility` program. Command options and their associated arguments are passed to the `utility` program through the command line interface. The command line interface works on all platforms allowing the `utility` program to be easily invoked by shell scripts and allowing its input and output to be redirected.

Command line options and arguments are listed in the following table:

Table 1: Command Line Arguments

Command Options	Argument	Description
-f	filename	Specifies the output filename. For example: To specify <code>utilrept</code> as the output file for all utility reports, run the following command: <code>utility -f utilrept -d -xs</code> For more information, see filename .

Command Options	Argument	Description
-D		Enables details for analyze, scan, and parm file checking. For more information, see detail command .
-d		Disables details for analyze and parm file checking. For more information, see detail command .
-T		Generates terse output report formats.
-v		Shows commands as they are executed in the command line.
-xp	parmfile	Checks the syntax of a parm file. For example: To check the syntax of parm file, run the following command: <code>utility -xp</code> For more information, see parmfile command .
-xc	alarmdef	Checks the syntax and sets the alarmdef file name to be used with -xa (analyze command). For example: For more information, see checkdef command .
-xa		Analyzes the data stored in the datastore against the alarmdef file. For example: To view alarm events as well as alarm summary, run the following command: <code>utility -xa -D</code> For more information, see analyze command .

Command Options	Argument	Description
-xs	datastore	Scans the datastore and produces a report. For example: To scan data, run the following command: <code>utility -D -xs</code> For more information, see scan command .
-? or ?		Displays command line syntax.

Utility Scan Report

The utility program's scan command reads the datastore and writes a report on the contents. The utility scan report provides you the information about the disk space utilized by each class of data.

For example:

If you run the command `utility -xs`, the Utility Scan report or the Class Summary report is generated, as shown in the following figure:

CLASSNAME	#	utility -xs	RECORDS	STARTTIME	ENDTIME	HH:MM:SS
SCOPE::CORE	0	0000/00/00	00:00:00	0000/00/00	00:00:00	00:00:00
SCOPE::FILESYSTEM	209150	2015/07/13	15:13:43	2015/07/21	16:01:00	192:47:17
SCOPE::CPU	41830	2015/07/13	15:13:43	2015/07/21	16:01:00	192:47:17
SCOPE::GLOBAL	20915	2015/07/13	15:13:43	2015/07/21	16:01:00	192:47:17
SCOPE::APPLICATION	45116	2015/07/13	15:13:43	2015/07/21	16:01:00	192:47:17
SCOPE::PROCESS	106857	2015/07/13	15:13:43	2015/07/21	16:01:00	192:47:17
SCOPE::DISK	44536	2015/07/13	15:13:43	2015/07/21	16:01:00	192:47:17
SCOPE::NETIF	41830	2015/07/13	15:13:43	2015/07/21	16:01:00	192:47:17
SCOPE::LVOLUME	0	0000/00/00	00:00:00	0000/00/00	00:00:00	00:00:00
SCOPE::TRANSACTION	0	0000/00/00	00:00:00	0000/00/00	00:00:00	00:00:00
SISPI::LOGINS	1690	2015/07/14	16:00:02	2015/07/21	16:00:02	168:00:00
SISPI::GLOBAL	169	2015/07/14	16:00:03	2015/07/21	16:00:03	168:00:00

If you run the command `utility -xs -D`, you will see the following output:

The comparison between the scan reports (version 11.xx and 12.xx) is listed in the section [Utility Scan Report](#) in the chapter *Comparing Operations Agent 12.xx with Earlier Versions*.

Utility Commands

This chapter gives a detailed description about the **utility** program's commands.

analyze

Use the `analyze` command to analyze the data in the datastore based on the alarm definitions in an alarm definitions (**alarmdef**) file and report resulting alarm status and activity. Before issuing the `analyze` command, you can run the `checkdef` command to check the alarm definitions syntax.

The default alarm definitions file for command line mode is **`/var/opt/perf/alarmdef`**.

The `analyze` command allows you to evaluate whether your alarm definitions are a good match against the historical data collected on your system. It also lets you decide if your alarm definitions will generate too many or too few alarms.

You can optionally run the `start`, `stop`, and `detail` commands with `analyze` to customize the `analyze` process.

If you want to see the alarm summary report only, issue the `detail off` command. In the command line mode, `detail off` is the default. To see the alarm events as well as the alarm summary, specify `detail on (-D)` command.

While the `analyze` command is executing, it lists alarm events such as `alarm start`, `end`, and `repeat status`. An alarm summary report shows a count of the number of alarms and the amount of time each alarm was active (`on`). The count includes `alarm starts` and `repeats`, but not `alarm ends`.

For Example

To view the Performance Alarm Summary report, run the following command:

```
utility -xa
```

You will see the following output:

```
Performance Alarm Summary:
```

Alarm	Count	Minutes
1	3	25
3	4	20
4	16	125

```
Analysis coverage using "/var/opt/perf/alarmdef":
```

```
Start: 05/05/2014 00:00:00    Stop: 05/05/2014 16:55:00
Total time analyzed:  Days: 0  Hours: 16  Minutes: 55
```

To view alarm events as well as alarm summary, run the following command:

```
utility -xa -D
```

checkdef

Use the `checkdef` command to check the syntax of the alarm definitions in an alarm definitions file and report warnings or errors that are found. This command also sets and saves the alarm definitions file name for use with the `analyze` command.

For Example

The `checkdef` command checks the alarm definitions syntax in the **alarmdef** file and then saves the name of the **alarmdef** file for later use with the `analyze` command.

```
utility -xc
```

When you are sure that the alarm definitions are correct, you can process them against the data in the datastore using the `analyze` command.

detail

Use the `detail` command to control the level of detail printed in the `analyze`, `parmfile` and `scan` reports. The default is `detail off` in command line mode.

Parameters

detail	on	-D	Prints the effective contents of the parm file as well as parm file errors. Prints complete <code>analyze</code> and <code>scan</code> reports.
	off	-d	In the parm file report, application definitions are <i>not</i> printed. In the <code>scan</code> report, collector collection times, initial parm file global information, and application definitions are <i>not</i> printed. In the <code>analyze</code> report, alarm events and alarm actions are <i>not</i> printed.

For examples of using the `detail` command, see the descriptions of the [analyze](#), [parmfile](#), and [scan](#) commands in this chapter.

help

Use the `help` command to access the **Utility** program's online help facility.

For Example

To view Utility program's command line arguments, run the following command:

```
utility -?
```

filename

Use the `filename` command to specify the output file for all **utility** reports. The contents of the report depends on other commands that are issued after the `filename` command.

For Example

```
utility -f <file name> -d -xs
```

To specify `utilrept` as the output file for all **utility** reports, run the following command:

```
utility -f utilrept -d -xs
```

The `<file name>` parameter in the `filename` command must represent a valid file name to which you have write access. Output appended to the end of existing file. If the file does not exist, then it is created.

To determine the current output file, run the `filename` command without parameters.

If the output file is not the `standard` output, most commands are shown in the output file as they are entered.

Note: On AIX systems, you cannot use the `-f` option with the `utility -xs` command to specify the output file.

parmfile

Use the `parmfile` command to view and check the syntax of the `parm` file settings that are used for data collection. All parameters are checked for correct syntax and errors are reported. After the syntax check is completed, only the applicable settings are reported.

For example

To check the syntax of the **parm** file, run the following command:

```
utility -xp
```

You will see the following output:

```
33 file names used to define applications
1 user names used to define applications
0 group names used to define applications

Parm File: "/var/opt/perf/parm" had 0 Warnings.
```

The `parmfile` command checks the syntax of the current **parm** file and reports any warnings or errors.

To view the list of logging parameter settings, run the following command:

```
utility -xp -v
```

scan

Use the `scan` command to read the datastore and write a report on its contents. The following commands affect the operation of the scan function:

<code>detail</code>	Specifies the amount of detail in the report. The default <code>detail on</code> , specifies full detail.
<code>list</code>	Redirects the output to another file. The default is to list to the standard list device.

For more information, see [detail](#), and [list](#).

For example: To scan data, run the following command:

```
utility -D -xs
```

Chapter 7: Using the Extract Program

Extract program is used to retrieve and analyze the historical data logged in the Operations Agent's datastore. The Extract program performs the export function. It reads data from datastore and exports the results to output files in the ASCII format.

Note: The oacore process must be running for extract program to function.

Running the Extract Program Using Command Line Interface

Operations Agent 12.xx supports only command line mode to run the Extract program. To retrieve data from the datastore run the Extract program using the command line mode. The syntax for the command line interface is similar to standard UNIX command line interfaces on other programs. It fits into the typical UNIX environment by allowing the Extract program to be easily invoked by shell scripts and allowing its input and output to be redirected into UNIX pipes.

Note: During an upgrade from Operations Agent version 11.xx to 12.xx the older data - data stored in the CODA database files, scope log files, and the DSI log files is retained in read-only mode. Extract program is able to read data from both old and new datastore.

The following table lists the command line options and arguments:

Table 3: Command Line Arguments

Command Option	Argument		Description
-b	date	time	Specifies the start date and time of an export function. Syntax: mm/dd/yyyy hh:mm:ss
-B		UNIX <i>start time</i>	Specifies the start time in UNIX format for an export function.
-e	date	time	Specifies the end date and time of an export function. Syntax: mm/dd/yyyy hh:mm:ss
-E		UNIX	Specifies the end time in UNIX format for an export

Table 3: Command Line Arguments, continued

Command Option	Argument	Description
	<i>stop time</i>	function.
-l	logfile	Specifies input log file for DSI data.
-C	classname	Specifies self-describing (DSI) data to export.
gapcdznituyhx GADZNITUYHX		<p>Specifies the type of data to be exported:</p> <p>g = global detail</p> <p>a = application detail</p> <p>p = process detail</p> <p>c = configuration detail</p> <p>d = disk device detail</p> <p>z = lvolume detail</p> <p>n = netif detail</p> <p>i = logical systems detail</p> <p>Note: logical systems detail is not supported on Windows and Linux platforms.</p> <p>t = transaction detail</p> <p>u = CPU detail</p> <p>y = filesystem detail</p> <p>h = Host Bus Adapter (HBA) detail</p> <p>x = core detail</p> <p>G = global summary</p> <p>A = application summary</p> <p>D = disk device summary</p> <p>Z=lvolume summary</p> <p>N = netif summary</p>

Table 3: Command Line Arguments, continued

Command Option	Argument	Description
		<p>I = logical systems summary</p> <p>Note: logical systems summary is not supported on Windows and Linux platforms.</p> <p>T = transaction summary</p> <p>U = CPU summary</p> <p>Y = filesystem summary</p> <p>H = HBA summary</p> <p>X = core summary</p>
-r	export template file	<p>Specifies an export template file for export function.</p> <p>Note: If the path of the export template file contains spaces, ensure you enclose it within double quotation marks.</p>
-f	filename	Specifies the output filename.
-xp	xopt	Exports data to ASCII format files.
-? or ?		Displays command line syntax.

Using the Export Function

The export command reads the data stored in the database and exports the results to output files. Output files can be used in a variety of ways such as reports, custom graphics packages, databases, and user-written analysis programs.

How to export data?

Follow the steps:

1. Specify the class type or data type. See ["Default Performance Metric Classes" on the next page](#)

For example: -a (Application class), -g (Global class)

- Specify the output file to export data.

Note:

If you do not specify the name of the output file, the exported data is written to the default output file. The default output file depends on the class of data that is exported.

For example: To export detail Global class data the default output file `xfrdGLOBAL.asc` is used. See ["Table 5: Default Output Files" on page 203](#)

- Run the export command.
- View the exported data. See ["Output of Exported Files" on page 206](#)

Note: Export template files `reptfile` and `reptall` are furnished with Performance Collection Component. These files are located in the `/var/opt/perf/directory`. `Reptfile` has a predefined set of metrics that are uncommented. It is used by default to perform common export tasks. In the `reptall` file all metrics are commented and it is used to export specific metrics. To customize the export template file, see [Producing a Customized Output File](#).

For example:

To export CPU detail data starting from July 26, 2014, type the following command:

```
extract -u -b 7/26/14 -xp
```

Where

-u	Specifies CPU details.
-b	Specifies starting date of export function.
-xp	Exports data to external format files.

Since no export template file is specified, the default export template file `reptfile` is used.

Default Performance Metric Classes

You can export the following classes of data:

Table 4: Class Types

Class Type	Summarization Interval
global	5, 15, 30, 60, 180, 360, 720, 1440, 10080 minutes.

Table 4: Class Types , continued

application	5, 15, 30, 60, 180, 360, 720, 1440, 10080 minutes.
process	This metrics class is not summarized
disk device	5, 15, 30, 60, 180, 360, 720, 1440, 10080 minutes.
lvolume	5, 15, 30, 60, 180, 360, 720, 1440, 10080 minutes.
transaction	5, 15, 30, 60, 180, 360, 720, 1440, 10080 minutes.
configuration	5, 15, 30, 60, 180, 360, 720, 1440, 10080 minutes.
netif	5, 15, 30, 60, 180, 360, 720, 1440, 10080 minutes.
cpu	5, 15, 30, 60, 180, 360, 720, 1440, 10080 minutes.
filesystem	5, 15, 30, 60, 180, 360, 720, 1440, 10080 minutes.
host bus adapter	5, 15, 30, 60, 180, 360, 720, 1440, 10080 minutes.
core	5, 15, 30, 60, 180, 360, 720, 1440, 10080 minutes.

Note: If you mention any other summarization intervals apart from those listed in the table, then the extract program summarizes to the nearest summarization interval.

Output Files

You can either use the default output files to export data or specify an output file prior to issuing the export command.

- If you specify the output file prior to issuing the `export` command, all data is exported to this single file

For example:

To export the detail global data to the output file named `myout`, run the following command:

```
extract -g -f myout -xp
```

Where

-g	Specifies global details.
-f	Specifies the output filename.
myout	Is the name of the output file.
-xp	Exports data.

- If the output file is set to default, the exported data is separated into different default output files depending on the type of data being exported.

For example:

```
extract -xp -g
```

The export command causes the global class of data to be exported to xfrdGLOBAL.asc file.

The following table lists all the default output files:

Table 5: Default Output Files

Default Output Files	Description
xfrdGLOBAL.asc	Global detail data file
xfrsGLOBAL.asc	Global hourly summary data file
xfrdAPPLICATION.asc	Application detail data file
xfrsAPPLICATION.asc	Application hourly summary data file
xfrdPROCESS.asc	Process detail data file
xfrdDISK.asc	Disk device detail data file
xfrsDISK.asc	Disk device hourly summary data file
xfrdVOLUME.asc	Logical volume detail data file
xfrsVOLUME.asc	Logical volume summary data file
xfrdNETIF.asc	Netif detail data file
xfrsNETIF.asc	Netif summary detail data file
xfrdCPU.asc	CPU detail data file
xfrsCPU.asc	CPU summary data file
xfrdFILESYSTEM.asc	Filesystem detail data file
xfrsFILESYSTEM.asc	Filesystem summary data file
xfrdTRANSACTION.asc	Transaction detail data file
xfrsTRANSACTION.asc	Transaction summary data file
xfrdCONFIGURATION.asc	Configuration data file
xfrdHBA.asc	Host Bus Adapter (HBA) detail data file
xfrsHBA.asc	HBA summary data file

Table 5: Default Output Files, continued

xfrdCORE.asc	Core detail data file
xfrsCORE.asc	Core summary data file

where ext= asc (ASCII).

Note: No output file is created *unless* you specify the type and associated items that match the data in the export template file prior to issuing the export command.

The default file names are created from the data type name. The prefix is either xfrd or xfrs depending on the data. xfrd is used for detailed data and xfrs is used for summary data.

For example, classname = ACCTG_INFO would have **export** file names of:

xfrdACCTG_INFO.asc	detailed ASCII data for ACCT_INFO
xfrsACCTG_INFO.asc	summarized ASCII data for ACCT_INFO

Export Template File Syntax

The export template file can contain all or some of the following information, depending on how you want your exported data to be formatted and what you want the export file to contain:

```

HEADINGS      [ON]
              [OFF]
SEPARATOR= " | "
SUMMARY=VALUE
OUTPUT=Filename
DATA TYPE DATATYPE
METRICS

```

For Example:

```

HEADINGS ON
SEPARATOR="|"
SUMMARY=60

*****
**
** The rest of the file consists of specifics about the metrics to be
** exported for a single data type. Each exportable class may have one
** set of definitions in which the following lines appear.
**
** DATA TYPE specifies the type (class) of data. (required)
**
** OUTPUT      specifies the name of the output file where the exported data is
**              to be written.  OUTPUT may be specified for each class named.
**              (optional)
**
**
** Individual metric names which belong to the class. Metrics will be exported
** in the order listed whenever possible. In order to be exported, metrics
** must not be commented out (that is, they must not contain an asterisk before
** the metric name).
**
** This sample report was generated with a section for each data type which
** is available in the currently open log file. Following the DATA TYPE
** line, each metric which is available for export for this data type is
** listed, but commented out. To select metrics for your report, delete
** the asterisk (*) in the first column.
**
*****
*****

*****
DATA TYPE GLOBAL

**..... Global Record Identification Metrics

* GBL_PROC_SAMPLE
* GBL_SYSTEM_UPTIME_HOURS
* GBL_SYSTEM_UPTIME_SECONDS
* GBL_STATTIME
* GBL_INTERVAL
* GBL_CSWITCH_RATE
* GBL_INTERRUPT_RATE
* GBL_INTERRUPT

```

Parameters

<p>headings</p>	<p>Specifies whether or not to include column headings for the metrics listed in the export file.</p> <p>If headings off is specified, no column headings are written to the file.</p> <p>If headings on is specified, ASCII format places the export title and the column</p>
-----------------	--

	headings for each column of metrics written <i>before</i> the first data records.
separator	Specifies the character that is printed between each field in ASCII formatted data. The default separator character is a vertical bar. Many programs prefer a comma as the field separator. You can specify the separator as any printing or non-printing character.
summary	Specifies the number of minutes of data to be summarized for each summary interval. The default intervals are 5, 15, 30, 60, 180, 360, 720, 1440, and 10080 minutes. If you mention other summarization intervals, the extract program summarizes to the nearest summarization interval.
output	Specifies the name of the output file where the exported data will be written. The output can be specified for each class or data type exported by placing output filename just after the line indicating the data type that starts the list of exported data items. Any valid file name can be specified for output.
data type	Specifies one of the exportable data types: global, application, process, disk, transaction, lvolume, netif, configuration, or DSI class name. This starts a section of the export template file that lists the data items to be copied when this type of data is exported.
metrics	Specifies the metrics to be included in the exported file. Metric names are listed, one per line, in the order you want them listed in the resulting file. You must specify the proper data type before listing items. The same export template file can include item lists for as many data types as you want. Each data type will be referenced <i>only</i> if you choose to export that type of data.

You can have more than one export template file on your system. Each one can define a set of exported file formats to suit a particular need. You use the `report` command to specify the export template file to be used with the `export` function.

Output of Exported Files

The contents of each exported file are:

Names (application, netif, lvolume, or transaction)	If headings on is specified.
Heading line1	If headings on is specified.
Heading line2	If headings on is specified.
first data record	

second data record	
...	
last data record	

Report title and heading lines are not repeated in the file.

For example:

If you export a global class data, you see the following output:

Time Stamp	CPU%	System CPU%	User CPU%	Nice CPU%	Idle CPU%	Wait CPU%	Phys I/Os	Phys Wr	Phys KB	Disk%
09/02/14 12:08:00	100.00	33.33	66.66	0.00	0.00	0.00	0	0.00	0.00	0.00
09/02/14 12:13:00	6.79	3.06	3.72	0.00	93.19	0.00	320	1.00	9.90	0.01
09/02/14 12:18:00	6.93	3.05	3.87	0.00	93.06	0.00	262	0.80	7.60	0.00
09/02/14 12:23:00	6.97	3.21	3.75	0.00	93.02	0.00	237	0.70	6.90	0.00
09/02/14 12:28:00	7.03	3.13	3.89	0.00	92.95	0.00	231	0.70	6.60	0.00
09/02/14 12:33:00	6.94	3.07	3.87	0.00	93.04	0.00	246	0.80	7.10	0.00
09/02/14 12:38:00	6.83	2.89	3.93	0.00	93.15	0.00	229	0.70	6.60	0.00
09/02/14 12:43:00	6.69	2.63	4.06	0.00	93.29	0.00	248	0.80	7.10	0.00
09/02/14 12:48:00	6.62	2.57	4.05	0.00	93.36	0.00	235	0.70	6.80	0.00
09/02/14 12:53:00	6.66	2.91	3.74	0.00	93.33	0.00	248	0.80	7.00	0.00
09/02/14 12:58:00	6.59	2.96	3.62	0.00	93.39	0.00	245	0.80	6.70	0.00
09/02/14 13:03:00	6.94	2.95	3.98	0.00	93.04	0.00	266	0.80	7.40	0.00
09/02/14 13:08:00	6.92	3.14	3.77	0.00	93.07	0.00	252	0.80	6.80	0.00
09/02/14 13:13:00	7.03	3.13	3.89	0.00	92.95	0.00	259	0.80	7.10	0.00
09/02/14 13:18:00	6.99	2.97	4.01	0.00	93.00	0.00	267	0.80	7.20	0.00
09/02/14 13:23:00	6.99	3.09	3.89	0.00	93.00	0.00	259	0.80	7.30	0.00
09/02/14 13:28:00	6.79	2.71	4.07	0.00	93.19	0.00	263	0.80	7.10	0.00
09/02/14 13:33:00	6.64	2.61	4.02	0.00	93.35	0.00	272	0.90	7.50	0.00
09/02/14 13:38:00	6.58	2.61	3.97	0.00	93.40	0.00	259	0.80	6.80	0.00
09/02/14 13:43:00	6.76	2.95	3.80	0.00	93.22	0.06	360	0.90	14.30	0.07
09/02/14 13:48:00	6.93	3.02	3.90	0.00	93.06	0.00	275	0.90	7.30	0.00
09/02/14 13:53:00	7.01	3.04	3.96	0.00	92.98	0.00	272	0.90	7.40	0.00
09/02/14 13:58:00	7.04	3.23	3.80	0.00	92.94	0.00	254	0.80	6.80	0.00
09/02/14 14:03:00	7.04	3.07	3.97	0.00	92.94	0.00	278	0.90	7.70	0.00
09/02/14 14:08:00	7.08	2.99	4.09	0.00	92.90	0.00	248	0.80	6.70	0.00
09/02/14 14:13:00	6.99	2.79	4.19	0.00	92.99	0.00	278	0.90	7.40	0.00
09/02/14 14:18:00	7.06	2.90	4.15	0.00	92.92	0.00	274	0.90	7.30	0.00
09/02/14 14:23:00	6.64	2.64	3.99	0.00	93.34	0.00	271	0.90	7.40	0.00
09/02/14 14:28:00	6.76	2.82	3.93	0.00	93.23	0.00	261	0.80	6.90	0.00
09/02/14 14:33:00	6.92	2.88	4.04	0.00	93.06	0.00	271	0.90	7.40	0.00
09/02/14 14:38:00	7.18	3.09	4.08	0.00	92.80	0.00	264	0.80	7.10	0.00
09/02/14 14:43:00	7.08	3.16	3.91	0.00	92.91	0.00	265	0.80	7.30	0.01
09/02/14 14:48:00	7.18	3.07	4.10	0.00	92.81	0.00	266	0.80	7.20	0.00
09/02/14 14:53:00	7.14	3.12	4.01	0.00	92.85	0.00	275	0.90	7.40	0.00
09/02/14 14:58:00	7.06	2.98	4.07	0.00	92.93	0.00	261	0.80	6.90	0.00
09/02/14 15:03:00	7.18	3.00	4.17	0.00	92.81	0.00	287	0.90	7.70	0.00
09/02/14 15:08:00	6.73	2.67	4.06	0.00	93.25	0.00	268	0.80	7.00	0.00
09/02/14 15:13:00	6.74	2.77	3.97	0.00	93.25	0.00	279	0.90	7.40	0.00
09/02/14 15:18:00	6.83	2.91	3.92	0.00	93.15	0.00	278	0.90	7.40	0.00
09/02/14 15:23:00	7.21	2.99	4.21	0.00	92.78	0.00	277	0.90	7.30	0.00
09/02/14 15:28:00	7.05	3.03	4.01	0.00	92.93	0.00	264	0.80	6.90	0.00
09/02/14 15:33:00	7.17	3.10	4.06	0.00	92.82	0.00	283	0.90	7.60	0.00
09/02/14 15:38:00	7.17	3.05	4.12	0.00	92.81	0.00	258	0.80	6.80	0.00
09/02/14 15:43:00	7.16	3.13	4.03	0.00	92.83	0.00	280	0.90	7.40	0.00

Note:

Even if **gbl_statdate** and **gbl_stattime** metrics are not selected in the reptfile or reptall, the exported files always contain the time stamp in the 24-hour format.

With Operations Agent 12.05, the metric headers displayed in the Extract output are consistent with the metric headers displayed with real time tools such as perfd and Glance.

Producing a Customized Output File

The export template file `repta11` is furnished with Performance Collection Component. This file is located in the `/var/opt/perf/directory`. You can use `repta11` to customize the output file as mentioned below:

Customize Export File:

To customize the export file, follow the steps:

1. In the export template file, uncomment the metrics that you want to export.
2. Save and export the file.

Notes on ASCII Format

ASCII (or text) format is best for copying files to a printer or terminal. The ASCII file format does not enclose fields with double quotes. Therefore, the data in ASCII files will be properly aligned when printed.

Numerical values are formatted based on their range and internal accuracy. Since all fields will not be of the same length, be sure to specify the separator you want to use to start each field.

The user-specified separator character (or the default blank space) separates the individual fields in ASCII formats. Blank spaces, used as separators, can be visually more attractive if you plan to print the report. Other characters can be more useful as separators if you plan to read the export template file with another program.

Using the comma as a separator is acceptable to many applications, but some data items may contain commas *that are not separators*. These commas can confuse analysis programs. The date and time formats can contain different special characters based on the native language specified when you execute the Extract program.

Note: To use a non-printing special character as a separator, enter it into your export template file immediately following the first double quote in the `separator` parameter.

Tip: If you have a printer that supports underlining, you can create a more attractive printout by specifying ASCII format and the vertical bar character (`separator=|`) and then printing the file with underlining turned on.

Extract Commands

This chapter describes the **Extract** program's commands with examples.

Table 6: Extract Commands

Command Options	Description
application	<p>Use the <code>application</code> option to specify the type of application data to be exported.</p> <p>For example:</p> <p>To export detail application data run the following command:</p> <pre>extract -a -r /var/opt/perf/myrept -xp</pre> <p>Since no output file is specified, the application data is exported to <code>xfrdAPPLICATION.asc</code>.</p> <p>The output file contains all application metrics specified in the <code>myrept</code> export template file.</p>
cpu	<p>Use the <code>cpu</code> option to specify the summarization level of CPU.</p> <p>For example:</p> <p>To export CPU detail data that was collected starting from July 26, 2014, run the following command:</p> <pre>extract -u -b 7/26/14 -xp</pre> <p>In this example, since no output file is specified, the CPU data is exported to <code>xfrdCPU.asc</code>.</p> <p>Since no export template file is specified, the default export template file, <code>reptfile</code> is used. All CPU metrics specified in <code>reptfile</code> are included in the output file.</p>
disk	<p>Use the <code>disk</code> option to specify the type of disk device data to be exported.</p> <p>For example:</p> <p>To export disk detail data that was collected starting July 5, 2014, run the following command:</p> <pre>extract -d -b 7/5/14 -xp</pre> <p>In this example, since no output file is specified, the disk detail data is exported to <code>xfrdDISK.asc</code>.</p> <p>Since no export template file is specified, the default export template file, <code>reptfile</code>, is used. All disk metrics specified in <code>reptfile</code> are included in the</p>

Table 6: Extract Commands, continued

	output file.
export	<p>Use the <code>export</code> option to start the process of copying data into an output file.</p> <p>For example:</p> <p>To export the global detail data, run the following command:</p> <pre>extract -xp -g</pre> <p>The global class of data is exported to <code>xfrdGLOBAL.asc</code> file.</p>
filesystem	<p>Use the <code>filesystem</code> option to specify the summarization level of filesystem data to be exported.</p> <p>For example:</p> <p>To export the filesystem detail data that was collected starting July 26, 2014, run the following command:</p> <pre>extract -y -b 7/26/14 -xp</pre> <p>The filesystem data is exported to <code>xfrdFILESYSTEM.asc</code>.</p> <p>Since no export template file is specified, the default export template file, <code>reptfile</code>, is used. All filesystem metrics specified in <code>reptfile</code> are included in the output file.</p>
global	<p>Use the <code>global</code> option to specify the amount of global data to be exported.</p> <p>For example:</p> <p>To export detail global data, run the following command:</p> <pre>extract -g -r /var/opt/perf/myrept -f myout -xp</pre> <p>The global data is exported to the output file named <code>myout</code>.</p> <p>The output file contains all global metrics specified in the <code>myrept</code> export template file.</p>
help	Use the <code>help</code> option to access online help.
lvolume	<p>Use the <code>lvolume</code> option to specify the type of logical volume data to be exported.</p> <p>Note: This option is supported only on HP-UX and Linux systems.</p> <p>For example:</p> <p>To export detail logical volume data, run the following command:</p> <pre>extract -z -r /var/opt/perf/myrept -xp</pre> <p>Since no output file is specified, the logical volume data is exported to</p>

Table 6: Extract Commands, continued

	<p>xfrdVOLUME.asc</p> <p>The output file contains metrics specified in the myrept export template file.</p>
netif	<p>Use the <code>netif</code> option to specify the type of logical network interface (LAN) data to be exported.</p> <p>For example:</p> <p>To export detail network interface data, run the following command:</p> <pre>extract -n -r /var/opt/perf/myrept -xp</pre> <p>Since no output file is specified, the network interface data is exported to <code>xfrdNETIF.asc</code></p> <p>The output file contains metrics specified in the myrept export template file.</p>
process	<p>Use the <code>process</code> option to specify whether or not to export process data.</p> <p>For example:</p> <p>To export detail process data, run the following command:</p> <pre>extract -p -r /var/opt/perf/myrept -xp</pre> <p>Since no output file is specified, the process data is exported to <code>xfrdPROCESS.asc</code></p> <p>The output file contains all process metrics specified in the myrept export template file.</p>
start	<p>Use the <code>start</code> option to set a starting date and time for the export functions. The default starting date is the date 30 full days before the last date in the log file, or if less than 30 days are present, the date of the earliest record in the log file.</p> <p><code>first</code> and <code>last</code> options are supported when used with <code>start (-b)</code> or <code>stop (-e)</code> commands.</p> <p>For example:</p> <ol style="list-style-type: none"> To export detail global data starting from June 5, 1999 8:00 am, run the following command: <pre>extract -g -b 06/05/99 8:00 -f myout -xp</pre> <p>The global data is exported to the output file named <code>myout</code>.</p> <p>Since no export template file is specified, the default export template file, <code>reptfile</code>, is used. All global metrics specified in <code>reptfile</code> are included in the output file.</p> To export only the last global data, run the following command:

Table 6: Extract Commands, continued

	<pre>extract -g -b last-f myout -xp</pre> <p>3. To export first and last global data to xfrdGLOBAL.asc file, run the following command:</p> <pre>extract -g -b first -e last -xp</pre>
stop	<p>Use the <code>stop</code> option to terminate an export function at a specified date and time. The default stopping date and time is the last date and time recorded in the log file.</p> <p>For example:</p> <p>To export the detail global data starting from June 5, 2014 8:00 am and ending at June 5, 2014 5:00 pm, run the following command:</p> <pre>extract -g -b 6/5/14 8:00 -e 6/5/14 17:00 -f myout -xp</pre> <p>The global data is exported to the output file named <code>myout</code>.</p> <p>Since no export template file is specified, the default export template file, reptfile, is used. All global metrics specified in <code>reptfile</code> are included in the output file.</p>
transaction	<p>Use the <code>transaction</code> option to specify the type of transaction data to be exported.</p> <p>For example:</p> <p>To export detail transaction data, run the following command:</p> <pre>extract -t -r /var/opt/perf/myrept -xp</pre> <p>Since no output file is specified, the transaction data is exported to <code>xfrdTRANSACTION.asc</code>.</p> <p>The output file contains all transaction metrics specified in the <code>myrept</code> export template file.</p>
host bus adapter	<p>Use the <code>host bus adapter</code> option to specify the amount of HBA data to be exported.</p> <p>For example:</p> <p>To export HBA detail data that was collected starting from July 26, 2014, run the following command:</p> <pre>extract -h -b 7/26/14 -xp</pre> <p>In this example since no output file is specified, the HBA data is exported to <code>xfrdHBA.asc</code>.</p> <p>Note:</p> <ul style="list-style-type: none"> The default export template file, <code>reptfile</code>, does not contain the HBA

Table 6: Extract Commands, continued

	<p>metrics. To export HBA data add the HBA metrics manually.</p> <ul style="list-style-type: none"> • This option is supported on Linux, Windows and HP-UX systems.
classname and logfile	<p>Use the <code>classname</code> option to specify the class of DSI data to be exported.</p> <p>Syntax</p> <pre>extract -xp -C <dsi class name> -l <log file path></pre>
core	<p>Use the <code>core</code> option to specify the type of bycore data to be exported.</p> <p>Note: This option is supported on Linux, Windows, HP-UX and Solaris systems.</p> <p>For example:</p> <p>To export detail application data run the following command:</p> <pre>extract -x -r /var/opt/perf/myrept -xp</pre> <p>The output file contains all application metrics specified in the myrept export template file.</p>

Chapter 8: Using the cpsh Program

You can use the cpsh program only if you enable the HP Ops OS Inst to Realtime Inst LTU or Glance Pak Software LTU.

The cpsh program provides a new command-line prompt, which enables you to view the real-time metric data collected from the monitored system.

Using the Interactive Mode

You can use the cpsh program in the interactive mode. If you run cpsh command without any options, the cpsh program opens a new command prompt. At this prompt, you can perform different tasks to view the details of the real-time metrics.

To open the cpsh prompt, follow these steps:

1. Log on to a system (with the root or administrative privileges) where the Operations Agent is installed.
2. Run the following command to open the cpsh prompt of the local system:

```
cpsh
```

Run the following command to open the cpsh prompt of a remote system:

```
cpsh -n <system_name>
```

where <system_name> is the fully-qualified domain name of the remote system.

or

```
cpsh -n <ip_address>
```

where <ip_address> is the IP address of the remote system.

Note: While opening the cpsh prompt of a remote system, make sure that the perfd process runs on the remote system. You can prevent other systems from accessing the performance data of the local system through the cpsh utility. For more information, see ["Restricting Access" on page 59](#).

The cpsh prompt opens.

To view metrics data in a well-structured format, run the cpsh command with the `-t` option.

For example:

```
cpsh -t
```

or

```
cpsh -n <system_name> -t
```

3. To view the details of the available commands for use with the cpsh prompt, type **help**.

View Real-Time Metrics

You can view the real-time values of the available metrics from the cpsh prompt. Before you perform any operation at the cpsh prompt, you must set the metric context. The **perfd** daemon and associated utilities process the available data based on the metric classes. Therefore, while using the cpsh utility to view real-time data, you must always set the metric class before performing any operations to view the available data.

To view the real-time values of the metrics of a metric class, follow these steps:

1. At the cpsh prompt, type **class** *<metric_class>*.
2. To list all the currently set metrics for the given class, type **list**. The list of all default metrics for the specified metric class appears.
3. To view values of the metrics that belong to the specified class, type **push** at the cpsh prompt. The cpsh program displays real-time values of the metrics in a tabular format.
4. To go back to the cpsh prompt, press **Ctrl+C**.

Modify a Metric Class

You can add additional available metrics to the list of default metrics for a metric class. To add or delete a metrics from a metric class at the cpsh prompt, follow these steps:

1. Open the cpsh prompt.
2. At the cpsh prompt, type **class** *<metric_class>*.
3. Type **list**. The list of all default metrics for the specified metric class appears.
4. To delete a metric, follow these steps:

At the csh prompt, type **delete** *<metric_name>*.

Type **list**. The list of metrics for the specified metric class does not include the deleted metric.

5. To add a metric to the metric class, follow these steps:

At the csh prompt, type **add** *<metric_name>*.

Type **list**. The list of metrics for the specified metric class includes the newly added metric.

View All the Available Metrics

To view all the available metrics that belong to a metric class, follow these steps:

1. Open the csh prompt.
2. At the csh prompt, type **class** *<metric_class>*.
3. Type **list all**. The list of all available metrics that belong to the specified metric class appears.

Organize a Metric Class

You can reorganize a metric class without performing sequential add and delete operation on the class.

To reorganize a metric class to include the metrics of your choice, follow these steps:

1. Open the csh prompt.
2. At the csh prompt, type **class** *<metric_class>*.
3. Type **init** *<metric_name>* *<metric_name>* *<metric_name>*

The specified metric class incorporates only the metrics specified with the init command.

View Metric Help

You can view the description of each real-time metric from the csh prompt. To view the metric description, follow these steps:

1. Open the csh prompt.
2. Type **class** *<metric_class>*.
3. Type **help** *<metric_name>* at the csh prompt. The details description of the metric appears.

View Summarized Metric Data

For the metrics of the GLOBAL and TABLE classes, you can view summarized data from the cpsh prompt. To view the summarized data, follow these steps:

1. Open the cpsh prompt.
2. At the cpsh prompt, type **class gbl** or **class tbl**.
3. Type **summ <interval>**. In this instance, <interval> is the summarization interval specified in seconds. <interval> must be a multiple of collection interval of **perfd** server to which cpsh is connected.

The cpsh utility displays the following measurements of the values of metrics that belong to the selected metric class:

- o Maximum
- o Minimum
- o Average
- o Standard deviation

Enabling Threshold and Filter Conditions

You can set the threshold and filter options based on your requirement in the Perfd process and view the data available with the set qualifying criteria.

Set the following options:

Threshold - This option is only available for metrics of the GLOBAL class. You can specify the global threshold and view the data available above the set global threshold value.

Filter - The filter option is available for metrics of all the classes.

You can set the conditions as:

- **filter** <metric> <operator> <value>
- **threshold** <metric> <operator> <value>

You can also set both the conditions to view only data that meets the set criteria.

Example 1: List the number of processes where utilization is more that 4 percent.

Use the filter condition to view data.

To view the filtered data, follow these steps:

1. Open the cpsH prompt.
2. At the cpsH prompt, type **class proc**
3. Type **filter proc_cpu_total_util > 4**

The cpsH utility displays the processes which have CPU utilization more than 4 percent.

4. Type **push**

The results appear as:

User	App	Interest	Process	ProcessName
Name	PID	ID Reason	Start Time	CPU Name
root	1028	4	05/07/2013 07:17:35	5.2 find

Example 2: List the number of processes where utilization is more that 4 percent and threshold is set to 1 percent.

Set the threshold option and then the filter option to view data.

To view the filtered data, follow these steps:

1. Open the cpsH prompt.
2. At the cpsH prompt, type **class proc**
3. Type **threshold proc_cpu_total_util > 1**
4. Type **filter proc_cpu_total_util > 3**

The cpsH utility displays the processes where the threshold value is set to more than 1 percent and also the CPU utilization is more than 3 percent.

5. Type **push**

The results appear as:

User	App	Interest	Process	ProcessName
Name	PID	ID Reason	Start Time	CPU Name
root	1028	4	05/07/2013 07:17:35	4.9 find

Chapter 9: Building Collections of Performance Counters on Windows

Performance Collection Component provides access to Windows performance counters that are used to measure system, application, or device performance on your system. You use the Extended Collection Builder and Manager (ECBM) to select specific performance counters to build data collections.

Note: Make sure that the **oacore** process is running before registering a policy and starting a collection.

Building a Performance Counter Collection

To build a collection, choose **ECB-ECM (Extended Collection Builder and Manager)** from the **Start** menu or run the command `mwecbm` using the command prompt to open the **Extended Collection Builder and Manager**.

The Extended Collection Builder and Manager window appears showing a list of Windows objects in the left pane. For instructions on building collections, choose **Help Topics** from the Help menu in the Extended Collection Builder and Manager window.

After you build your collections of Windows performance counters, use the **Extended Collection Manager** pane to register, start, and stop new and existing collections. You can also use the **Manage** option to start, stop, or delete.

Managing a Performance Counter Collection

To manage your data collections, use the **Extended Collection Manager** pane at the bottom of the Extended Collection Builder and Manager. Initially, no collections appear because you must register a collection before you can start collecting data.

After you register or store the collection you created, the Extended Collection Manager pane shows a list of current collections. The Extended Collection Manager pane also displays the state of every collection and enables you to view information (properties) about the collection itself. For instructions

on managing your collections, choose **Help Topics** from the **Help** menu in the Extended Collection Builder and Manager window.

Tips for Using Extended Collection Builder and Manager

The `<InstallDir>\paperdocs\ovpa\C\monxref.txt` file contains a cross-reference of Performance Collection Component metrics to Windows performance counters and commands. Logging data through the Extended Collection Builder and Manager for metrics already collected by Performance Collection Component incurs additional system overhead.

When you use the Extended Collection Builder to create collections, default metric names are assigned to Windows performance counters for internal use with the Performance Collection Component. These default names are generally not meaningful or easy to decipher. To make metric names more meaningful or match them to the metric names supplied by their source application, modify metric attributes by right-clicking or double clicking the metric name after you drag it from left to right pane in the Extended Collection Builder and Manager window. (See the Extended Collection Builder and Manager online help for detailed instructions.)

If you start 61 or more collections, the collections beyond 60 go into error states. This may cause problems with other collections.

If you collect logical disk metrics from a system configured with **Wolfpack**, you must restart the collection in order to collect data for any new disk instances not present when the collection was registered.

Successful deletion of collections requires restarting Performance Collection Component after deleting the collection. If Performance Collection Component is not restarted, you might get an error during the delete operation. This error typically means that some files were not successfully deleted. You may need to manually delete any files and directories that remain after you restart Performance Collection Component.

Extended Collection Builder and Manager may report missing values for some metrics with cache counters. The problem may occur under some circumstances when a metric value gets an overflow. A message is also sent to the ECBM status file. You can resolve the problem by restarting the collection.

Note: ECBM cannot process metric value above 2147483647.

Explanations of Extended Collection Builder and Manager concepts, and instructions on creating and viewing data collections are available in the Extended Collection Builder and Manager online help. To

view online help, from your desktop select **Start --> All Programs --> Operations Agent --> Performance Collection Component --> ECB-ECM Online Help**. You can select **Extended Collections** from the Agent menu in the Performance Collection Component main window and select **Help Topics** from the Help menu in the Extended Collection Builder and Manager window. Online help is available by selecting the **Help** button in the dialog boxes that appear in the Extended Collection Builder and Manager.

Administering ECBM from the Command line

You can run the ECBM program from the <rpmtools>\bin directory using the Windows Command prompt.

- Collections can be managed from the command line using the following command:

```
\<InstallDir>\bin\mwcmcmd.exe
```

- To display various options, type the following command:

```
\<InstallDir>\bin\mwcmcmd /?
```

- To start the stopped collections, type the following command:

```
mwcmcmd start <collection_name(s)>
```

- To start a new collection from a variable-instance policy, type the following command:

```
mwcmcmd start <policy_name> <collect_name> <instance(s)> [options]
```

The following options are available:

-i <sampling_interval>- change sampling interval (seconds)

-l <logfile_path_name> - change default log location

-a <alarm_file> - change the alarm definitions file

- To stop the active collections, type the following command:

```
mwcmcmd stop <collection_name(s)>
```

- To register a policy file, type the following command:

```
mwcmcmd register <policy_file> <collection/policy_name> [options]
```

The following options are available only when registering a fixed-instance policy file:

`-i <sampling_interval>` - change sampling interval (seconds)

`-l <logfile_path_name>` - change default log location

`-a <alarm_file>` - change the alarm definitions file

- To delete a single collection:

`mwcmcmd delete <collection/policy_name> [option]`

The following option is available when deleting a collection:

`-r` - restarts the Performance Collection Component

Note: When deleting a policy or collection, all the associated database files will be deleted. Also, the process **oacore** stops while deleting a collection, and starts automatically after the collection is deleted.

- To delete multiple collections or policies:

`mwcmcmd delete { <collection/policy_name(s)> | -c | -all }`

`-c` - deletes ALL collections

`-a` - deletes ALL collections and policies

Note: When deleting more than one policy or collection at a time, the Performance Collection Component will be automatically restarted and all the associated database files will be deleted.

- To list all registered collections and policies, type the following command:

`mwcmcmd list`

- To list the properties of a collection or policy, type the following command:

`mwcmcmd properties <collection/policy_name>`

- To list the variable-instance objects in a policy, type:

`mwcmcmd objects <policy_name>`

Chapter 10: Overview of Baselineing

Baselineing is a process to compute and provide reference values based on the historical data¹ stored in the metric datastore. To compute baseline data for a specific time period, metric data collected at corresponding time periods from previous weeks is used. Baseline data includes minimum, maximum, average, and standard deviation values. Baseline data is computed at the end of every hour and is stored in the metrics datastore.

Baseline data is used to:

- Provide reference values to monitor daily performance
- Provide reference values to analyze performance trends
- Dynamically set optimal threshold values to analyze the pattern of resource utilization

The baseline data computed by the Operations Agent is used by the SI-AdaptiveThresholdingMonitor policy to monitor performance and resource utilization. For more information, see the topic *Adaptive Thresholding Policies* in *HPE Operations Smart Plug-in for System Infrastructure User Guide*.

Configuring Baseline on the Operations Agent Node

Baselineing is not enabled by default on the Operations Agent node. To configure baselineing on a Operations Agent node, follow the steps:

1. Update the `baseline.cfg` file

Note: `baseline.cfg` is a plain text file used to define the metrics that you want to monitor.

- a. Log on to the Operations Agent node.
- b. Go to the following directory:

On Windows:

`%ovdatadir%`

On HP-UX/Linux/Solaris:

`/var/opt/perf/`

- c. Update the `baseline.cfg` file and define the metrics that you want to monitor in the following

format:

<Class>:<Metric>

In this instance:

- <Class> is the metrics class.
- <Metric> is the metrics for which baseline data must be computed.

Note: Baseline data is computed only for gauge metrics.

For example:

Global:GBL_CPU_TOTAL_UTIL

In this instance:

<Class>	Global
<Metric>	GBL_CPU_TOTAL_UTIL

2. Enable baselineing on the Operations Agent node

- Go to the following directory:

On Windows X64:

```
%ovinstalldir%bin\win64
```

On Windows X86:

```
%ovinstalldir%bin
```

On AIX:

```
/usr/lpp/OV/bin
```

On HP-UX/Linux/Solaris:

```
/opt/OV/bin
```

- Run the following command:

```
ovconfchg -ns oacore -set ENABLE_BASELINE TRUE
```

By default, the value is FALSE.

Baseline data is computed only at the end of every hour. If you want the baseline data to be computed immediately after you enable baseline, you must restart the **oacore** process. Run the following command to restart **oacore**:


```
ovc -restart oacore
```

Note: You can also use the XPL configurations to enable baseline. Follow the steps:

1. On the OM console select **Policy management** → **Policy groups** → **Infrastructure Management** → **v12.0** → **Settings and Thresholds** → **Agent Settings** → **OPC_PERL_INCLUDE_INSTR_DIR**
2. Set the variable `ENABLE_BASELINE` to `TRUE` and then deploy the policy on all desired nodes.

For more information, see the *HPE Operations Smart Plug-in for System Infrastructure User Guide*.

Monitoring Metrics

Note: After baseline data is computed, use the Measurement Threshold policy to monitor the desired metrics. You can use the default Infrastructure SPI policies, such as `SI-ConfigureBaselining` policy and `SI-AdaptivethresholdingMonitor` to monitor the baseline metrics. For more information, see the *HPE Operations Smart Plug-in for System Infrastructure User Guide*.

Based on the classes defined in the `baseline.cfg` file, corresponding baseline classes are created in the Metrics Datastore.

Note: Baseline classes end with `_BASELINE`.

For example:

If you specify the following metrics in the `baseline.cfg` file:

```
Disk:BYDSK_UTIL
```

```
Global:GBL_CPU_TOTAL_UTIL
```

Two baseline classes are created:

```
DISK_BASELINE
```

```
GLOBAL_BASELINE
```

Run the utility `-xs` command to view information about the baseline classes:

# utility -xs				
CLASSNAME	RECORDS	STARTTIME	ENDTIME	HH:MM:SS
SCOPE::DISK	295	2015/07/31 12:30:40	2015/07/31 14:51:00	2:20:20
SCOPE::NETIF	426	2015/07/31 12:30:40	2015/07/31 14:51:00	2:20:20
SCOPE::TRANSACTION	994	2015/07/31 12:30:40	2015/07/31 14:51:00	2:20:20
SCOPE::PROCESS	60262	2015/07/31 12:30:40	2015/07/31 14:52:00	2:21:20
SCOPE::CPU	284	2015/07/31 12:30:40	2015/07/31 14:51:00	2:20:20
SCOPE::CORE	0	0000/00/00 00:00:00	0000/00/00 00:00:00	00:00:00
SCOPE::APPLICATION	579	2015/07/31 12:30:40	2015/07/31 14:51:00	2:20:20
SCOPE::FILESYSTEM	4118	2015/07/31 12:30:40	2015/07/31 14:51:00	2:20:20
SCOPE::GLOBAL	142	2015/07/31 12:30:40	2015/07/31 14:51:00	2:20:20
SCOPE::LVOLUME	0	0000/00/00 00:00:00	0000/00/00 00:00:00	00:00:00
SCOPE::DISK_BASELINE	6	2015/07/31 13:00:00	2015/07/31 14:00:00	1:00:00
SCOPE::GLOBAL_BASELINE	2	2015/07/31 13:00:00	2015/07/31 14:00:00	1:00:00

Baseline Classes

For every metrics specified in the `baseline.cfg` file, sixteen baseline metrics are created.

For example:

If you specify `GBL_CPU_TOTAL_UTIL` in the `baseline.cfg` file, sixteen global baseline metrics are created.

Note: You can run the `ovcodautl-obj` command to view the baseline metrics.

Run the following command to view the baseline metrics created for the `GLOBAL_BASELINE` class:

```
ovcodautl -ds SCOPE -o GLOBAL_BASELINE -obj
```

GLOBAL_BASELINE	ATT	I64	INSTANCEID
GLOBAL_BASELINE	KEY	UTF8	GBL_SYSTEM_ID
GLOBAL_BASELINE	GGE	R64	GBL_CPU_TOTAL_UTIL_MAX_BL
GLOBAL_BASELINE	GGE	R64	GBL_CPU_TOTAL_UTIL_MIN_BL
GLOBAL_BASELINE	GGE	R64	GBL_CPU_TOTAL_UTIL_AVG_BL
GLOBAL_BASELINE	GGE	R64	GBL_CPU_TOTAL_UTIL_STDDEV_BL
GLOBAL_BASELINE	GGE	R64	GBL_CPU_TOTAL_UTIL_STDDEV_BL_LASTHOUR
GLOBAL_BASELINE	GGE	I64	GBL_CPU_TOTAL_UTIL_BL_SAMPLESIZE
GLOBAL_BASELINE	GGE	R64	GBL_CPU_TOTAL_UTIL_MAX_BL_LASTHOUR
GLOBAL_BASELINE	GGE	R64	GBL_CPU_TOTAL_UTIL_MIN_BL_LASTHOUR
GLOBAL_BASELINE	GGE	R64	GBL_CPU_TOTAL_UTIL_AVG_BL_LASTHOUR
GLOBAL_BASELINE	GGE	I64	GBL_CPU_TOTAL_UTIL_BL_SAMPLESIZE_LASTHOUR
GLOBAL_BASELINE	GGE	I64	GBL_CPU_TOTAL_UTIL_S0
GLOBAL_BASELINE	GGE	I64	GBL_CPU_TOTAL_UTIL_S0_LASTHOUR
GLOBAL_BASELINE	GGE	R64	GBL_CPU_TOTAL_UTIL_S1
GLOBAL_BASELINE	GGE	R64	GBL_CPU_TOTAL_UTIL_S1_LASTHOUR
GLOBAL_BASELINE	GGE	R64	GBL_CPU_TOTAL_UTIL_S2
GLOBAL_BASELINE	GGE	R64	GBL_CPU_TOTAL_UTIL_S2_LASTHOUR

The summarization types are Average (Avg), Minimum (Min), Maximum (Max), Standard Deviation (STDDEV), and Sample size (SAMPLESIZE)

The baseline metrics that end with `_BL` include the historic data (along with the last hour data).

The baseline metrics that end with `_BL_LASTHOUR` include only the last hour data.

You can create a Measurement Threshold policy to compare the last hour baseline data with historic data.

For example: You can use Measurement Threshold policy to compare `GBL_CPU_TOTAL_UTIL_AVG_BL_LASTHOUR` with `GBL_CPU_TOTAL_UTIL_AVG_BL`.

The baseline metrics `S0`, `S1` and `S2` are used to compute Standard Deviation.

Note: To stop the baselining functionality on the Operations Agent node, run the following command:

```
ovconfchg -ns oacore -set ENABLE_BASELINE FALSE
```

How to troubleshoot when Baselineing is not functioning?

Follow the steps:

1. **Check if baseline is enabled.**
 - a. Log on to Operations Agent node.
 - b. Go to the following directory:

On 32-bit versions of Windows:

```
"%ovinstalldir%/bin"
```

On 64-bit versions of Windows:

```
"%ovinstalldir%/bin/win64"
```

On AIX:

```
/usr/lpp/OV/bin
```

On HP-UX/Linux/Solaris:

```
/opt/OV/bin
```

- c. Run the following command:

```
ovconfget oacore ENABLE_BASELINE
```

If baseline is enabled then the output is **TRUE**.

2. Check ovcodautl output for baseline metrics.

- a. Go to the following directory:

On 32-bit versions of Windows:

```
"%ovinstalldir%/bin"
```

On 64-bit versions of Windows:

```
"%ovinstalldir%/bin/win64"
```

On AIX:

```
/usr/lpp/OV/bin
```

On HP-UX/Linux/Solaris:

```
/opt/OV/bin
```

- b. Run the following command:

```
ovcodautl -ds SCOPE -o <CLASS>_BASELINE -rawonly
```

For example:

```
ovcodautl -ds SCOPE -o GLOBAL_BASELINE -rawonly
```

Then check the output if the desired baseline metric is listed.

3. Check if the metrics for which baseline data has to be computed is mentioned in the baseline configuration file.

- a. Check if the metric is mentioned in correct format in the following files:

On Windows:

```
%ovdatadir%/baseline.cfg
```

On HP-UX/Linux/Solaris/AIX:

```
/var/opt/perf/baseline.cfg
```

Note:

Metrics for which baseline data has to be computed must be in one of the following formats:

[Baseline]

<Class>:<Metric>

For example:

Global:GBL_CPU_TOTAL_UTIL

4. **Check if metrics mentioned in the baseline configuration file is valid for the platform you are using.**

Note: Baseline data is computed only for gauge metrics.

- a. Go to the following directory:

On 32-bit versions of Windows:

```
"%ovinstalldir%/bin"
```

On 64-bit versions of Windows:

```
"%ovinstalldir%/bin/win64"
```

On AIX:

```
/usr/lpp/OV/bin
```

On HP-UX/Linux/Solaris:

```
/opt/OV/bin
```

- b. Run the following command:

```
ovcodutil -ds SCOPE -o <Class> -m <Metrics> -rawonly
```

In this instance:

<Class> is the metrics class.

<Metrics> is the metric for which baseline data must be computed.

Check if metrics mentioned in the baseline configuration file are listed in the output.

¹Historical Data is the data collected till the previous hour and stored in the metric datastore.

Chapter 11: Overview of Node Resolution

A node can have a single or multiple network interfaces. IP address associated with each network interface may have a host name. Operations Agent selects the host name using the operating system specific APIs based on system configuration (example, `gethostname /getaddrinfo /getnameinfo`). It automatically configures `OPC_NODENAME` parameter that provides the value of the local host name used by the Operations Agent. `OPC_NODENAME` is an internal parameter and you cannot configure it. When a node has multiple IP addresses or host names, you can overwrite the default configuration by setting `OPC_IP_ADDRESS` to a specific IP address. The host name associated with the specified IP address is assigned to `OPC_NODENAME`. In case the IP address is not associated to a host name, you can configure `OPC_NAMESRV_LOCAL_NAME` to a specific host name. You can use the `LOCAL_NODE_NAME` variable available under the `xp1.net` namespace to overwrite the local node name of the system. The defined values of `OPC_NODENAME`, `OPC_IP_ADDRESS` and `LOCAL_NODE_NAME` are used to determine several variables in policy. The message variables are as follows:

- `<$MSG_GEN_NODE>`: Returns the IP address of the node that sends the message.
- `<$MSG_GEN_NODE_NAME>`: Returns the host name of the node that sends the message.
- `<$MSG_NODE>`: Returns the IP address of the node on which the original event took place.
- `<$MSG_NODE_NAME>`: Returns the host name of the node on which the original event took place.

Note: The `OPC_NODENAME`, `OPC_IP_ADDRESS` and `OPC_NAMESRV_LOCAL_NAME` parameters are available under the `eaagt` namespace.

Following are some examples:

Example 1:

Following are the IP addresses and host names in a multiple network interface environment:

```
IP1 abc.test.com abc
```

```
IP2 xyz.test.com xyz
```

where IP1 and IP2 are the two different IP addresses.

abc.test.com and xyz.test.com are the Fully Qualified Domain Names (FQDN).

abc and xyz are the host names.

xyz is the host name of the local system.

The two parameters `OPC_NODENAME`, `<MSG_GEN_NODE_NAME>` automatically set the FQDN value to `xyz.test.com` and `<MSG_GEN_NODE>` parameter automatically sets the IP address to `IP2`. These configurations happen by default because `xyz` is the host name of the local system.

Choose to configure a different IP address (such as `IP1`) using the following command:

```
ovconfchg -ns eaagt -set OPC_IP_ADDRESS IP1
```

As a result of this configuration, the parameters `OPC_NODENAME` and `<MSG_GEN_NODE_NAME>` will set the FQDN value to `abc.test.com`. `<MSG_GEN_NODE>` will set the IP address to `IP1`.

Example 2:

Following are the IP addresses and host names in a multiple network interface environment:

```
IP1 xyz.test.com xyz
```

```
IP2 xyz.test.com xyz
```

where `IP1` and `IP2` are the two different IP addresses.

`xyz.test.com` is the Fully Qualified Domain Name (FQDN).

`xyz` is the host name of the local system.

The `<MSG_GEN_NODE>` parameter automatically sets the IP address to `IP1`. This configuration happens by default because `xyz` is the host name of the local system and `IP1` is the first IP address (as per the order of IP addresses) that is associated with `xyz`.

Choose to configure a different IP address (such as `IP2`) using the following command:

```
ovconfchg -ns eaagt -set OPC_IP_ADDRESS IP2
```

As a result of this configuration, `<MSG_GEN_NODE>` will set the IP address to `IP2`.

Example 3:

In a multiple network interface environment, a Windows IPv6 system has four configured IP addresses in the following order:

```
IP1
```

```
IP2
```

```
IP3
```

```
IP4
```

The IP addresses are associated with host names in the following order; IP1 is not associated to a host name.

```
IP3 xyz.test.com xyz
```

```
IP2 xyz.test.com xyz
```

```
IP4 xyz.test.com xyz
```

By default, the `<$MSG_GEN_NODE>` parameter automatically sets the IP address to IP2 because IP2 is the first configured IP address (as per the order of IP addresses) that is associated to the local system name.

Choose to configure a different IP address using the following command:

```
ovconfchg -ns eaagt --set OPC_IP_ADDRESS <IP_address>
```

As a result of this configuration, `<$MSG_GEN_NODE>` will set to the specific IP address.

Example 4:

Following are the IP address and host names in a single network interface environment:

```
IP1 abc.test.com abc
```

```
IP1 xyz.test.com xyz
```

where IP1 is the IP address.

abc.test.com and xyz.test.com are the Fully Qualified Domain Names (FQDN).

abc and xyz are the host names.

xyz is the host name of the local system.

By default, the two parameters `OPC_NODENAME` and `<$MSG_GEN_NODE_NAME>` automatically set the FQDN value to xyz.test.com because xyz is the host name of the local system.

You can choose to configure the IP address IP1 using the following command:

```
ovconfchg -ns eaagt --set OPC_IP_ADDRESS IP1
```

As a result of this configuration, `OPC_NODENAME` and `<$MSG_GEN_NODE_NAME>` will set the FQDN value to abc.test.com. These configuration happen because abc.test.com is the first FQDN associated with the IP address IP1.

Example 5:

Following are the IP addresses in a multiple network interface environment:

IP1

IP2 xyz.test.com xyz

where IP1 and IP2 are the two different IP addresses.

IP1 is not associated to a host name.

xyz.test.com is the Fully Qualified Domain Name (FQDN).

xyz is the host name of the local system.

By default, the two parameters OPC_NODENAME, <MSG_GEN_NODE_NAME> automatically set the FQDN value to xyz.test.com and <MSG_GEN_NODE> parameter automatically sets the IP address to IP2.

Choose to configure a different IP address (such as IP1) using the following command:

```
ovconfchg -ns eaagt -set OPC_IP_ADDRESS IP1
```

As a result of this configuration, OPC_IP_ADDRESS is set to IP1 and is not associated to a host name. You can configure the host name corresponding to OPC_IP_ADDRESS using the below command:

```
ovconfchg -ns eaagt -set OPC_NAMESRV_LOCAL_NAME <host_name>
```

where <host_name> is any name like abc.test.com. After running both the commands, the two parameters OPC_NODENAME and <MSG_GEN_NODE_NAME> will set the FQDN value to abc.test.com.

Note: The parameter OPC_IP_ADDRESS set in a NAT environment behaves in the same manner as described in the above examples.

Example 6:

Following are the IP address and host names in a single network interface environment:

IP1 abc.test.com abc xyz

IP1 xyz.test.com xyz

where IP1 is the IP address

abc.test.com and xyz.test.com are the Fully Qualified Domain Names (FQDN).

abc and xyz are the host names.

xyz is the host name of the local system.

By default, the two parameters `OPC_NODENAME` and `<$MSG_GEN_NODE_NAME>` automatically set the FQDN value to `abc.test.com`. In case the external event originates from node with name `xyz` then the `<$MSG_NODE_NAME>` parameter automatically sets to `xyz`.

You can choose to configure `OPC_NODENAME`, `<$MSG_GEN_NODE_NAME>`, and also `<$MSG_NODE_NAME>` to the FQDN value `xyz.test.com` by performing one of the following options:

- Interchange the order of the entries in hosts file.
- Remove the alias **xyz** from the first entry.

Example 7:

Following are the IP addresses and host names of two systems:

```
IP1  abc.test.com  abc
```

```
IP2  xyz.test.com  xyz
```

where IP1 and abc are the IP address and host name of remote system.

IP2 and xyz are the IP address and host name of local system.

`abc.test.com` and `xyz.test.com` are the Fully Qualified Domain Names (FQDN).

When an event originates from IP1 (for example, a trap originates from the remote node abc) the following configurations happen by default:

- Parameter `<$MSG_NODE>` sets to IP1 as `<$MSG_NODE>` returns the IP address of the node on which the original event took place.
- Parameter `<$MSG_NODE_NAME>` sets to abc as `<$MSG_NODE_NAME>` returns the host name of the node on which the original event took place.
- Parameter `<$MSG_GEN_NODE>` sets to IP2 as `<$MSG_GEN_NODE>` returns the IP address of the node that sends the message.
- Parameter `<$MSG_GEN_NODE_NAME>` sets to xyz as `<$MSG_GEN_NODE_NAME>` returns the host name of the node that sends the message.

When the `LOCAL_NODE_NAME` variable is configured, the Operations Agent uses `LOCAL_NODE_NAME` as the hostname of the local system.

To configure a different local host name (such as xyz), use the following command:

```
ovconfchg -ns xp1.net -set LOCAL_NODE_NAME xyz
```

As a result of this configuration, the parameters OPC_NODENAME and <MSG_GEN_NODE_NAME> will set the LOCAL_NODE_NAME variable value to xyz.

Chapter 12: Logging and Tracing

You can diagnose and troubleshoot problems in the Operations Agent by using the logging and tracing mechanisms. The Operations Agent stores error, warning, and general messages in log files for easy analysis.

The tracing mechanism helps you trace specific problems in the agent's operation; you can transfer the trace files generated by the tracing mechanism to HPE Support for further analysis.

Logging

The Operations Agent writes warning and error messages and informational notifications in the `System.txt` file on the node. The contents of the `System.txt` file reveal if the agent is functioning as expected. You can find the `System.txt` file in the following location:

On Windows:

`%ovdatadir%\log`

On HP-UX/Linux/Solaris:

`/var/opt/OV/log`

In addition, the Operations Agent adds the status details of the Performance Collection Component and coda in the following files:

On Windows:

- `%ovdatadir%\log\System.txt`
- `%ovdatadir%\status.perfalarm`
- `%ovdatadir%\status.ttd`
- `%ovdatadir%\status.mi`
- `%ovdatadir%\status.perfd-<port>`
- `%ovdatadir%\hpcs\hpcstrace.log`

Tip: In this instance, *<port>* is the port used by `perfd`. By default, `perfd` uses the port 5227. To change the default port of `perfd`, see [Configuring the RTMA Component](#).

On HP-UX/Linux/Solaris:

- /var/opt/OV/log/System.txt
- /var/opt/perf/status.perfalarm
- /var/opt/perf/status.ttd
- /var/opt/perf/status.mi
- /var/opt/perf/status.perfd
- /var/opt/OV/hpcs/hpcstrace.log

Configure the Logging Policy

The System.txt file can grow up to 1 MB in size, and then the agent starts logging messages in a new version of the System.txt file. You can configure the message logging policy of the Operations Agent to restrict the size of the System.txt file.

To modify the default logging policy, follow these steps:

1. Log on to the node.
2. Go to the following location:
On Windows:
`%ovdatadir%conf\xpl\log`
On HP-UX/Linux/Solaris:
`/var/opt/OV/conf/xpl/log`
3. Open the log.cfg file with a text editor.
4. The BinSizeLimit and TextSizeLimit parameters control the byte size and number of characters of the System.txt file. By default, both the parameters are set to 1000000 (1 MB and 1000000 characters). Change the default values to the desired values.
5. Save the file.
6. Restart the Operations Monitoring Component with the following commands:
 - a. `ovc -kill`
 - b. `ovc -start`

Tracing

Before you start tracing an Operations Agent application, you must perform a set of prerequisite tasks, which includes identifying the correct application to be traced, setting the type of tracing, and generating a trace configuration file (if necessary).

Command line options and arguments for tracing are listed in the following table:

Command Line Options

Command Options	Description
-host	Specifies the host name of the machine to connect to. Syntax: -host <host_name>
-off	Turns off all traces.
-cf -configuration	Specifies the name of the configuration file for dynamic changes. Syntax: -cf <config_file_name>
-vc -viewconfig	Displays the trace configuration of all the applications.
-app -application	Specifies the list of application names to be configured.
-cm -component	Specifies the list of component names to be configured.
-sink	Specifies the trace output file name. Syntax: -sink <filename>
-gc -generate_configuration	Specifies the name of the configuration file to be generated. Syntax: -gc <file_name>
-rd -resetdefault	Resets all the .ini files to its default content.
-h -help	Displays command line syntax.
-version	Displays the tool version number.

Before you begin tracing an Operations Agent process, perform the following tasks:

1. [Identify the Application](#)
2. [Set the Tracing Type](#)
3. *Optional.* [Create the Configuration File](#)

Identify the Application

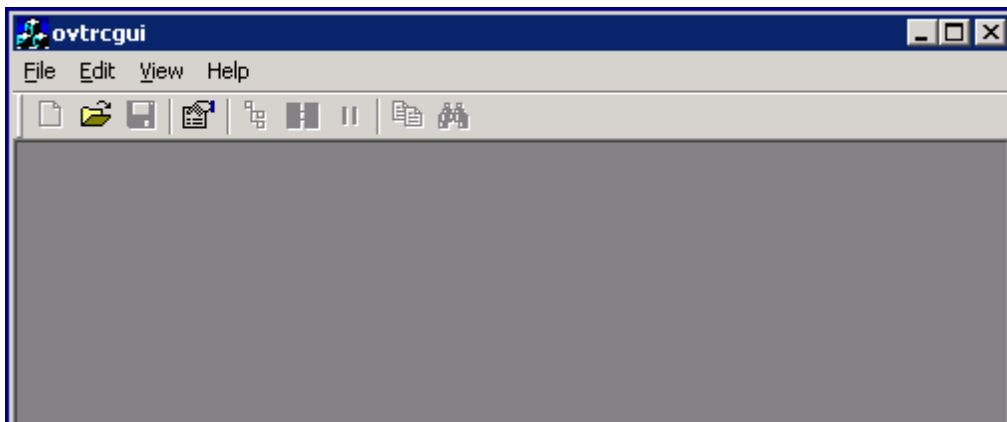
On the managed system, identify the HPE Software applications that you want to trace. Use the `ovtrccfg -vc` option to view the names of all trace-enabled applications and the components and categories defined for each trace-enabled application.

Alternatively, you can use the `ovtrcgui` utility to view the list of trace-enabled applications.

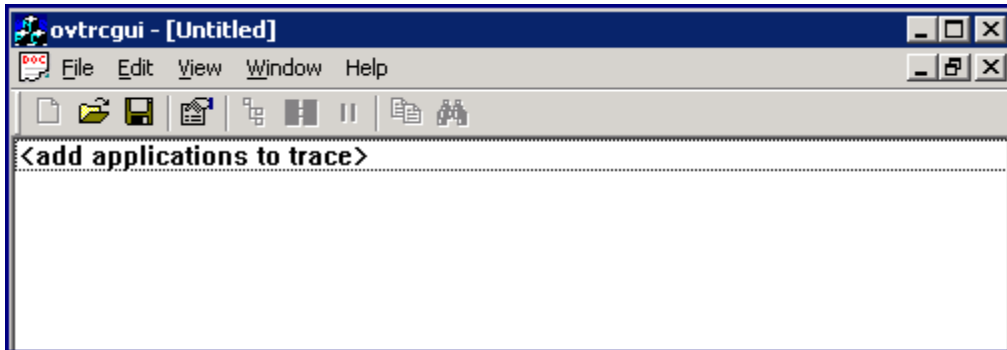
Note: `ovtrcgui` utility is supported only on Windows systems.

To use the `ovtrcgui` utility to view the list of trace-enabled applications, follow these steps:

1. Run the `ovtrcgui.exe` file from the `%OvInstallDir%\support` directory. The `ovtrcgui` window opens.

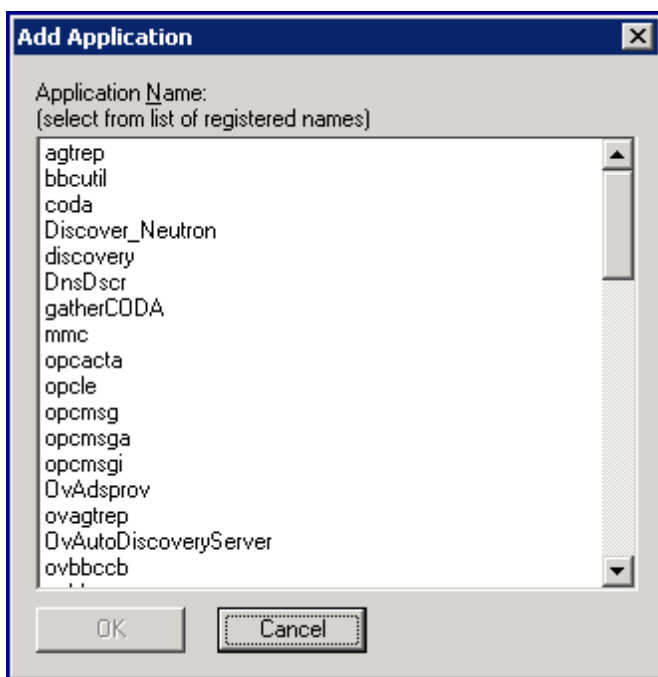


2. In the `ovtrcgui` window, click **File** → **New** → **Trace Configuration**. A new trace configuration editor opens.



3. In the `ovtrcgui` window, click **Edit** → **Add Application**. Alternatively, right-click on the editor,

and then click **Add Application**. The Add Application window opens.



The Add Application window presents a list of available trace-enabled applications.

Set the Tracing Type

Before you enable the tracing mechanism, decide and set the type of tracing that you want to configure with an application. To set the type of tracing, follow these steps:

Determine the type of tracing (static or dynamic) you want to configure, and then follow these steps:

1. Go to the location `<data_dir>/conf/xpl/trc/`
2. Locate the `<application_name>.ini` file. If the file is present, go to step 3 below. If the `<application_name>.ini` file is not present, follow these steps:
 - o Create a new file with a text editor.
 - o Add the following properties to the file in the given order: DoTrace, UpdateTemplate, and DynamicTracing.

Tip: Do not list the properties in a single line. List every property in a new line. For example:

```
DoTrace=
```



```
UpDateTemplate=
```

```
DynamicTracing=
```

- Save the file.
3. Open the `<application_name>.ini` file with a text editor.
 4. To enable the static tracing, make sure that the `DoTrace` property is set to ON and the `DynamicTracing` property is set to OFF.
 5. To enable the dynamic tracing, make sure that the `DoTrace` and `DynamicTracing` properties are set to ON.
 6. Make sure that the `UpdateTemplate` property is set to ON.
 7. Save the file.

For the dynamic trace configuration, you can enable the tracing mechanism even after the application starts. For the static trace configuration, you must enable the tracing mechanism before the application starts.

Introduction to the Trace Configuration File

Syntax

```
TCF Version <version_number>
```

```
APP: "<application_name>"
```

```
SINK: File "<file_name>" "maxfiles=[1..100];maxsize=[0..1000];"
```

```
TRACE: "<component_name>" "<category_name>" <keyword_list>
```

Each line of the syntax is described in detail in the following sections.

Create the Configuration File

If you want to enable the tracing mechanism without the help of a configuration file, skip this section and go to the section *Enabling Tracing and Viewing Trace Messages with the Command-Line Tools* below.

You can create the trace configuration file with the command-line tool `ovtrccfg`, with a text editor, or with the `ovtrcgui` utility (only on Windows nodes).

Enabling Tracing and Viewing Trace Messages with the Command-Line Tools

The procedure outlined below covers the general sequence of steps required to enable tracing. To enable the tracing mechanism, follow these steps:

1. Enable tracing dynamically for the specific applications mentioned in the configuration file using the command `ovtrccfg`.

```
/opt/OV/support/ovtrccfg -cf <configuration_file_name>
```

where `<configuration_file_name>` is the name of the trace configuration file created in the section above *Create the Configuration File*.

Note: If you do not want to use a trace configuration file, you can enable tracing with the following command:

```
/opt/OV/support/ovtrccfg -app <application>[-cm <component>]
```

2. If you configure the static tracing mechanism, start the application that you want to trace.
3. Run the application specific commands necessary to duplicate the problem that you want to trace. When the desired behavior has been duplicated, tracing can be stopped.
4. Make a trace monitor request using `ovtrcmon`.
To monitor trace messages, run one of the following commands or a similar command using additional `ovtrcmon` command options:

- To monitor trace messages from `/opt/OV/bin/trace1.trc` and direct trace messages to a file in the text format:

```
/opt/OV/support/ovtrcmon -fromfile /opt/OV/bin/trace1.trc -tofile  
/tmp/traceout.txt
```

- To view trace messages from `/opt/OV/bin/trace1.trc` in the verbose format:

```
/opt/OV/support/ovtrcmon -fromfile /opt/OV/bin/trace1.trc -verbose
```

- To view trace messages from `/opt/OV/bin/trace1.trc` in the verbose format and direct the trace message to a file:

```
/opt/OV/support/ovtrcmon -fromfile /opt/OV/bin/trace1.trc -short >  
/tmp/traces.trc
```

5. To stop or disable tracing using `ovtrccfg`, run the following command:
`/opt/OV/support/ovtrccfg -off`
6. Collect the trace configuration file and the trace output files. Evaluate the trace messages or package the files for transfer to HPE Software Support Online for evaluation. There may be multiple versions of the trace output files on the system. The `Maxfiles` option allows the tracing mechanism to generate multiple trace output files. These files have the extension `.trc` and the suffix `n` (where `n` is an integer between 1 and 99999).

For example:

```
SINK: File "coda.trc" "force=0;maxfiles=10;maxsize=10;"
root@/var/opt/OV/tmp/trc->lrt
total 93092
-rw-r----- 1 root bin 10475620 Aug 12 15:30 coda_00013.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00014.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00015.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00016.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00017.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00018.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00019.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00020.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:31 coda_00021.trc
-rw-r----- 1 root bin 1027187 Aug 12 15:31 coda_00022.trc
root@/var/opt/OV/tmp/trc->
```

Enabling Tracing and Viewing Trace Messages with the Tracing GUI

On the Windows nodes, you can use the `ovtrcgui` utility to configure tracing and view the trace messages.

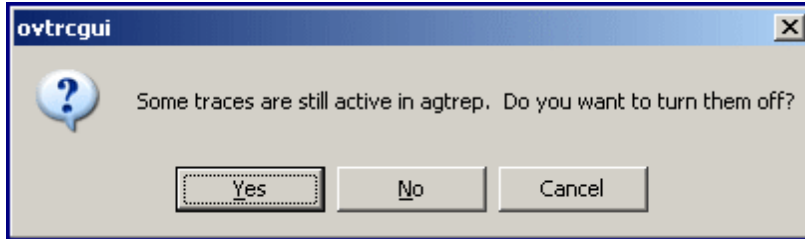
Enable the Tracing Mechanism

To enable the tracing mechanism with the `ovtrcgui` utility and without the help of a trace configuration file, follow these steps:

1. Follow Step 1 through Step 6 in ["Using the Tracing GUI" on page 249](#).
2. Close the trace configuration editor.

3. Click **No** when prompted to save changes to Untitled.

The following message appears:



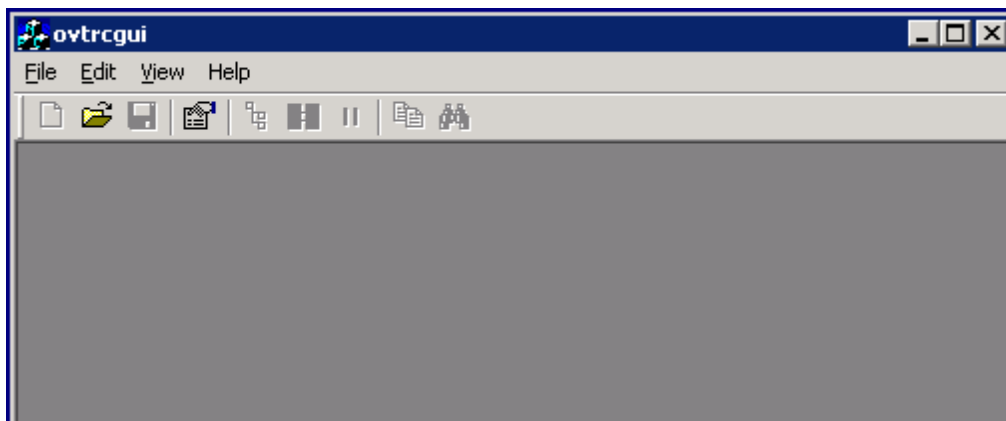
4. Click **No**. If you click **Yes**, the ovtrcgui utility immediately disables the tracing mechanism.

To enable the tracing mechanism with the ovtrcgui utility using a trace configuration file, go to the location on the local system where the trace configuration file is available, and then double-click on the trace configuration file. Alternatively, open the ovtrcgui utility, click **File** → **Open**, select the trace configuration file, and then click **Open**.

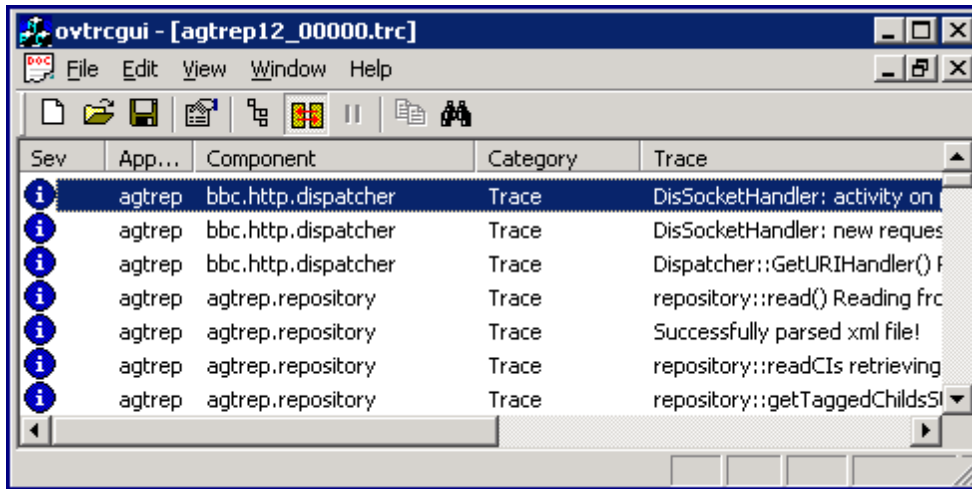
View Trace Messages

To view the trace output files with the ovtrcgui utility, follow these steps:

1. Run the ovtrcgui.exe file from the %OvInstallDir%\support directory. The ovtrcgui window opens.

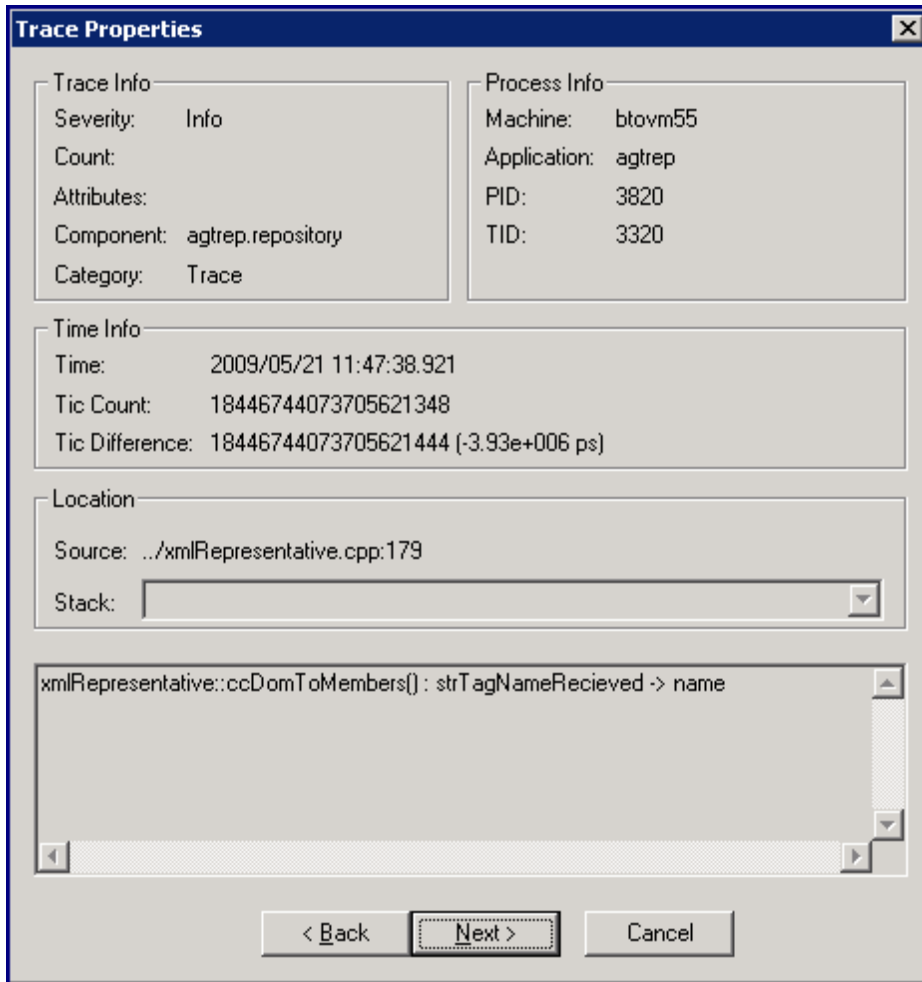


2. Click **File** → **Open**. The Open dialog box opens.
3. Navigate to the location where the trace output file is placed, select the .trc file, and then click **Open**. The ovtrcgui utility shows the contents of the .trc file.



Every new line in the .trc file represents a new trace message.

4. Double-click a trace message to view the details. The Trace Properties window opens.



The Trace Properties window presents the following details:

- Trace Info:
 - *Severity*: The severity of the trace message.
 - *Count*: The serial number for the message.
 - *Attributes*: The attribute of the trace message.
 - *Component*: Name of the component that issues the trace message.
 - *Category*: An arbitrary name assigned by the traced application.
 - Process Info:
 - *Machine*: Hostname of the node.
 - *Application*: Name of the traced application.
 - *PID*: Process ID of the traced application.
 - *TID*: Thread ID of the traced application.
 - Time Info:
 - *Time*: The local-equivalent time and date of the trace message.
 - *Tic count*: A high-resolution elapsed time.
 - *Tic difference*:
 - Location
 - *Source*: Line number and file name of the source generating the trace.
 - *Stack*: A description of the calling stack in the traced application.
5. Click **Next** to view the next trace message.
 6. After viewing all the trace messages, click **Cancel**.




Use the Trace List View

By default, the `ovtrcgui` utility displays the trace messages for a trace file in the trace list view. The trace list view presents the trace messages in a tabular format.

The trace list view presents every trace message with the help of the following columns:

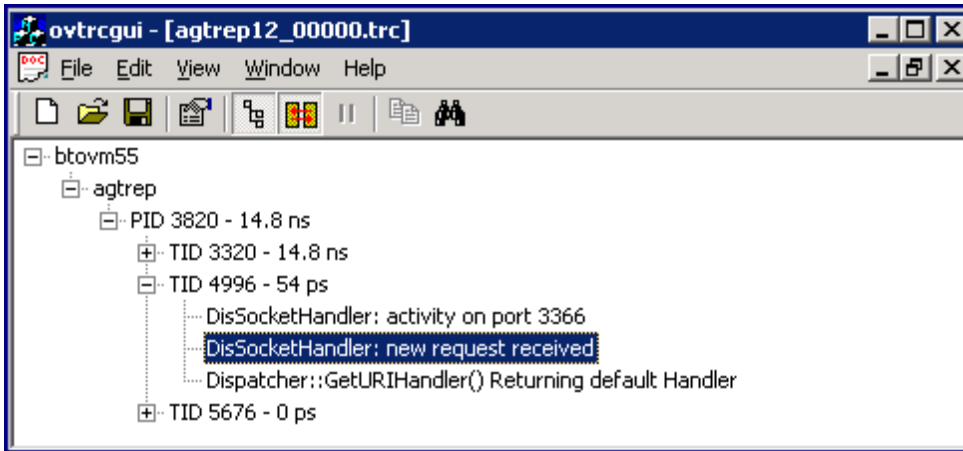
Table 13: Trace List View


Column	Description
Severity	Indicates the severity of the trace message. The view uses the following icons to display the severity of the messages:

Column	Description
	Info 
	Warning 
	Error 
Application	Displays the name of the traced application.
Component	Displays the name of the component of the traced application that generated the trace message.
Category	Displays the category of the trace message.
Trace	Displays the trace message text.

Use the Procedure Tree View

You can view the trace messages in a structured format in the procedure tree view. The procedure tree view sorts the messages based on the process IDs and thread IDs and presents the data in the form of a tree view.

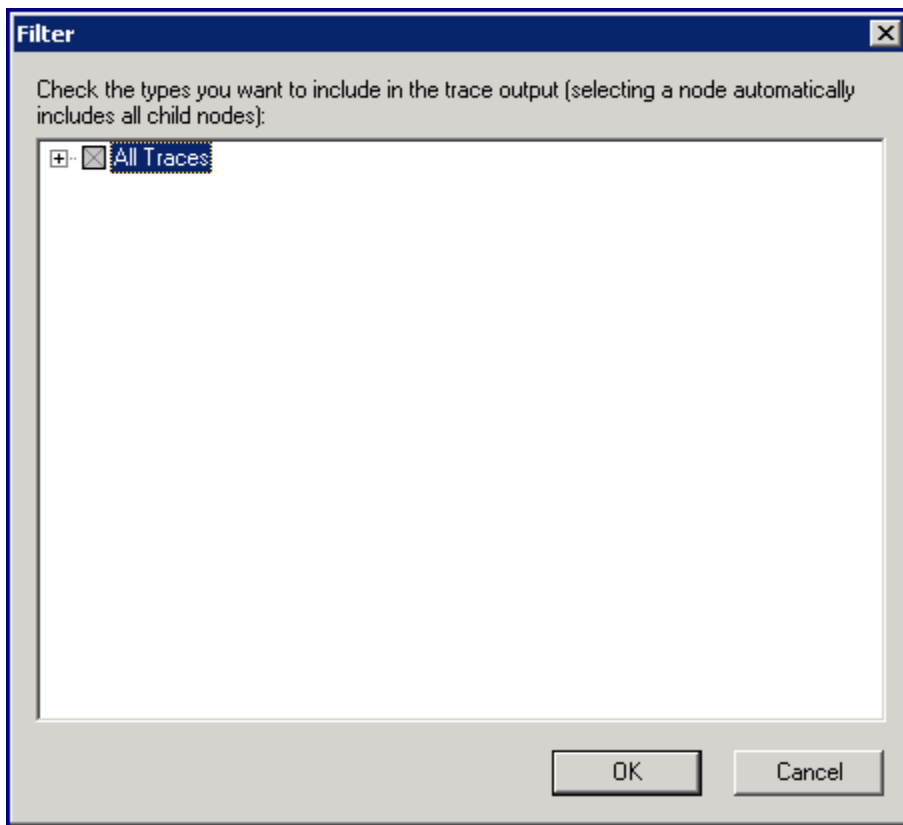


You can expand the process IDs and thread IDs to view trace messages. To go back to the trace list view, click .

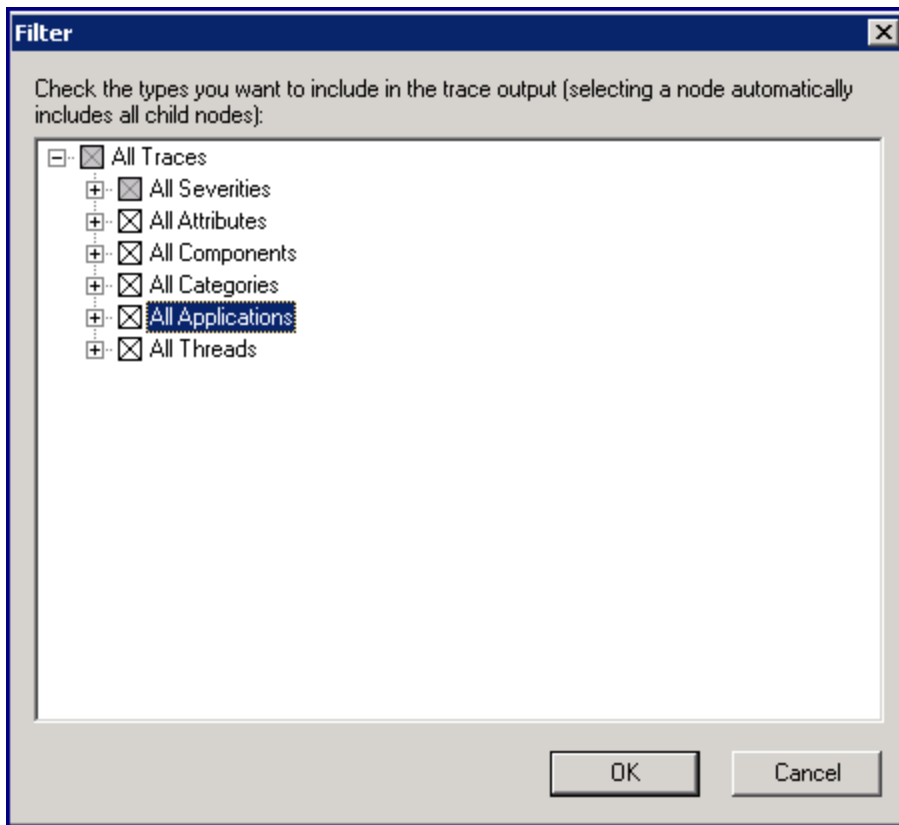
Filter Traces

The `ovtrcgui` utility displays all the trace messages that are logged into the trace output files based on the configuration set in the trace configuration file. You can filter the available messages to display only the messages of your choice in the `ovtrcgui` console. To filter the available trace messages, follow these steps:

1. In the `ovtrcgui` console, click **View** → **Filter**. The Filter dialog box opens.



2. Expand **All Traces**. The dialog box lists all filtering parameters in the form of a tree.

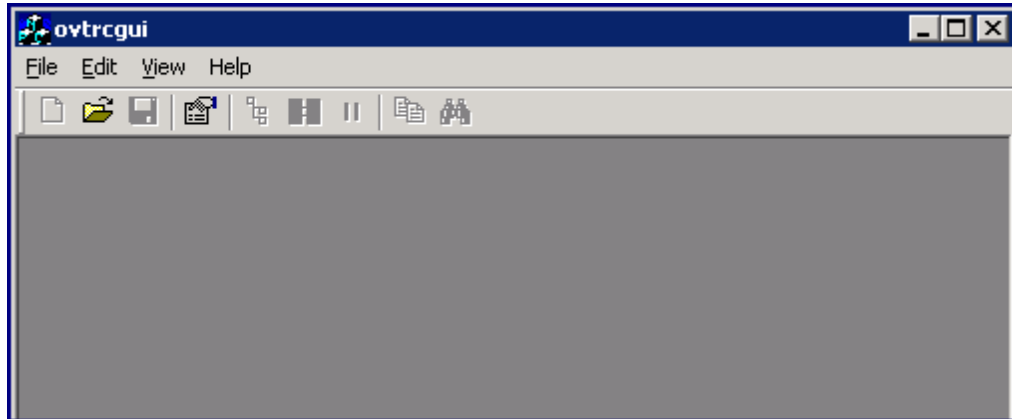


3. Expand the parameters to make selections to filter the trace messages.
4. Click **OK**. You can see only the filtered messages in the ovtrcgui console.

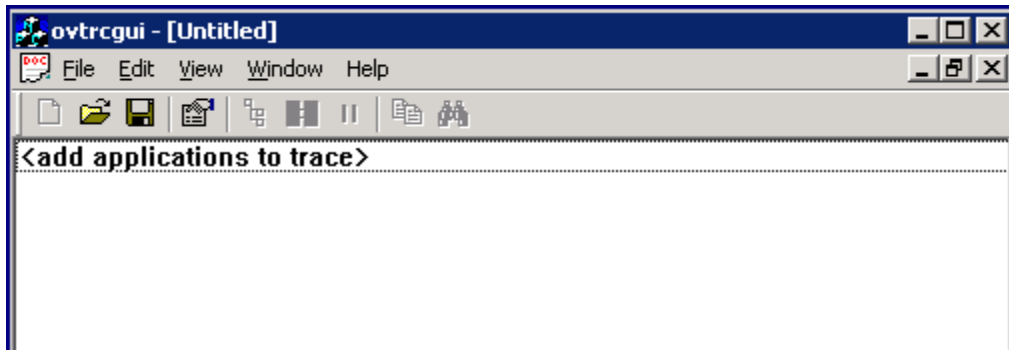
Using the Tracing GUI

On the Windows nodes, you can use the tracing GUI (the ovtrcgui utility) to create the trace configuration file. To use this utility and create a trace configuration file, follow these steps:

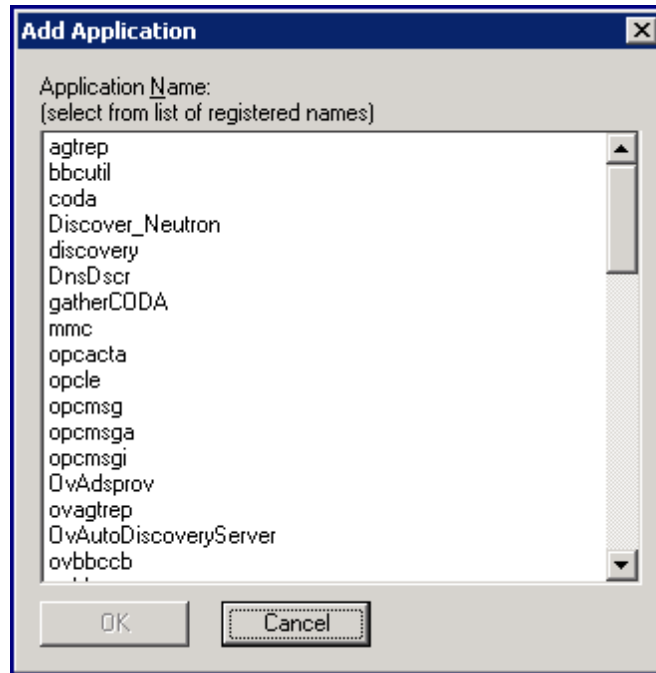
1. Run the ovtrcgui.exe file from the %OvInstallDir%\support directory. The ovtrcgui window opens.



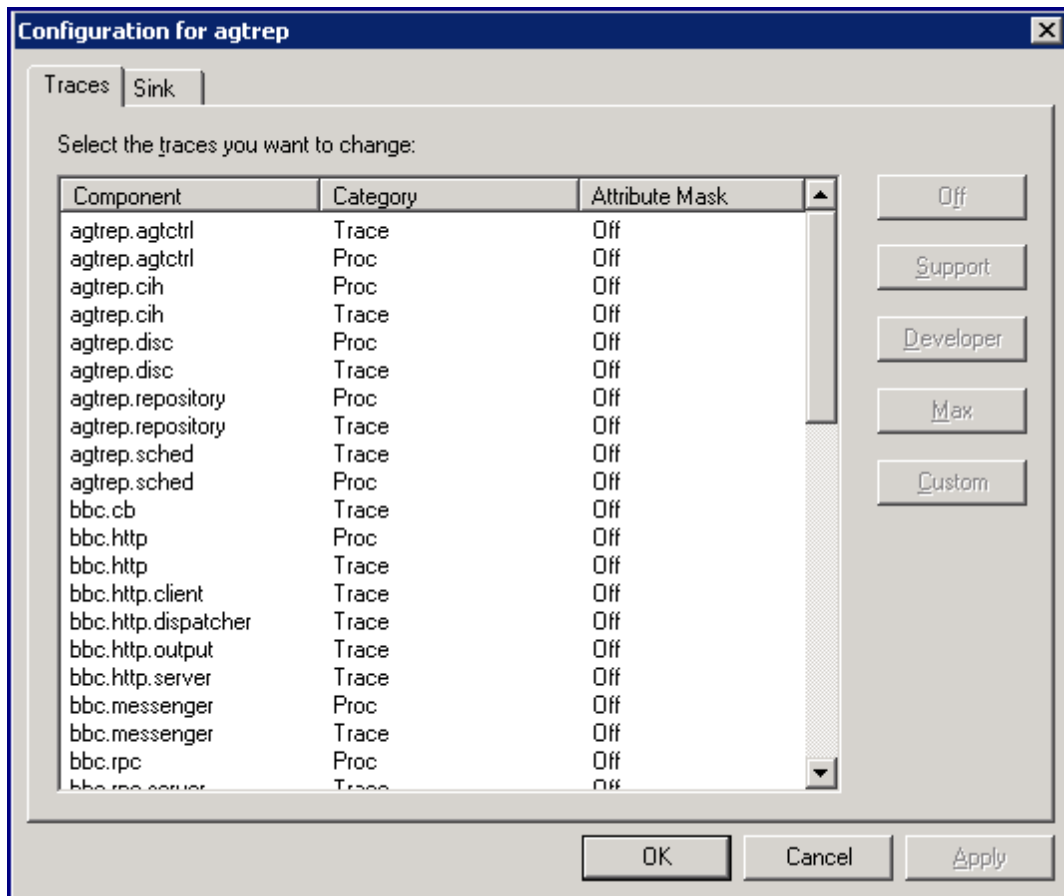
2. In the ovtrcgui window, click **File** → **New** → **Trace Configuration**. A new trace configuration editor opens.



3. In the ovtrcgui window, click **Edit** → **Add Application**. Alternatively, right-click on the editor, and then click **Add Application**. The Add Application window opens.

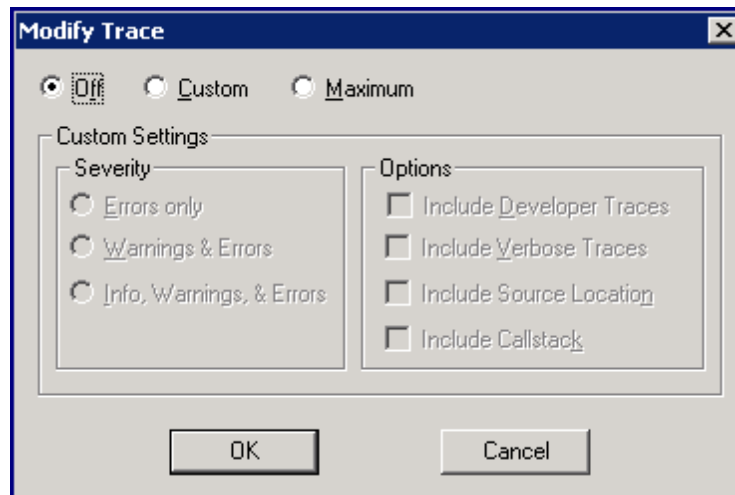


4. Select the application that you want to trace, and then click **OK**. The Configuration for *<application>* window opens.



The Traces tab of the Configuration for *<application>* window lists all the components and categories of the selected application. By default, tracing for all the components and categories are set to Off.

5. In the Traces tab, click a component and category pair, and then click one of the following buttons:
 - **Support:** Click this to gather trace messages marked as informational notifications.
 - **Developer:** Click this to gather trace messages marked as informational notifications along with all developer traces.
 - **Max:** Click this to set the maximum level of tracing.
 - **Custom:** When you click Custom, the Modify Trace window opens.



In the Modify Trace window, select the custom options, select the trace levels and options of your choice, and then click **OK**.

Tip: In the Configuration for *<application>* window, you can click **Off** to disable tracing for a component-category pair.

6. Click **OK**.
7. Go to the Sink tab.
8. Specify the name of the trace output file in the File Name text box. The file extension must be `.trc`.

Tip: Specify the complete path for the `.trc` file.

9. Specify the number of historic files from the drop-down list (see [maxfiles](#)).
10. Specify the maximum file size from the drop-down list (see [maxsize](#)).
11. Click **Apply**.
12. Click **OK**.

Note: The `ovtrcgui` utility enables the tracing mechanism when you click **OK**.

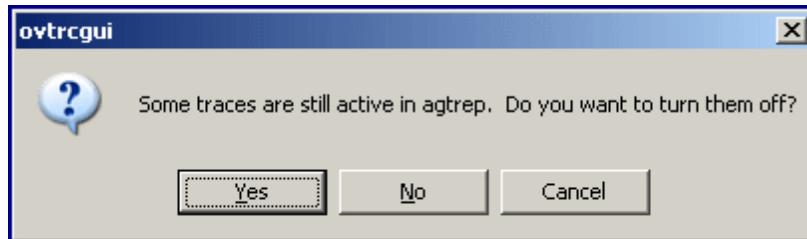
13. Click **File** → **Save**. The Save As dialog box opens.

In the Save As dialog box, browse to a suitable location, specify the trace configuration file name with the `.tcf` extension in the File name text box, and then click **Save**.

14. The `ovtrcgui` utility saves the new trace configuration file into the specified location with the specified name and enables the tracing mechanism based on the configuration specified in the file.

You can open the trace configuration file with the `ovtrcgui` utility and add new configuration details.

15. If you try to close the trace configuration editor or the `ovtrcgui` window, the following message appears:



16. If you click **No**, the tracing mechanism continues to trace the configured applications on the system. If you click **Yes**, the `ovtrcgui` utility immediately disables the tracing mechanism.

Part IV: Logging Custom Data

The Operations Agent enables you to log custom data into the Metrics Datastore along with the default performance metric classes. You can either use submittal APIs or DSI to log custom data into the Metrics Datastore.

The process of logging custom data into the Metrics Datastore is simplified with the use of Perl based scriptable APIs. You can use APIs to log multi-instance data and 64-bit data types. For more information about APIs, see [Using Perl Application Programming Interface for Submitting Custom Data](#).

To submit custom data using DSI, you must create a class specification file that defines the model for logging data and then log the data into the Metrics Datastore. You cannot log 64-bit data types and multi-instance data using DSI. For more information about DSI see, [Overview of Data Source Integration](#).

Note:

- It is recommended that you use API to log custom data into the Metrics Datastore.
- Logging *future* data through APIs would result in undefined behavior when summarized data is queried.

Chapter 13: Using Perl Application Programming Interface for Submitting Custom Data

The process of custom data logging has been simplified with the use of Perl based scriptable APIs. You can use APIs to seamlessly log custom data into the Metrics Datastore. With APIs you can log 64-bit data types and multi-instance data into the datastore.

Note: You can use APIs to log data into the Metrics Datastore only if the **oacore** process is running.

Operations Agent 12.xx provides the following APIs to log custom data into the Metrics Datastore:

API Type	Summarization Type	Data Type	How to choose an API Type
AddCounter	Gives the last value of the summarized interval.	Integer	Use the Counter metrics to submit a cumulative count of activity, such as CPU times, physical IOs, paging, network packet counts and the like, to the datastore.
AddGauge	Gives the average of all values in the summarization interval.	Integer, Real	Use the Gauge metrics to submit an instant value at the time of the observation, such as the run queue, number of users, file system space utilization and the like, to the datastore.
AddAttribute	Gives the data that does not change frequently. No summarization is applicable. Note: History of Attribute data is not saved in the datastore. Whenever an Attribute is changed, it is replaced in the datastore.	Integer, Real, String	Use the Attribute metrics to submit static definitions or value such as the OS name, version, release, physical memory, CPU clock speed and the like, to the datastore.

AddString	Gives the last value of the summarization interval.	String	Use the String metrics to submit string values that change often.
-----------	---	--------	---

Note:

- Ensure that you do not submit a metric using different API types.

In the following instance:

```
AddCounter("Website:Global:UserLogins", "www.abcdefg.com", "Cumulative User Logins", 6000);
AddGauge("Website:Global:UserLogins", "www.abcdefg.com", "Cumulative User Logins", 5000);
```

When you submit the metric 'UserLogins' using both Counter and Gauge API type, you will get an incorrect result.

- After submitting a metric, you cannot change the type of the metric.

For example: After you submit a metric as Counter you cannot change it to Gauge or Attribute.

The following terms are used in the document:

- MetricObservation object contains data to be logged into the datastore along with the collection time stamp.
- MetricObservationList contains several MetricObservation objects.
- OAAccess is a client that connects to the datastore.
- SubmitObservations API is used to submit MetricObservationList to the Metrics Datastore.

SubmitObservations returns one of the following values:

- 0 - MetricObservationList submitted successfully to the Metrics Datastore
- -1 - Submission of MetricObservationList failed
- -2 - **oacore** process is not running

Submitting Custom Data

Syntax

```
<AddMethod>(<data source name>:<class name>:<metric name>, <Instance Identifier>,
<Label of the metric>,<value of metrics>)
```

API Arguments

The table below lists the arguments:

Argument	Description
Fully Qualified Metrics Name	Specifies the fully qualified metrics name. It includes the data source name, class name and metrics name. <data source name>:<class name>:<metric name>
Instance Identifier	Specifies the instance of the class. It can be an integer, real or a string value.
Label	Specifies label of the metric.
Value	Specifies the value of the metrics to be submitted. Note: For the AddGauge methods, Value can be either an integer or a real value. For the AddAttribute method, Value can be an integer, real or string value.

For Example:

```
AddGauge("MySQL:Global:MySQLMemUtil", "MyHost:3306", "MySQL Total CPU utilization percentage", 20.12);
```

Where:

AddMethod	AddGauge
Data source name	MySQL
Class name	Global
Metric name	MySQLMemUtil
Instance Identifier	MyHost:3306
Label of the metric	MySQL Total CPU utilization percentage
Value of metrics	20.12

How to Submit Custom Data to the Metrics Datastore

To submit data into the Metrics Datastore, create MetricObservation objects. Add the MetricObservation objects to the MetricObservationList.

Note: Any number of MetricObservation objects can be added to a MetricObservationList.

Submit the MetricObservationList to the Metrics Datastore using the SubmitObservations API.

Note: An OAAccess object must be created to invoke the SubmitObservations API.

The following table lists the steps to submit data to the Metrics Datastore along with an example:

No.	Steps to submit custom data to the Metrics Datastore	For example
1.	Create OAAccess.	<code>\$access = oaperlapi::OAAccess->new();</code>
2.	Create a MetricObservationList object.	<code>\$mol = oaperlapi::MetricObservationList->new();</code>
3.	Create a MetricObservation object with a time stamp. Note: Multiple MetricObservation objects can be created with different timestamps. If time is zero, then current time is taken. For every class of data that you submit, ensure that the timestamp of MetricObservations are in the increasing order.	<code>\$interval = time;</code> <code>\$mo = oaperlapi::MetricObservation->new(\$interval);</code>
4.	Add values to the MetricObservation object using methods such as AddGauge or AddAttribute.	<code>\$mo->AddAttribute("MySql:Global:MySQLVersion", "MyHost:3306", "MySQL version", "5.6.0");</code> <code>\$mo->AddGauge("MySql:Global:MySQLMemUtil", "MyHost:3306", "MySQL Total CPU utilization percentage", 20.12);</code>
5.	Add the MetricObservation objects to the MetricObservationList object.	<code>\$mol->AddObservation(\$mo);</code>

6.	Submit the MetricObservationList to the Metrics Datastore using SubmitObservations API.	\$access->SubmitObservations (\$mol);
----	---	---------------------------------------

For Example:

```
$access = oaperlapi::OAAccess->new();
$mol = oaperlapi::MetricObservationList->new();
$interval = time;
$mo = oaperlapi::MetricObservation->new($interval);
$mo->AddAttribute("MySql:Global:MySQLVersion", "MyHost:3306", "MySQL version", "5.6.0");
$mo->AddGauge("MySql:Global:MySQLMemUtil", "MyHost:3306", "MySQL Total CPU utilization
percentage", 20.12);
$mol->AddObservation($mo);
$access->SubmitObservations($mol);
```

Use Case to Submit Data to Datastore Using APIs

Scenario

John is a web administrator of the website www.abcdefg.com. He wants analyze the number of users who visit his website from the USA. John also wants to monitor the number of purchases made between 10 A.M and 11 A.M.

Description

John wants to know the following values:

- Number of users who logged-in during an hour from the USA.
- Number of purchases made in an hour.

Actor

John - Web administrator

Precondition

The time duration to monitor user log-ins and purchases is one hour.

Time Interval	Cumulative User Log-ins	Location	Purchases in an hour
---------------	-------------------------	----------	----------------------

10:00 – 10:30	6000	USA	400
10:30 – 11:00	8000	USA	600

Note:

Ensure that the path is prefixed with <InstallDir>/nonOV/perl/a/bin

Use the following commands to run the following script

On Windows X64:

```
perl -I <InstallDir>\support -I <InstallDir>\nonOV\perl\a\lib\site_
perl\5.16.0\MSWin32-AMD64-multi-thread <perl script>
```

On Windows X86:

```
perl -I <InstallDir>\support -I <InstallDir>\nonOV\perl\a\lib\site_
perl\5.16.0\MSWin32-x86-multi-thread <perl script>
```

On HP-UX PA-RISC:

LD_PRELOAD needs to set to the path of liboaperlapi.sl while running the perl script

Example:

```
LD_PRELOAD=/opt/OV/nonOV/perl/a/lib/site_perl/5.16.0/PA-RISC2.0-thread-
multi/liboaperlapi.sl /opt/OV/nonOV/perl/a/bin/perl <perl script>
```

On UNIX:

```
perl <perl script>
```

To submit the data to the datastore run the following Perl Script:

```
use oaperlapi;
```

```
$now_string = localtime;
```

Step 1: Create the access object

```
$access = oaperlapi::OAAccess->new();
```

Step 2: Create a MetricObservationList object

```
$molist = oaperlapi::MetricObservationList->new();
```

```
$obsTimestamp = 1415939400;
```

Note: In this instance, \$obsTimestamp represent the time when the first record was logged into the datastore.

Step 3: Create a MetricObservation object with a time stamp

```
$mdbl = oaperlapi::MetricObservation->new($obsTimestamp);
```

Step 4: Add values to the MetricObservation object

```
$mdbl->AddAttribute("Website:Global:Location", "www.abcdefg.com", "Location",
"USA");
$mdbl->AddCounter("Website:Global:UserLogins", "www.abcdefg.com", "Cumulative User
Logins", 6000);
$mdbl->AddGauge("Website:Global:Purchases", "www.abcdefg.com", "User Purchases in
an hour", 400);
$mdbl->AddString("Website:Global:Name", "www.abcdefg.com", "Name of User",
"TestUser");
```

Step 5: Add MetricObservation objects to the MetricobservationList

```
$molist->AddObservation($mdbl);
# Go to the next Timestamp
$obsTimestamp += 1800;
```

Repeat steps 3,4, and 5 to create another observation as shown:

```
$mdbl = oaperlapi::MetricObservation->new($obsTimestamp);
$mdbl->AddCounter("Website:Global:UserLogins", "www.abcdefg.com", "Cumulative User
Logins", 8000);
$mdbl->AddGauge("Website:Global:Purchases", "www.abcdefg.com", "User Purchases in
an hour", 600);
$molist->AddObservation($mdbl);
```

Step 6: Submit data in bulk

```
$access->SubmitObservations($molist);
```

The following data is logged into the datastore:

```
===
11/14/14 10:00:00|GLOBAL_ID          |www.abcdefg.com|
11/14/14 10:00:00|LOCATION            |USA             |
11/14/14 10:00:00|USERLOGINS       |                |6000           |
11/14/14 10:00:00|PURCHASES       |                |400.00        |
11/14/14 10:00:00|NAME             |TestUser       |
===
11/14/14 10:30:00|GLOBAL_ID          |www.abcdefg.com|
11/14/14 10:30:00|LOCATION            |USA             |
11/14/14 10:30:00|USERLOGINS       |                |8000           |
11/14/14 10:30:00|PURCHASES       |                |600.00        |
11/14/14 10:00:00|NAME
```

Special Characters Usage in Domain, Class and Metric Names

The domain, class and metric names does not support the following characters:

\, /, *, :, ;, ?, ", <, >, |, [,]

Note: The following characters are replaced by an underscore _:

\, /, *, :, ;, ?, ", <, >, |

The brackets [,] are replaced by parentheses (,).

The characters that the domain, class and metric names support are listed in the following table:

Character	Name
A-Z, a-z	uppercase and lowercase letters
0-9	numbers
-	hyphen
_	underscore
.	period
	space
,	comma
()	parentheses
{}	braces
\$	dollar sign
#	number sign
=	equal sign

Note: All the other special characters not mentioned are replaced by an underscore _.

Chapter 14: Overview of Data Source Integration

The Data Source Integration (DSI) feature helps you to log custom data, define alarms, and access metrics from new sources of data beyond the metrics logged by the Performance Collection Component's **oacore** data collector. Metrics can be acquired from datasources such as databases, LAN monitors, and end-user applications.

You can use the Performance Manager to view data logged by DSI, along with the standard performance metrics logged by the Operations Agent's data collector. You can also use the extract program to export the data logged using DSI, for display in spreadsheets or similar analysis packages.

Upgrading to Operations Agent 12.xx

With the Operations Agent 12.05, data collected by DSI is stored in the Metrics Datastore. For every class of DSI data that is logged into the Metrics Datastore, a database file is created. However, to preserve the backward compatibility, the command line continues to support the logfile argument.

```
sdlcomp <Class specification file> <log file name>
```

The DSI compiler **sdlcomp** will not create files based on the log file name; instead the log file name is used as a datasource name.

For example:

If `"/tmp/test_log"` or `"C:\test_log"` is used as a log file argument (in the command line), `test_log` is used as a datasource name.

Before upgrading from Operations Agent 11.xx to 12.xx, ensure that the DSI datasource is added to the `datasources` file located at:

On HP-UX/Linux/Solaris:

```
/var/opt/OV/conf/perf
```

On Windows:

```
%ovdatadir%\conf\perf
```

Add the following entry to the `datasources` file:

```
DATASOURCE=<Datasource Name>LOGFILE=<DSI Logfile Path>
```


The model to log DSI data is created in the Metrics Datastore only if the DSI datasource is added to the `datasources` file.

Note: Any option that is not supported with the Operations Agent 12.xx is ignored and a message as shown in the following example appears:

For example:

```
MAX INDEXES is obsoleted and will be ignored
INDEX BY is obsoleted and will be ignored
CAPACITY is obsoleted and will be ignored
ROLL BY is obsoleted and will be ignored
ACTION is obsoleted and will be ignored
PRECISION is obsoleted and will be ignored
```

Removing custom data sources logged to CODA Database in Operations Agent 11.xx

After upgrading to Operations Agent 12.xx from 11.xx, the custom data and model in the older files are retained in the read-only mode. These old data sources are not removed if you run the `sdlutil` (or `ddfutil`) command.

You can remove the old data sources either before or after upgrading to Operations Agent 12.xx. After upgrading to Operations Agent 12.xx, follow the steps to remove all the custom data sources logged to CODA database:

1. Stop the **oacore** process
2. Remove complete CODA logfile set
3. Start the **oacore** process
4. Run the `sdlutil` command to complete the cleanup.

All CODA log files located at `%0vDataDir%\datafiles\coda*` are removed.

How DSI Works

Follow these steps to log the DSI data into the Metrics Datastore:

1. Create a model to log DSI data into the Metrics Datastore. For more information see, [Creating a Model to Log DSI Metrics to the Metrics Datastore](#).
 - a. Create a Class Specification file. For more information see, "[Creating a Class Specification File](#)" on page 267

- b. Run the following command to compile the class specification file using the DSI compiler, **sdldcomp**:

```
sdldcomp <class specification file> <log file name>
```

For more information see, "[Compiling the Class Specification File Using the DSI Compiler](#)" on [page 276](#)

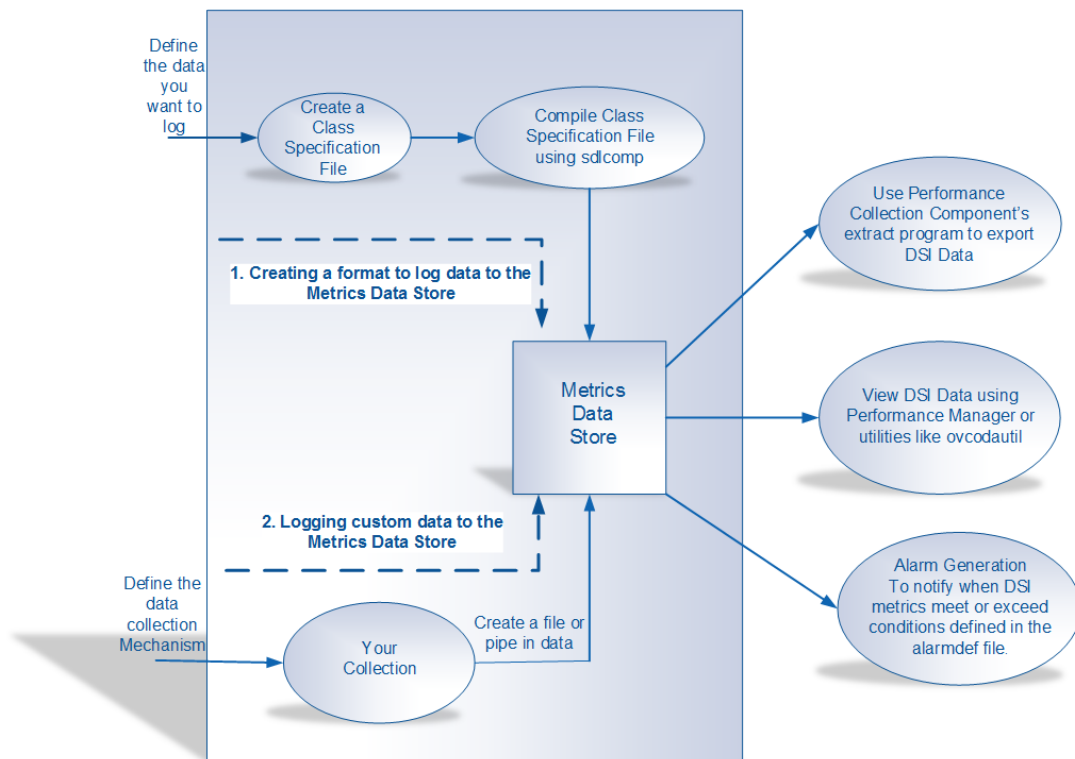
The model to log data (meta data) is saved in the Metrics Datastore.

- Log DSI data into the Metrics Datastore. For more information see, "[Logging DSI Metrics into the Metrics Datastore](#)" on [page 278](#)

Pipe custom data into dsilog. Dsilog logs data into the Metrics Datastore

The following diagram shows how data collected by DSI is logged into the Metrics Datastore.

Figure 17 Data Source Integration Process



You can use the Performance Collection Component's **extract** program to export DSI data. You can view the DSI data with the Performance Manager or utilities like `ovcodauti1`. You can configure alarms to notify when DSI metrics exceed conditions defined in the `alarmdef` file.

Chapter 15: Creating a Model to Log DSI Metrics into the Metrics Datastore

Before logging the DSI metrics into the Metrics Datastore, you must create and log the model for logging DSI metrics. Follow the steps to log the model into the Metrics Datastore:

1. [Create a class specification file](#)
2. [Log the model into the Metrics Datastore](#)

Creating a Class Specification File

For each source of incoming data, you must create a class specification file to describe the model for storing the incoming data. To create the file, use the ["Class Specification Syntax" on the next page](#).

The class specification file contains:

- Class description that assigns a name and numeric ID to the incoming data set. For more information about class description see, ["Class Description" on page 269](#).
- Metric description that defines the metrics to be logged. A metric description defines names and describes a data item. For more information about metrics description see, [Metrics Descriptions](#).

To generate the class specification file, use any text editor or word processor that lets you save the file as an ASCII text file. Enter the following information in the class specification:

- Data class name and ID number
- Label name (optional) that is a substitute for the class name. (For example, if a label name is present, it can be used in Performance Manager.)
- Metric names

Here is an example of a class specification:

```
CLASS VMSTAT_STATS = 10001
LABEL "VMSTAT data"
RECORDS PER HOUR 120
;

METRICS
```

```
RUN_Q_PROCS    = 106
LABEL "Procs in run q"
;

BLOCKED_PROCS  = 107
LABEL "Blocked Processes"
;
```

You can include multiple classes in a class specification file.

When you run the DSI compiler, **sdlcomp**, the model to log DSI data is logged in the Metrics Datastore.

Class Specification Syntax

Use the following conventions to create a class specification file:

- Syntax statements shown in brackets [] are optional.
- Commas can be used to separate syntax statements for clarity anywhere except directly preceding the semicolon, which marks the end of the class specification and the end of each metric specification.
- Comments start with # or //. Everything following a # or // on a line is ignored.
- Add a semicolon after every class description and metric description.
- Statements are not case-sensitive.

Note: User-defined descriptions, such as *metric_label_name* or *class_label_name*, cannot be the same as any of the keyword elements of the DSI class specification syntax.

For example:

```
CLASS class_name = class_id_number
[LABEL "class_label_name"]
[RECORDS PER HOUR number]
;

METRICS
metric_name = metric_id_number
[LABEL "metric_label_name"]
;
```

Class Description

To create a class description, assign a name to a group of metrics from a specific datasource.

You must begin the class description with the CLASS keyword. The final parameter in the class specification must be followed by a semicolon.

Syntax

```
CLASS class_name = class_id_number  
[LABEL "class_label_name"]  
[RECORDS PER HOUR number]  
;
```

CLASS

The class name and class ID identify a group of metrics from a specific datasource.

Syntax

```
CLASS class_name = class_id_number
```

How to Use It

The *class_name* and *class_ID_number* must meet the following requirements:

- *class_name* is alphanumeric and can have a maximum of 20 characters. *class_name* must start with an alphabetic character and can contain underscores (but no special characters). It is not case-sensitive.
- *class_ID_number* must be numeric and can have a maximum of 6 digits.
- The *class_name* and *class_ID_number* must each be unique for all the classes you define and cannot be the same as any application defined in the Performance Collection Component's **parm** file. For information about the **parm** file, see [Using the parm file](#).

For example

```
CLASS VMSTAT_STATS = 10001;
```

LABEL

The class label identifies the class as a whole. It is used instead of the class name in the Performance Manager.

Syntax

```
[ LABEL "class_label_name"]
```

How To Use It

The *class_label_name* must meet the following requirements:

- It must be enclosed in double quotation marks.
- It can be up to 48 characters long.
- It cannot be the same as any of the keyword elements of the DSI class specification syntax.
- If it contains a double quotation mark, precede it with a backslash (\). For example, you would enter "\"my\" data" if the label is "my" data.
- If no label is specified, the *class_name* is used as the default.

Example

```
CLASS VMSTAT_STATS = 10001  
LABEL "VMSTAT data";
```

Records per Hour

The RECORDS PER HOUR setting determines how many records are written to the database file every hour. The default number for RECORDS PER HOUR is 12 to match Performance Collection Component's measurement interval of data sampling once every five minutes (60 minutes/12 records = logging every five minutes).

The default number or the number you enter may require the logging process to summarize data before it becomes part of the database file. The method used for summarizing each data item is specified in the metric description. For more information, see [Summarization Method](#).

Syntax

```
[RECORDS PER HOUR number]
```

How To Use Records per Hour

The logging process uses this value to summarize incoming data to produce the number of records specified. For example, if data arrives every minute and you have set RECORDS PER HOUR to 6 (every 10 minutes), 10 data points are summarized to write each record to the class. Some common RECORDS PER HOUR settings are shown below:

```
RECORDS PER HOUR 6      --> 1 record/10 minutes
RECORDS PER HOUR 12     --> 1 record/5 minutes
RECORDS PER HOUR 60     --> 1 record/minute
RECORDS PER HOUR 120    --> 1 record/30 seconds
```

If `dsilog` receives no metric data for an entire logging interval, a missing data indicator is logged for that metric.

For example

```
CLASS VMSTAT_STATS = 10001
LABEL "VMSTAT data"
RECORDS PER HOUR 6;
```

In this example, a record will be written every 10 minutes.

Default Settings

The default settings for the class description are:

```
LABEL (class_name)
RECORDS PER HOUR 12
```

To use the defaults, enter only the CLASS keyword with a *class_name* and numeric *class_id_number*.

Sample Class Specification

```
CLASS VMSTAT_STATS = 10001
LABEL "VMSTAT data"
RECORDS PER HOUR 120;
METRICS
RUN_Q_PROCS = 106
LABEL "Procs in run q"
;
```

```
BLOCKED_PROCS = 107
LABEL "Blocked Processes"
;
SWAPPED_PROCS = 108
LABEL "Swapped Processes"
;
AVG_VIRT_PAGES = 201
LABEL "Avg Virt Mem Pages"
;
FREE_LIST_SIZE = 202
LABEL "Mem Free List Size"
;
PAGE_RECLAIMS = 303
LABEL "Page Reclaims"
;
ADDR_TRANS_FAULTS = 304
LABEL "Addr Trans Faults"
;
PAGES_PAGED_IN = 305
LABEL "Pages Paged In"
;
PAGES_PAGED_OUT = 306
LABEL "Pages Paged Out"
;
PAGES_FREED = 307
LABEL "Pages Freed/Sec"
;
MEM_SHORTFALL = 308
LABEL "Exp Mem Shortfall"
;
CLOCKED_PAGES = 309
```



```
LABEL "Pages Scanned/Sec"  
;  
DEVICE_INTERRUPTS = 401  
LABEL "Device Interrupts"  
;  
SYSTEM_CALLS = 402  
LABEL "System Calls"  
;  
CONTEXT_SWITCHES = 403  
LABEL "Context Switches/Sec"  
;  
USER_CPU = 501  
LABEL "User CPU"  
;  
SYSTEM_CPU = 502  
LABEL "System CPU"  
;  
IDLE_CPU = 503  
LABEL "Idle CPU"  
;
```

Metrics Descriptions

The metric name and ID number identify the metric being collected.

Syntax

METRICS

metric_name = *metric_id_number*

How To Use It

The metrics section must start with the METRICS keyword before the first metric definition. Each metric must have a metric name that meets the following requirements:

- Metrics name can have a maximum of 20 characters.
- It must begin with an alphabet.
- Metrics name can have only alphanumeric characters and underscores.
- It is not case-sensitive.

The metric ID number can have a maximum of 6 characters.

The *metric_name* and *metric_id_number* must each be unique among all the metrics you define in the class. The combination *class_name:metric_name* must be unique for this system, and it cannot be the same as any *application_name:metric_name*.

Each metric description is separated from the next by a semicolon (;). The order of the metric names in this section of the class specification determines the order of the fields when you export the logged data.

A timestamp metric is automatically inserted as the first metric in each class. If you want the timestamp to appear in a different position in exported data, include the short form of the internally defined metric definition (DATE_TIME;) in the position you want it to appear. To omit the timestamp and to use a UNIX timestamp (seconds since 1/1/70 00:00:00) that is part of the incoming data, choose the `-timestamp` option when starting the `dsilog` process.

Note: You must compile each class using `sd1comp` and then start logging the data for that class using the `dsilog` process, regardless of whether you have reused metric names.

LABEL

The metric label identifies the metric in Performance Manager graphs and exported data.

Syntax

```
[LABEL "metric_label_name"]
```

How To Use It

Specify a text string, surrounded by double quotation marks, to label the metric in graphs and exported data. Up to 48 characters are allowed. If no label is specified, the metric name is used to identify the metric.

Note: If the label contains a double quotation mark, precede it with a backslash (\). For example, you would enter `"\"my\" data"` if the label is "my" data.

For example

```
METRICS
RUN_Q_PROCS = 106
LABEL "Procs in run q";
```

Summarization Method

The summarization method determines how to summarize data if the number of records exceeds the number set in the `RECORDS PER HOUR` option of the `CLASS` section. The summarization method is only valid for numeric metrics.

Syntax

```
[{TOTALLED | AVERAGED | SUMMARIZED BY metric_name}]
```

How To Use It

`SUMMARIZED BY` should be used when a metric is not being averaged over time, but over another metric in the class. For example, assume you have defined metrics `TOTAL_ORDERS` and `LINES_PER_ORDER`. If these metrics are given to the logging process every five minutes but records are being written only once each hour, to correctly summarize `LINES_PER_ORDER` to be (total lines / total orders), the logging process must perform the following calculation every five minutes:

- Multiply `LINES_PER_ORDER * TOTAL_ORDERS` at the end of each five-minute interval and maintain the result in an internal running count of total lines.
- Maintain the running count of `TOTAL_ORDERS`.
- At the end of the hour, divide total lines by `TOTAL_ORDERS`.

To specify this kind of calculation, you would specify `LINES_PER_ORDER` as `SUMMARIZED BY TOTAL_ORDERS`.

If no summarization method is specified, the metric defaults to `AVERAGED`.

For example:

```
METRICS
ITEM_1_3 = 11203
LABEL "TOTAL_ORDERS"
TOTALLED;
ITEM_1_5 = 11205
LABEL "LINES_PER_ORDER"
SUMMARIZED BY ITEM_1_3;
```

Compiling the Class Specification File Using the DSI Compiler

Run the DSI compiler, **sdlcomp** with the class specification file.

Compiler Syntax

```
sdlcomp <class specification file> <log file name>
```

In this instance:

<class specification file> is the name of the file that contains the class specifications. If it is not in the current directory, it must be fully qualified.

<log file name> is used as a datasource name.

The DSI compiler **sdlcomp** will not create files based on log file name. Instead the log file name is used as a datasource name.

For example:

If `"/tmp/test_log"` or `"C:\test_log"` is given as the logfile set argument (in the command line), `test_log` will be used as a datasource name.

The model for logging DSI data is logged in the Metrics Datastore

sdlcomp Compiler

The **sdlcomp** compiler checks the class specification file for errors. If no errors are found, it adds the datasource, class and metric descriptions to the Metrics Datastore.

Changing a Class Specification

To change a class specification file, you must recreate the whole datasource as follows:

1. Stop the `dsilog` process.
2. Export the data from the Metrics Datastore for all the classes.

3. Run `sdlutil` to remove the datasource. See [Managing Data with sdlutil](#) for information.
4. Update the class specification file.
5. Run `sdlcomp` to recompile the class specification.
6. Run `dsilog` to start logging based on the new class specification.

Chapter 16: Logging DSI Metrics into the Metrics Datastore

To log the DSI metrics into the Metrics Datastore, ensure that the model for logging DSI data is logged into the datastore. Modify the alarm definitions file, **alarmdef**, to notify when DSI metrics exceed defined conditions. For more information about defining alarms for DSI metrics, see [Defining Alarms for DSI Metrics](#). Start the collection process from the command line. Pipe the data from the collection process to **dsilog** (or use some other method to get it to **stdin**) with the appropriate variables and option set.

Syntax

```
dsilog <sdl-file-name> <data-class-name> [options]
```

In this instance:

<sdl-file-name> is the name of the root self-describing file.

<data-class-name> is the name of the data class.

Table 1: dsilog parameters and options

Variables and Options	Definitions
-c <char>	Uses the specified character as a string delimiter or separator.
-i <fifo>	Indicates that the input should come from the <code>fifo</code> named. The <code>fifo</code> is the alternative for the input stream.
-timestamp	Timestamp needs to be specified as the first column in the input data file. This needs to be in EPOCH format.
-?	Displays the syntax description.
-vers	Displays print version information.
-s seconds	Indicates the number of seconds by which to summarize the data. A zero (-s 0) turns off summarization and all incoming data is logged. Any value other than zero (0) is ignored and the summarization rate for non-zero value defaults to RECORDS PER HOUR.

Note: The **dsilog** program is designed to receive a continuous stream of data. Therefore, it is important to structure scripts so that **dsilog** receives continuous input data. Do not write scripts that create a new **dsilog** process for new input data points. This can cause duplicate timestamps to be written to the **dsilog** file, and can cause problems for Performance Manager and **per-fa1arm** when reading the file.

See [Examples of Data Source Integration](#), for examples of problematic and recommended scripts

dsilog Logging Process

The **dsilog** process requires that you either devise your own program or use an existing process. You can then pipe this data into **dsilog**, which logs the data into the datastore. A separate logging process must be used for each class you define.

dsilog expects to receive data from **stdin**. To start the logging process, you could pipe the output of the process you are using to collect data to **dsilog** as shown in the following example:

```
vmstat 60 | dsilog <logfile name> <class name>
```

You can only have one pipe (|) in the command line. This is because with two pipes, UNIX buffering will hold up the output from the first command until 8000 characters have been written before continuing to the second command and piping out to the log file.

You could also use a **fifo** (named pipe).

For example:

```
mkfifo -m 777 myfifo  
dsilog logfile_set class -i myfifo &  
vmstat 60 > myfifo &
```

The **&** causes the process to run in the background.

How dsilog Processes Data

The **dsilog** program scans each input data string, parsing delimited fields into individual numeric or text metrics. A key rule for predicting how the data will be processed is the validity of the input string. A valid input string requires that a delimiter be present between any specified metric types (numeric or text). A blank is the default delimiter.

You *must* include a new line character at the end of any record fed to DSI in order for DSI to interpret it properly.

Managing Data with sdlutil

Use the `sdlutil` program to manage the data in the DSI database files. You can do the following:

- list the defined class and metric information to `stdout`. You can redirect output to a file.
- remove datasource, classes, metrics and data from the Metrics Datastore
- display version information.

Syntax

`sdlutil <log file name>[option]`

Variables and Options	Definitions
logfile name	is used as a datasource name.
-rm all	removes datasource, classes, metrics and data from the Metrics Datastore
-vers	displays version information.
-?	displays the syntax description.

For example:

Run the following command to remove the datasource, `test_log` along with the classes, metrics and data from the Metrics Datastore.

```
sdlutil /tmp/test_log -rm all
```


Chapter 17: Using the DSI Data Logged into the Metrics Datastore

After DSI data is logged into the Metrics Datastore, you can export the data using the Performance Collection Component's **extract** program. You can also configure alarms to occur when DSI metrics exceed the defined conditions.

Here are ways to use the logged DSI data:

- Export the data for use in reporting tools such as spreadsheets.
- Display exported DSI data using analysis tools such as in Performance Manager.
- Monitor alarms using Operations Manager.

Defining Alarms for DSI Metrics

Use the `alarmdef` file on the Performance Collection Component system to define alarms on DSI metrics. These alarms notify you when the DSI metrics meet or exceed conditions that you have defined. The `alarmdef` file is located in the `var/opt/perf/` configuration directory of Performance Collection Component.

Whenever you specify a DSI metric name in an alarm definition, it should be fully qualified; that is, preceded by the `datasource_name`, and the `class_name` as shown:

```
<datasource_name>:<class_name>:<metric_name>
```

- `datasource_name` is the name you have used to configure the data source in the `datasources` file.
- `class_name` is the name you have used to identify the class in the class specification for the data source. You do not need to enter the `class_name` if the metric name is unique (not reused) in the class specification.
- `metric_name` is the data item from the class specification for the data source.

However, if you choose not to fully qualify a metric name, you need to include the `USE` statement in the `alarmdef` file to identify which `datasource` to use. For more information see the [USE statement](#) in the *Chapter Performance Alarms*.

To activate the changes made to the `alarmdef` file, run the `ovpa restart alarm` command in the command line.

For detailed information on the alarm definition syntax, how alarms are processed, and customizing alarm definitions, see [Performance Alarms](#).

Alarm Processing

As data is logged by `dsilog` it is compared to the alarm definitions in the `alarmdef` file to determine if a condition is met or exceeded. When this occurs, an alert notification or action is triggered.

You can configure where you want alarm notifications to be sent and whether you want local actions performed. Alarm notifications can be sent to the central Performance Manager analysis system where you can draw graphs of metrics that characterize your system performance. Local actions can be performed on the Performance Collection Component system. Alarm information can also be sent to Operations Manager.

Exporting DSI Data

To export the DSI data from the Metrics Datastore, use the `extract` program's `export` function. For more information, see [Using the Extract Program](#). Use the `ovcodauti1 -obj` command to view the list of DSI datasources and classes that can be exported.

Example of Using Extract to Export DSI Log File Data

```
extract -xp -l logfile_set -C class [options]
```

You can use `extract` command line options to do the following:

- Specify an export output file.
- Set begin and end dates and times for the first and last intervals to export.
- Specify a separation character to put between metrics on reports.
- Choose whether or not to display headings and blank records for intervals when no data arrives and what the value displayed should be for missing or null data.

- Display exported date/time in UNIX format or date and time format.
- Set additional summarization levels.

Viewing Data in Performance Manager

You can centrally view, monitor, analyze, compare, and forecast trends in DSI data using the Performance Manager. Performance Manager helps you identify current and potential problems. It provides the information you need to resolve problems before productivity is affected.

Chapter 18: Examples of Data Source Integration

Data source integration is a powerful and flexible technology. This chapter contains examples of using DSI to collect and log data into the Metrics Datastore.

Writing a dsilog Script

The **dsilog** code is designed to receive a continuous stream of data rows as input. This stream of input is summarized by **dsilog** according to the specification directives for each class, and one summarized data row is logged per requested summarization interval. Performance Manager and `perfa1arm` work best when the timestamps written in the log conform to the expected summarization rate (records per hour). This happens automatically when **dsilog** is allowed to do the summarization.

dsilog process for each arriving input row, which may cause problems with Performance Manager and `perfa1arm`. This method is not recommended.

- Problematic **dsilog** script
- Recommended **dsilog** script

Example 1 - Problematic dsilog Script

In the following script, a new **dsilog** process is executed for each arriving input row.

```
while :
do
    feed_one_data_row | dsilog sdlname classname
    sleep 50
done
```

Example 2 - Recommended dsilog Script

In the following script, one **dsilog** process receives a continuous stream of input data. `feed_one_data_row` is written as a function, which provides a continuous data stream to a single **dsilog** process.

```
# Begin data feed function

feed_one_data_row()

{

while :

do

# Perform whatever operations necessary to produce one row
# of data for feed to a dsilog process

sleep 50

done

}

# End data feed function

# Script mainline code

feed_one_data_row | dsilog sdlname classname
```

Logging vmstat Data

This example shows you how to set up data source integration using default settings to log the first two values reported by `vmstat`.

The procedures needed to implement data source integration are:

- Creating a class specification file.
- Compiling the class specification file.
- Starting the **dsilog** logging process.

Creating a Class Specification File

The class specification file is a text file that you create to describe the class, or a set of incoming data, as well as each individual number you intend to log as a metric within the class. The file can be created with the text editor of your choice. The file for this example of data source integration should be created in the **/tmp/** directory.

The following example shows the class specification file is used to describe the first two vmstat numbers for logging in a class called VMSTAT_STATS. Because only two metrics are defined in this class, the logging process ignores the remainder of each vmstat output record. Each line in the file is explained in the comment lines that follow it.

```
CLASS VMSTAT_STATS = 10001;
# Assigns a unique name and number to vmstat class data

# The semicolon is required to terminate the class section
# of the file.

METRICS
# Indicates that everything that follows is a description
# of a number (metric) to be logged.

RUN_Q_PROCS = 106;
# Assigns a unique name and number to a single metric.
# The semicolon is required to terminate each metric.

BLOCKED_PROCS = 107;
# Assigns a unique name and number to another metric.
# The semicolon is required to terminate each metric.
```

Compiling the Class Specification File

When you compile the class specification file using `sdlcomp`, the file is checked for syntax errors. If none are found, `sdlcomp` creates or updates a set of log files to hold the data for the class.

Use the file name that you gave to the class specification file and then specify a logfile path. The logfile name in the logfile path is used as the datasource name.

Note: No log files are created.

In the command and compiler output example below, **/tmp/vmstat.spec** is the class specification file name and **/tmp/VMSTAT_DATA** is the logfile path.

```
-> sdlcomp /tmp/vmstat.spec /tmp/VMSTAT_DATA
```

This example creates a datasource called VMSTAT_DATA. If there are syntax errors in the class specification file, messages indicating the problems are displayed and the datasource is not created.

To verify if the class VMSTAT_STATS was successfully added to datasource, run the following command:

```
ovcodutil -ds VMSTAT_DATA -obj
```

Starting the dsilog Logging Process

Now you can pipe the output of `vmstat` directly to the **dsilog** logging process. Use the following command:

```
vmstat 60 | dsilog /tmp/VMSTAT_DATA VMSTAT_STATS &
```

This command runs `vmstat` every 60 seconds and sends the output directly to the VMSTAT_STATS class in the VMSTAT_DATA datasource. The command runs in the background. You could also use `remsh` to feed `vmstat` in from a remote system.

Note that the following message is generated at the start of the logging process:

```
Metric has invalid data. Ignore to end of line, metric value exceeds maximum.
```

This message is a result of the header line in the `vmstat` output that **dsilog** cannot log. Although the message appears on the screen, **dsilog** continues to run and begins logging data with the first valid input line.

Accessing the Data

You can use **extract** program command line arguments to export data from the class. For example:

```
extract -xp -l /tmp/VMSTAT_DATA -C VMSTAT_STATS
```

Note that to export DSI data, you must be root or the creator of the log file.

Logging the Number of System Users

The next example uses `who` to monitor the number of system users. Again, we start with a class specification file.

```
# who_wc.spec

#
# who word count DSI spec file
#
CLASS who_metrics = 150
LABEL "who wc data"
;
METRICS
who_wc = 151
label "who wc"
averaged
;
sdlcomp ./who_wc.spec ./who_wc_log.
```

Unlike `sar`, you cannot specify an interval or iteration value with `who`, so we create a script that provides, at a minimum, interval control.

```
#!/bin/ksh who_data_feed

while :
do
    # sleep for one minute (this should correspond with the
sleep 60

    # Pipe the output of who into wc to count
    # the number of users on the system.
who | wc -l > /usr/tmp/who_data

    # copy the data record to the pipe being read by dsilog.
```



```
cat /usr/tmp/who_data > ./who.fifo
```

```
done
```

Again we need a fifo and a script to supply the data to **dsilog**, so we return to the step by step process.

1. Create two fifos; one will be the dummy fifo used to “hold open” the real input fifo

```
.# Dummy fifo.  
mkfifo ./hold_open.fifo  
# Real input fifo for dsilog.  
mkfifo ./who.fifo
```

2. Start **dsilog** using the `-i` option to specify the input coming from a fifo. It is important to start **dsilog** before starting the `who` data feed.

```
dsilog ./who_wc_log who_metrics \-i ./who.fifo &
```

3. Start the dummy process to hold open the input fifo.

```
cat ./hold_open.fifo \> ./who.fifo &
```

4. Start the `who` data feed script (`who_data_feed`).

```
./who_data_feed &
```

Chapter 19: Using Metric Streaming

Operations Agent enables you to log custom metrics into the Metrics Datastore along with the default system performance metric classes.

With the Operations Agent 12.01, custom and system performance metrics are also available for streaming by using the Metric Streaming functionality. You can configure metric streaming by using the **Metric Streaming Configuration** policy.

Metrics Streaming enables you to stream metrics data to a target subscriber (For example: Performance Engine) and use the data for graphing and analysis. You can [submit data](#) to **hpsensor** and deploy the **Metric Streaming Configuration** policy from the Operations Manager i (OMi) to the Operations Agent to stream metrics data. For more information about the **Metric Streaming Configuration** policy, see the *OMi Administration Guide*.

Note: Compute Sensor (hpsensor) is a light-weight **performance** and **log data** collection process.

Streaming Metrics

To stream metrics data, REST based interface is provided to [submit data](#) to **hpsensor**. **hpsensor** will publish the submitted data to the target subscriber at a [configured interval](#).

Before submitting data to **hpsensor**, you must post [registration](#) information to **hpsensor** using the REST API provided. Also, you have to create a policy in the Operations Manager i (OMi) under the **Metric Streaming Configuration** policy type to choose the list of metrics to be streamed, provide the target URL and then deploy that policy to the Operations Agent from OMi.

Note: Metrics are not streamed if the target receiver is not available.

The following REST APIs are available for registration and data submission from local system:

1. [Registration](#) : /v4/RTCM/Register
2. [Data Submission](#): /v4/RTCM/SubmitData

Usage:

The following URI can be used to POST data to **hpsensor** using a http client:

- For registration: `http://<localhost>:<port>/hpcs/v4/RTCM/Register`
- For data submission: `http://<localhost>:<port>/hpcs/v4/RTCM/SubmitData`

In this instance,

`<localhost>` is a host name resolving to the loop-back address. The loop-back addresses are: 127.0.0.1 for IPv4 and `[::1]` for IPv6.

`<port>` is the **hpsensor** port. Run the following command to get the **hpsensor** port:

```
<OvInstallBinDir>ovconfget hpsensor PORT
```

hpsensor accepts data in **v4 JSON** (JavaScript Object Notation) format. The significance of v4 JSON format is, metric centric data registration and data submission. The data should contain the following objects:

1. Title
2. Version
3. Metric Payload consisting of the MetaData and Data Fields

Following is the schema for v4 JSON format:

```
[{
  "Title": "Real Time Metrics",
  "Version": "v4",
  "Metric Payload": [
    {
      "MetaData": {
        "MetricName": {
          "description": "The unique name of the metric",
          "type": "string"
        },
        "CategoryType": {
          "description": "The category type of the metric
(attribute/counter/gauge/...)",
```

```
        "type": "string"
    },
    "DataType": {
        "description": "The data type of the metric",
        "type": "string"
    },
    "ClassName": {
        "description": "The class name of the metric",
        "type": "string"
    },
    "Unit": {
        "description": "The unit of the metric value",
        "type": "string"
    },
    "Label": {
        "description": "The label of the metric",
        "type": "string"
    },
    "DataSource": {
        "description": "The data source the metric belongs to",
        "type": "string"
    },
    "Key": {
        "description": "Key number of the metric, key value 0... n
for Key metric",
        "type": "integer"
    },
    },
```

```
"Interface": {
    "description": "The submittal interface of collection",
    "type": "string"
},
"CollectorName": {
    "description": "The name of the metric collector",
    "type": "string"
},
"CollectorExec": {
    "description": "The collector executable to be executed",
    "type": "string"
},
"OriginalMetricName": {
    "description": "The domain the metric belongs to",
    "type": "string"
},
"required": [
    "MetricName",
    "ClassName",
    "DataSource",
    "Key"
],
>Data": {
    "Instances": [
        {
            "value": {
```

```
        "description": "The value of the metric  
for an instance",  
  
        "type": "string/integer/real"  
    },  
    "timestamp": {  
        "description": "The epoch time when the  
metric was collected",  
  
        "type": "integer"  
    },  
    "dimensions": {  
        "keyinstance": {  
            "description": "The key to  
uniquely identify an instance",  
  
            "type": "string/integer/real"  
        },  
        "minItems": 1,  
        "uniqueItems": true  
    },  
    "minItems": 1,  
    "uniqueItems": true  
  }  
]  
}]
```

The data consists of the following parameters:

Parameter	Mandatory		Description
	For Registration	For Data Submission	
Title	Yes	Yes	Must be "Real Time Metrics" .
Version	Yes	Yes	Must be "v4" as the REST API provided accepts data in v4 JSON format.
MetricName	Yes	Yes	The unique name for the metric.
CategoryType	<i>Optional</i>	<i>Optional</i>	Category type of the metric. The accepted category types are: Key, Attribute, Gauge and Counter.
DataType	<i>Optional</i>	<i>Optional</i>	Data type of the metric. The accepted data types are: INT32, UINT32, DOUBLE, INT64, UINT64, STRING, BOOL and TIME.
Description	<i>Optional</i>	<i>Optional</i>	Description of the metric.
Label	<i>Optional</i>	<i>Optional</i>	Label of the metric.
Unit	<i>Optional</i>	<i>Optional</i>	Unit of the metric. For example: %, kbps, mbps and so on.
Key	Yes	Yes	This field identifies whether this metric is a key metric. The value -1 means it not a key and the value 0 or above means, this metric is a key metric. The value of key metrics define the instances of the class.
ClassName	Yes	Yes	Class name of the metric.
DataSource	Yes	Yes	The data source of the metric.
CollectorName	Yes	<i>Optional</i>	Collector name of the metric.
CollectorExec	<i>Optional</i>	<i>Optional</i>	Path to the collector of the metric with the executable command.
Interval	Yes	<i>Optional</i>	Collection interval for custom metrics.
Interface	Yes	<i>Optional</i>	Interface of the metric. Stream or Legacy. Stream is used when you want to stream metrics to hpsensor without logging them into Metrics Datastore and Legacy is used when you want to log the data into Metrics Datastore using the legacy interfaces like DDF or bulk submission and also stream it to hpsensor . If not specified, the default interface is Legacy.
value	<i>Must for key metric</i>	Yes	Value of the metric. The value specified should match with the DataType of the metric.

timestamp	<i>Must for key metric</i>	Yes	The epoch time when the metric value was collected.
dimensions	<i>Optional</i>	Yes	Key instance name for the metric.

Note: If you do not use **HP Ops OS Inst to Realtime Inst LTU**, then system performance metrics available for streaming will be limited. To get complete access to system performance metrics, use **HP Ops OS Inst to Realtime Inst LTU**.

Registration

The REST API `/v4/RTCM/Register` is used to post registration information to **hpsensor**. The registration information must contain the following objects:

1. Title
2. Version
3. Metric Payload consisting of the MetaData and Data Fields (for **Key** metric instances)

For example: Consider a use case where you want to post registration information to **hpsensor** to stream data for 2 custom metrics (1 metric and 1 key metric). For this scenario, the following registration information in v4 JSON format can be posted:

```
[{
  "Title": "Real Time Metrics",
  "Version": "v4",
  "Metric Payload": [
    {
      "MetaData": {
        "MetricName": "Metric_ACTIVE_PROC",
        "CategoryType": "Attribute",
        "DataType": "DOUBLE",
        "Description": "Metric_Description",
        "Label": "Active Proc",
```



```
        "Unit": "N/A",
        "Key": -1,
        "ClassName": "Example_Class",
        "DataSource": "Example_DataSouce",
        "CollectorName": "Example_Collector",
        "CollectorExec": "$0vDataDir$bin/instrumentation/Example_
collector.sh Collector",
        "Interface": "Stream"
    }
},
{
    "MetaData": {
        "MetricName": "Instance_NAME",
        "CategoryType": "Attribute",
        "DataType": "STRING",
        "Description": "Metric_Description",
        "Label": "Metric Name1",
        "Unit": "N/A",
        "Key": 0,
        "ClassName": "Example_Class",
        "DataSource": "Example_DataSource",
        "CollectorName": "Example_Collector",
        "CollectorExec": "$0vDataDir$bin/instrumentation/Example_
collector.sh",
        "Interface": "Stream"
    },
    "Data": {
```

```
    "Instances": [{
      "value": "inst_1",
      "timestamp": 1459724247
    }, {
      "value": "inst_2",
      "timestamp": 1459724247
    }
  ]
}
}
]
```

Use the REST API `/v4/RTCM/Register` to post the registration information to **hpsensor**. For more information on how to use this REST API, see the [usage section](#).

Data Submission

The REST API `/v4/RTCM/SubmitData` is used to submit custom metrics data to **hpsensor**. The submitted data must contain the following objects:

1. Title
2. Version
3. Metric Payload consisting of the MetaData and Data Fields (for all metrics)

Note: For data submission, instances must be specified using the dimensions tag for all metrics, else data submission will be discarded.

For example: Consider a use case where you want to submit custom metrics data to **hpsensor** to stream data for 2 custom metrics (1 metric and 1 key metric). For this scenario, the following data can be submitted in v4 JSON format:

```
[{
  "Title": "Real Time Metrics",
```

```
"Version": "v4",
"Metric Payload": [
  {
    "MetaData": {
      "MetricName": "Metric_ACTIVE_PROC",
      "CategoryType": "Attribute",
      "DataType": "DOUBLE",
      "Description": "Metric_Description",
      "Label": "Active Proc",
      "Unit": "N/A",
      "Key": -1,
      "ClassName": "Example_Class",
      "DataSource": "Example_DataSource"
    },
    "Data": {
      "Instances": [{
        "value": "0.0",
        "timestamp": 1459725391,
        "dimensions": {
          "Instance_NAME": "inst_1"
        }
      }, {
        "value": "0.0",
        "timestamp": 1459725391,
        "dimensions": {
          "Instance_NAME": "inst_2"
        }
      }
    ]
  }
]
```

```
        ]]  
    }  
},  
{  
  "MetaData": {  
    "MetricName": "Instance_NAME",  
    "CategoryType": "Attribute",  
    "DataType": "STRING",  
    "Description": "Metric_Description",  
    "Label": "Metric Name1",  
    "Unit": "N/A",  
    "Key": 0,  
    "ClassName": "Example_Class",  
    "DataSource": "Example_DataSource"  
  },  
  "Data": {  
    "Instances": [{  
      "value": "inst_1",  
      "timestamp": 1459725391,  
      "dimensions": {  
        "Instance_NAME": "inst_1"  
      }  
    }, {  
      "value": "inst_2",  
      "timestamp": 1459725391,  
      "dimensions": {  
        "Instance_NAME": "inst_2",
```

```
    }  
  }]  
}  
}]  
}]
```

Use the REST API `/v4/RTCM/SubmitData` to submit metrics data to **hpsensor**. For more information on how to use this REST API, see the [usage section](#).

Special Characters in Registration and Data Submission

hpsensor accepts data requests in URI format and special characters are likely to occur as part of MetricName, DataSource, ClassNames, or Instance Names. Special characters supported and not supported for various entities are as listed:

Entity	Special characters supported	Special characters not supported
MetricName/DataSource/ClassName	<code>_~!@\$%^&*()-+= {[]}_;<>.'?</code>	<code>/,#\:"</code>
InstanceName	<code>_~!@\$%^&*-+={} _.'?</code>	<code>/,#"[]();<></code>

Note: For Registration and Data Submission:

- Number of MetaData fields can vary.
- Number of instances of a metrics can vary and all metrics may not have the same number of instances.
- There can be one or more dimensions for an instance.
- Combination of `<*>` characters can be used as a wildcard in the **Metric Streaming Configuration** while filtering instances policy. For example: `<*>.company.com`
- The unsupported characters mentioned for InstanceName can be used for Registration and Data Submission, but cannot be used in **Metric Streaming Configuration** policy for exact match of these characters. Instead, you can use wildcard `<*>` to match any character.

Configuring Publish Interval

hpsensor will publish the custom and system performance metrics data to the target subscriber at a configured interval. By default, the publish interval is 10 seconds. You can use the following XPL variable to configure the default publish interval:

Variable	Namespace	Description	Restart Required?	Default Value	Type
PUBLISH_INTERVAL	hpsensor	This parameter sets the interval at which custom and system performance metrics data is published to the target subscriber.	No	10	Integer

To configure the default publish interval, follow the steps:

1. Log on to the Operations Agent node as an administrator.
2. Run the following command:

```
<OvInstallBinDir>ovconfchg -ns hpsensor -set PUBLISH_INTERVAL <value>
```

The default value is 10 (in seconds) and the minimum value allowed is 5 (in seconds).

Note: The publish interval set will be applicable for all the targets.

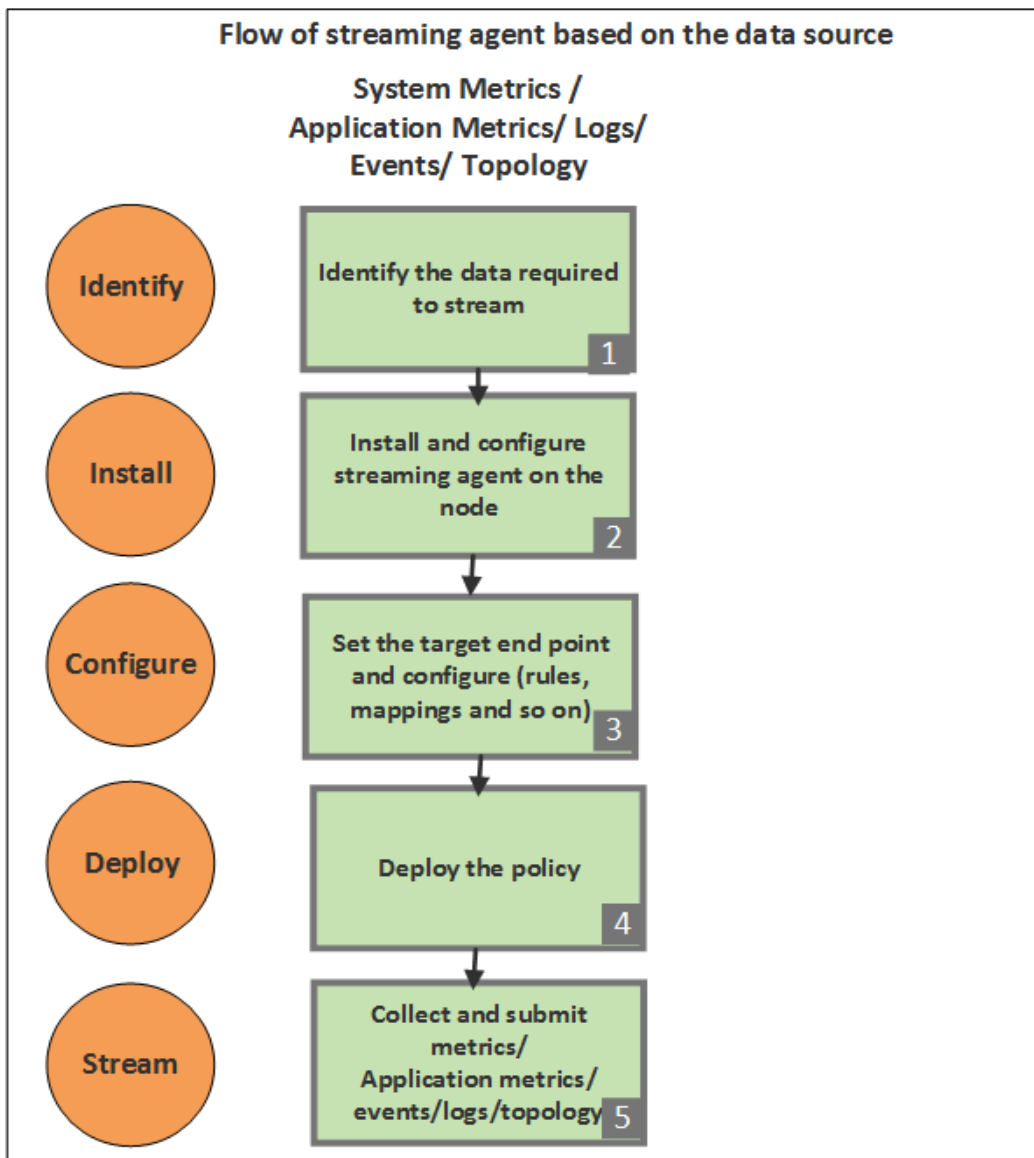
Resource Utilization and Scalability

The data submitted to **hpsensor** for Metric Streaming will be stored in memory and **hpsensor** keeps only last interval value of a metric. If you submit more data points then there will be significant amount of memory utilization. Hewlett Packard Enterprise recommends that the number of data points submitted are not more than 100000 for optimum memory utilization.

Chapter 20: Working of Streaming Edition of Operations Agent

Streaming agent collects and streams system performance metrics, application metrics, logs, events, and topology from third party domain managers and application monitoring components. The data processing and streaming functions are completely policy driven.

You can choose the streaming flows based on the data type (metrics/ logs/ events/ topology). The following diagram shows the workflow of streaming agent:



You can integrate data using one of the supported policy types based on data that you want to stream:

Table 1 Supported Policies

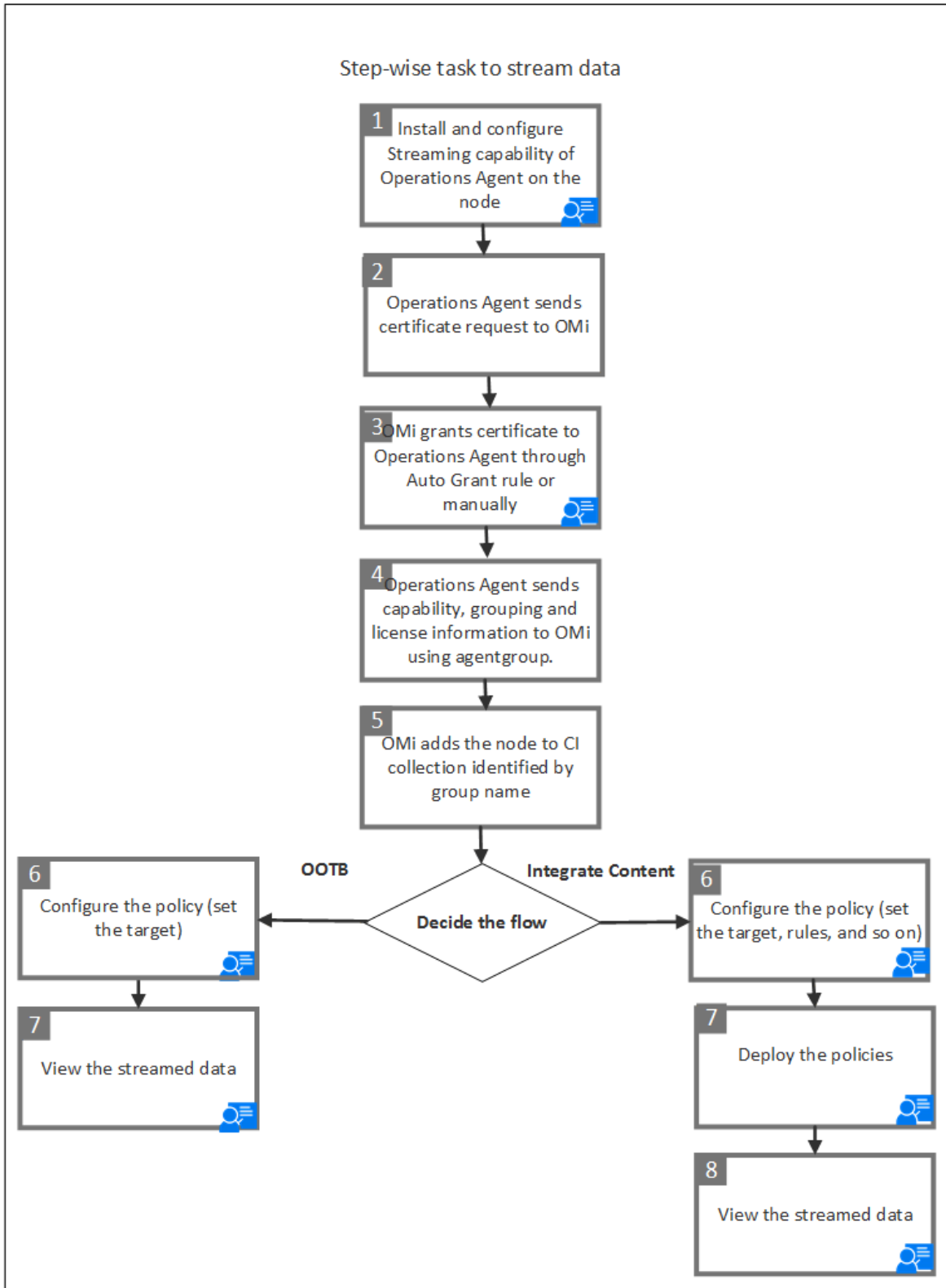
Type	Policy name
Metrics (system and application metrics)	Metric Streaming Configuration
	Metrics from Structured Log File and Data forwarding
	Metrics from REST WS
	Metrics from XML file
	Metrics from Perl script
Generic (metric/ log/ event)	Generic Output from Structured Log File and Data forwarding
	Generic Output from REST Web Service
	Generic Output from XML file
	Generic Output from Perl Script
	Generic Output from Windows Event Log
Event	Event from Structured Log File
	Event from XML file
	Event from Perl Script
	Event from REST Web Service
Topology	Topology from REST Web Service
	Topology from XML file

Stream and view data on the configured target end points

To stream metrics, events, logs or topology and view the results in the target end points, you can choose from the following:

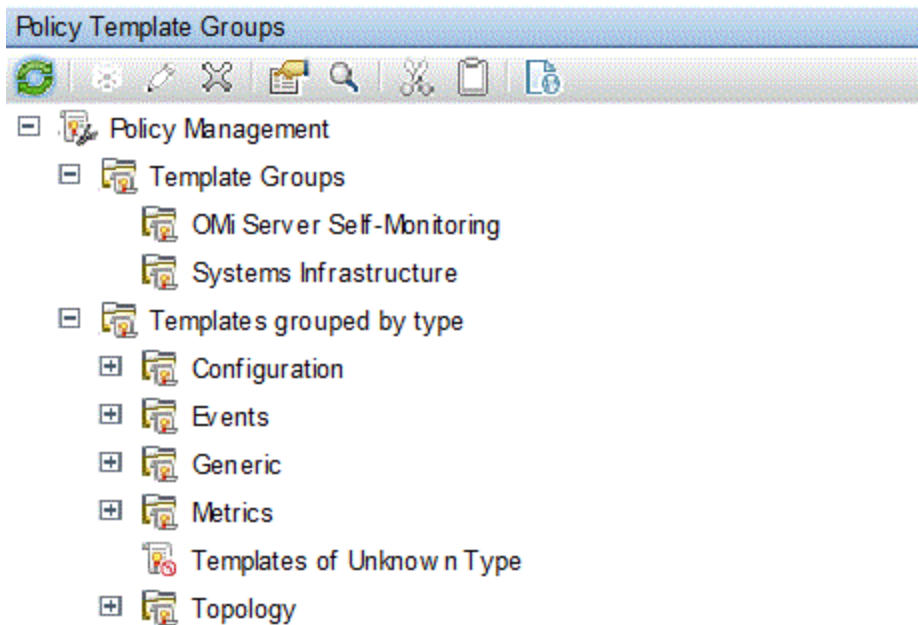
- Stream out of the box content from monitoring solutions (for example: OMi MP for Infrastructure)
- Integrate data (metrics/ logs/ events/ topology) from 3rd party domain managers and application monitoring solutions.

Step-wise pictorial representation for OOTB content or Integrate data from application monitoring solutions and 3rd party domain managers.



Alternatively, you can follow the steps to stream data (OOTB content)

1. Log on to the node as an administrator.
2. Install and configure streaming agent from the Operations Agent media. For more information, see the *Operations Agent Installation Guide*.
3. Log on to OMi.
4. Check the list of monitored nodes with streaming edition from **Administration > Setup and Maintenance > Monitored Nodes**.
5. Open **CI Type Manager > ConfigurationItem > CI Collection** and verify the default agent group **Operations Agents (streaming edition)** or the agent group name provided at the time of installation.
6. Open the Policy Template Manager, select **Administration > Monitoring > Policy Templates**. For more information about Policy Templates, see *OMi Help: Policy Templates*.



7. Click **Metric Streaming Configuration or Data Forwarding policy type** from **Administration > Monitoring > Policy Templates**, and then do of the following:
 - a. Edit the policy. `Sys_SystMetricStreaming Editor` or `Sys_DataForwarding` opens. For more information about editing policy, see *OMi Help: Metric Configuration* or *OMi Help: Configure Data Forwarding Policies*.
 - b. Select the metrics that you want to stream for `Sys_Metricstreaming Policy`. For `Sys_DataForwarding` policy, complete the configuration as required. For more information, see *OMi Help: Configure Data Forwarding Policies*.

8. In the Target endpoint, type the details such as Performance Engine server details.
9. To view the results of Metrics, Topology, Events of the streamed data, open the respective views (performance dashboard, event perspective, topology perspective) to view graphs and results of the host of components such as applications, system and network.

Part V: Transaction Tracking

Operations Agent can track transactions as they progress through applications. Performance Collection Component and GlancePlus work together to help you define and track transaction data from applications instrumented with Application Response Measurement (ARM) calls.

Transactions are monitored only after transaction logging is turned on in the **parm** file. The Transaction Tracking daemon, `tttd`, and the Measurement Interface daemon, `midaemon`, collect and synchronize the transaction data for your application as it runs. The data is stored in the `midaemon`'s shared memory segment where it can be used by Performance Collection Component or GlancePlus.

Transaction tracking provides a client view of elapsed time from the beginning to the end of a transaction, helps you manage service level agreements (SLAs), and generate alarms when Service level objectives (SLOs) exceed the conditions defined in the `alarmdef` file.

Chapter 21: What is Transaction Tracking?

This chapter describes:

- [Improving Performance Management](#)
- [A Scenario: Real Time Order Processing](#)
- [Monitoring Transaction Data](#)

Improving Performance Management

You can improve your ability to manage system performance with the transaction tracking capability of the Operations Agent and GlancePlus.

As the number of distributed mission-critical business applications increases, application and system managers need more information to tell them how their distributed information technology (IT) is performing.

- Has your application stopped responding?
- Is the application response time unacceptable?
- Are your service level objectives (SLOs) being met?

The transaction tracking capabilities of Performance Collection Component and GlancePlus allow IT managers to build an end-to-end manageability of their client/server IT environment in business transaction terms. With Performance Collection Component, you can define what a business transaction is and capture transaction data that makes sense in the context of *your* business.

When your applications are instrumented with the standard Application Response Measurement (ARM) API calls, these products provide extensive transaction tracking and end-to-end management capabilities across multi-vendor platforms.

Benefits of Transaction Tracking

- Provides a client view of elapsed time from the beginning to the end of a transaction.
- Provides transaction data.

- Helps you manage service level agreements (SLAs).

These topics are discussed in more detail in the remainder of this section.

Client View of Transaction Times

Transaction tracking provides you with a client view of elapsed time from the beginning to the end of a transaction. When you use transaction tracking in your Information Technology (IT) environment, you see the following benefits:

- You can accurately track the number of times each transaction executes.
- You can see how long it takes for a transaction to complete, rather than approximating the time as happens now.
- You can correlate transaction times with system resource utilization.
- You can use your own business deliverable production data in system management applications, such as data used for capacity planning, performance management, accounting, and charge-back.
- You can accomplish application optimization and detailed performance troubleshooting based on a real unit of work (your transaction), rather than representing actual work with abstract definitions of system and network resources.

Transaction Data

When Application Response Measurement (ARM) API calls have been inserted in an application to mark the beginning and end of each business transaction, you can then use the following resource and performance monitoring tools to monitor transaction data:

- Performance Collection Component provides the registration functionality needed to log, report, and detect alarms on transaction data. Transaction data can be viewed in the Performance Manager, Glance, or by exporting the data from Performance Collection Component log files into files that can be accessed by spreadsheet and other reporting tools.
- Performance Manager graphs performance data for short-term troubleshooting and for examining trends and long-term analysis.
- Glance displays detailed real time data for monitoring your systems and transactions moment by moment.

- Performance Manager, Glance, or the Operations Manager message browser allow you to monitor alarms on service level compliance.

Individual transaction metrics are described in [Chapter 22, Transaction Metrics](#).

Service Level Objectives

Service level objectives (SLOs) are derived from the stated service levels required by business application users. SLOs are typically based on the development of the service level agreement (SLA). From SLOs come the actual metrics that Information Technology resource managers need to collect, monitor, store, and report on to determine if they are meeting the agreed upon service levels for the business application user.

An SLO can be as simple as monitoring the response time of a simple transaction or as complex as tracking system availability.

A Scenario: Real Time Order Processing

Imagine a successful television shopping channel that employs hundreds of telephone operators who take orders from viewers for various types of merchandise. Assume that this enterprise uses a computer program to enter the order information, check merchandise availability, and update the stock inventory. We can use this fictitious enterprise to illustrate how transaction tracking can help an organization meet customer commitments and SLOs.

Based upon the critical tasks, the customer satisfaction factor, the productivity factor, and the maximum response time, resource managers can determine the level of service they want to provide to their customers.

[Chapter 23, Transaction Tracking Examples](#) contains a pseudocode example of how ARM API calls can be inserted in a sample order processing application so that transaction data can be monitored with Performance Collection Component and Glance.

Requirements for Real Time Order Processing

To meet SLOs in the real time order processing example described above, resource managers must keep track of the duration required to complete the following critical tasks:

- Enter order information
- Query merchandise availability
- Update stock inventory

The key customer satisfaction factor for customers is how quickly the operators can take their order.

The key productivity factor for the enterprise is the number of orders that operators can complete each hour.

To meet the customer satisfaction and productivity factors, the response times of the transactions that access the inventory database, adjust the inventory, and write the record back must be monitored for compliance to established SLOs. For example, resource managers may have established an SLO for this application that 90 percent of the transactions must be completed in five seconds or less.

Preparing the Order Processing Application

ARM API calls can be inserted into the order processing application to create transactions for `inventory response` and `update inventory`. Note that the ARM API calls must be inserted by application programmers *prior* to compiling the application. See [Chapter 23, Transaction Tracking Examples](#) for an example order processing program (written in pseudocode) that includes ARM API calls that define various transactions.

For more information on instrumenting applications with ARM API calls, see the [Application Response Measurement 2.0 API Guide](#).

Monitoring Transaction Data

When an application that is instrumented with ARM API calls is installed and running on your system, you can monitor transaction data with Performance Collection Component, GlancePlus, or Performance Manager.

... with Performance Collection Component

Using Performance Collection Component, you can collect and log data for named transactions, monitor trends in your SLOs over time, and generate alarms when SLOs are exceeded. Once these trends have been identified, Information Technology costs can be allocated based on transaction volumes. Performance Collection Component alarms can be configured to activate a technician's

pager, so that problems can be investigated and resolved immediately. For more information, see [Chapter 24, Advanced Features](#).

Performance Collection Component is required for transaction data to be viewed in Performance Manager.

... with Performance Manager

Performance Manager receives alarms and transaction data from Performance Collection Component. For example, you can configure Performance Collection Component so that when an order processing application takes too long to check stock, Performance Manager receives an alarm and sends a warning to the resource manager's console as an alert of potential trouble.

In Performance Manager, you can select **TRANSACTION** from the Class List window for a data source, then **graph transaction metrics** for various transactions. For more information, see *Performance Manager Online Help*.

... with GlancePlus

Use GlancePlus to monitor up-to-the-second transaction response time and whether or not your transactions are performing within your established SLOs. GlancePlus helps you identify and resolve resource bottlenecks that may be impacting transaction performance. For more information, see the *GlancePlus Online Help*, which is accessible through the GlancePlus Help menu.

Guidelines for Using ARM

Instrumenting applications with the ARM API requires some careful planning. In addition, managing the environment that has ARMed applications in it is easier if the features and limitations of ARM data collection are understood. Here is a list of areas that could cause some confusion if they are not fully understood.

1. In order to capture ARM metrics, `ttd` and `midaemon` must be running. For Performance Collection Component, the `oacore` collector must be running to log ARM metrics. The `ovpa start` script starts all required processes. Likewise, Glance starts `ttd` and `midaemon` if they are not already active. (See [Transaction Tracking Daemon \(ttd\) in Chapter 19](#))
2. Re-read the transaction configuration file, `ttd.conf`, to capture any newly-defined transaction names. (See [Transaction Configuration File \(ttd.conf\) in Chapter 19](#))
3. Performance Collection Component, user applications, and `ttd` must be restarted to capture any *new* or modified transaction ranges and service level objectives (SLOs). (See [Adding New Applications in Chapter 19](#))

4. Strings in user-defined metrics are ignored by Performance Collection Component. Only the first six non-string user-defined metrics are logged. (See [How Data Types Are Used in Chapter 24](#))
5. Using dashes in the transaction name has limitations if you are specifying an alarm condition for that transaction. (See "... with Performance Collection Component" in the section [Alarms in Chapter 20](#))
6. Performance Collection Component will only show the first 60 characters in the application name and transaction name. (See [Specifying Application and Transaction Names in Chapter 19](#))
7. Limit the number of unique transaction names that are instrumented. (See [Limits on Unique Transaction in Chapter 20](#))
8. Do not allow ARM API function calls to affect the execution of an application from an end-user perspective. (See [ARM API Call Status Returns in Chapter 19](#))
9. Use shared libraries for linking. (See the section "[C Compiler Option Examples by Platform](#)" on [page 353](#))

Chapter 22: How Transaction Tracking Works

The following components of Performance Collection Component and GlancePlus work together to help you define and track transaction data from applications instrumented with Application Response Measurement (ARM) calls.

- The Measurement Interface daemon, `midaemon`, is a daemon process that monitors and reports transaction data to its shared memory segment where the information can be accessed and reported by Performance Collection Component, Performance Manager, and GlancePlus. On HP-UX systems, the `midaemon` also monitors system performance data.
- The transaction configuration file, `/var/opt/perf/ttd.conf`, is used to define transactions and identify the information to monitor for each transaction.
- The Transaction Tracking daemon, `ttd`, reads, registers, and synchronizes transaction definitions from the transaction configuration file, `ttd.conf`, with the `midaemon`.

Support of ARM 2.0

ARM 2.0 is a superset of the previous version of Application Response Measurement. The new features that ARM 2.0 provides are user-defined metrics, transaction correlation, and a logging agent. Performance Collection Component and GlancePlus support user-defined metrics and transaction correlation but *do not* support the logging agent.

However, you may want to use the logging agent to test the instrumentation in your application. The source code for the logging agent, `logagent.c`, is included in the ARM 2.0 Software Developers Kit (SDK) that is available from the following web site:

<http://regions.cmg.org/regions/cmgarmlw>

For information about using the logging agent, see the [Application Response Measurement 2.0 API Guide](#).

Note: The *Application Response Measurement 2.0 API Guide* uses the term “application-defined metrics” instead of “user-defined metrics”.

Support of ARM API Calls

The Application Response Measurement (ARM) API calls listed below are supported in Performance Collection Component and GlancePlus.

<code>arm_init()</code>	Names and registers the application and (optionally) the user.
<code>arm_getid()</code>	Names and registers a transaction class, and provides related transaction information. Defines the context for user-defined metrics.
<code>arm_start()</code>	Signals the start of a unique transaction instance.
<code>arm_update()</code>	Updates the values of a unique transaction instance.
<code>arm_stop()</code>	Signals the end of a unique transaction instance.
<code>arm_end()</code>	Signals the end of the application.

See your current [Application Response Measurement 2.0 API Guide](#) and the `arm (3)` man page for information on instrumenting applications with ARM API calls as well as complete descriptions of the calls and their parameters. For commercial applications, check the product documentation to see if the application has been instrumented with ARM API calls.

For important information about required libraries, see the [Transaction Libraries](#) later in this manual.

arm_complete_transaction Call

In addition to the ARM 2.0 API standard, the ARM agent supports the `arm_complete_transaction` call. This call can be used to mark the end of a transaction that has completed when the start of the transaction could not be delimited by an `arm_start` call. The `arm_complete_transaction` call takes as a parameter the response time of the completed transaction instance.

In addition to signaling the end of a transaction instance, additional information about the transaction can be provided in the optional data buffer. See the `arm (3)` man page for more information on this optional data as well as a complete description of this call and its parameters.

Sample ARM-Instrumented Applications

For examples of how ARM API calls are implemented, see the sample ARM-instrumented applications, `armsample1.c`, `armsample2.c`, `armsample3.c`, and `armsample4.c`, and their build script, `Make.armsample`, in the `<InstallDir>/examples/arm/` directory.

- `armsample1.c` shows the use of simple standard ARM API calls.
- `armsample2.c` also shows the use of simple standard ARM API calls. It is similar in structure to `armsample1.c`, but is interactive.
- `armsample3.c` provides examples of how to use the user-defined metrics and the transaction correlator, provided by version 2.0 of the ARM API. This example simulates a client/server application where both server and client perform a number of transactions. (Normally application client and server components would exist in separate programs, but they are put together for simplicity.)

The client procedure starts a transaction and requests an ARM correlator from its `arm_start` call. This correlator is saved by the client and passed to the server so that the server can use it when it calls `arm_start`. The performance tools running on the server can then use this correlator information to distinguish the different clients making use of the server.

Also shown in this program is the mechanism for passing user-defined metric values into the ARM API. This allows you to not only see the response times and service-level information in the performance tools, but also data which may be important to the application itself. For example, a transaction may be processing different size requests, and the size of the request could be a user-defined metric. When the response times are high, this user-defined metric could be used to see if long response times correspond to bigger size transaction instances.

- `armsample4.c` provides an example of using user-defined metrics in ARM calls. Different metric values can be passed through `arm_start`, `arm_update`, and `arm_stop` calls. Alternatively, `arm_complete_transaction` can be used where a tran cannot be delimited by `start/stop` calls.

Specifying Application and Transaction Names

Although ARM allows a maximum of 128 characters each for application and transaction names in the `arm_init` and `arm_getid` API calls, Performance Collection Component shows *only* a maximum of 60 characters. All characters beyond the first 60 will not be visible. However, GlancePlus allows you to view up to 128 characters.

Performance Collection Component applies certain limitations to how application and transaction names are shown in extracted or exported transaction data. These rules also apply to viewing application and transaction names in Performance Manager.

The application name *always* takes precedence over the transaction name. For example, if you are exporting transaction data that has a 65-character application name and a 40-character transaction name, *only* the application name is shown. The last five characters of the application name are not visible.

For another example, if an application name contains 32 characters and the transaction name has 40 characters, Performance Collection Component shows the entire application name but the transaction name appears truncated. A total of 60 characters are shown. Fifty-nine characters are allocated to the application and transaction names and one character is allocated to the underscore (`_`) that separates the two names. This is how the application name “WarehouseInventoryApplication” and the transaction name “CallFromWestCoastElectronicSupplier” would appear in Performance Collection Component or Performance Manager:

```
WarehouseInventoryApplication_CallFromWestCoastElectronicSup
```

Note: The 60-character combination of application name and transaction name must be unique if the data is to be viewed with Performance Manager.

Transaction Tracking Daemon (ttd)

The Transaction Tracking daemon, `ttd`, reads, registers, and synchronizes transaction definitions from `ttd.conf` with `midaemon`.

`ttd` is started when you start up Performance Collection Component's **oacore** data collector with the `ovpa start` command. `ttd` runs in background mode when dispatched, and errors are written to the file `/var/opt/perf/status.ttd`.

`midaemon` must also be running to process the transactions and to collect performance metrics associated with these transactions (see next page).

Caution: We strongly recommend that you do not stop `ttd`.

If you must stop `ttd`, any ARM-instrumented applications that are running *must* also be stopped before you restart `ttd` and Performance Collection Component processes. `ttd` must be running to capture all `arm_init` and `arm_getid` calls being made on the system. If `ttd` is stopped and restarted, transaction IDs returned by these calls will be repeated, thus invalidating the ARM metrics

Use the `ovpa` script to start Performance Collection Component processes to ensure that the processes are started in the correct order. `ovpa stop` will *not* shut down `ttd`. If `ttd` must be shut down for a reinstall of any performance software, use the command `<InstallDir>/bin/ttd -k`. However, we do not recommend that you stop `ttd`, except when reinstalling Performance Collection Component.

If Performance Collection Component is not on your system, GlancePlus starts `midaemon`. `midaemon` then starts `ttd` if it is not running *before* `midaemon` starts processing any measurement data.

See the `ttd` man page for complete program options.

ARM API Call Status Returns

The `ttd` process must always be running in order to register transactions. If `ttd` is killed for any reason, while it is not running, `arm_init` or `arm_getid` calls will return a “failed” return code. If `ttd` is subsequently restarted, new `arm_getid` calls may re-register the same transaction IDs that are already being used by other programs, thus causing invalid data to be recorded.

When `ttd` is killed and restarted, ARM-instrumented applications may start getting a return value of `-2` (`TT_TTDNOTRUNNING`) and an `EPIPEerrno` error on ARM API calls. When your application initially starts, a client connection handle is created on any initial ARM API calls. This client handle allows your application to communicate with the `ttd` process. When `ttd` is killed, this connection is no longer valid and the next time your application attempts to use an ARM API call, you may get a return value of `TT_TTDNOTRUNNING`. This error reflects that the *previous* `ttd` process is no longer running even though there is another `ttd` process running. (Some of the ARM API call returns are explained in the *arm* (3) man page.)

To get around this error, you must restart your ARM-instrumented applications if `ttd` is killed. First, stop your ARMed applications. Next, restart `ttd` (using `<InstallDir>/bin/ovpa start` or `<InstallDir>/bin/ttd`), and then restart your applications. The restart of your application causes the creation of a new client connection handle between your application and the `ttd` process.

Some ARM API calls will not return an error if the `midaemon` has an error. For example, this would occur if the `midaemon` has run out of room in its shared memory segment. The performance metric `GBL_TT_OVERFLOW_COUNT` will be `> 0`. If an overflow condition occurs, you may want to shut down any performance tools that are running (except `ttd`) and restart the `midaemon` using the `-smdvss` option to specify more room in the shared memory segment. (For more information, see the *midaemon* man page.)

We recommend that your applications be written so that they continue to execute even if ARM errors occur. ARM status should not affect program execution.

The number of active client processes that can register transactions with `ttd` via the `arm_getid` call is limited to the `maxfiles` kernel parameter. This parameter controls the number of open files per process. Each client registration request results in `ttd` opening a socket (an open file) for the RPC connection. The socket is closed when the client application terminates. Therefore, this limit affects only the number of active clients that have registered a transaction via the `arm_getid` call. Once this limit is reached, `ttd` will return `TT_TTDNOTRUNNING` to a client's `arm_getid` request. The `maxfiles` kernel parameter can be increased to raise this limit above the number of active applications that will register transactions with `ttd`.

Measurement Interface Daemon (midaemon)

The Measurement Interface daemon, `midaemon`, is a low-overhead process that continuously collects system performance information. The `midaemon` must be running for Performance Collection Component to collect transaction data or for GlancePlus to report transaction data. It starts running when you run the `oacore` process or `perfd` process or when starting GlancePlus.

Performance Collection Component and GlancePlus require both the `midaemon` and `ttd` to be running so that transactions can be registered and tracked. The `ovpa` script starts and stops Performance Collection Component processing, including the `midaemon`, in the correct order. GlancePlus starts the `midaemon`, if it is not already running. The `midaemon` starts `ttd`, if it is not already running.

See the "[CPU Overhead](#)" on page 329 section later in this manual for information on the `midaemon` CPU overhead.

See the `midaemon` man page for complete program options.

Note: If `pid_max`¹ is set to a very high value (approximately 4 million), the Operations Agent performance viewing tools (`cps`, `glance`) using `midaemon` allocates virtual memory for `pid_max` entries. These tools using `midaemon` consumes more virtual memory to monitor all the process data.

For example:

If the number of processes running in a system is 100, the virtual memory can be allocated only for 100 processes but when `pid_max` is set to a very high value (approximately 4 million), the virtual memory can be allocated for all `pid_max` processes irrespective of the number of processes running in the system.

¹`pid_max` is the maximum number of Process IDs (PIDs) that can run in a system. The maximum limit of `pid_max` (`PID_MAX_LIMIT`) is approximately 4 million.

Transaction Configuration File (ttd.conf)

The transaction configuration file, `/var/opt/perf/ttd.conf`, allows you to define the application name, transaction name, the performance distribution ranges, and the service level objective you want to meet for each transaction. The `ttd` reads `ttd.conf` to determine how to register each transaction.

Customization of `ttd.conf` is optional. The default configuration file that ships with Performance Collection Component causes *all* transactions instrumented in any application to be monitored.

If you are using a commercial application and don't know which transactions have been instrumented in the application, collect some data using the default `ttd.conf` file. Then look at the data to see which transactions are available. You can then customize the transaction data collection for that application by modifying `ttd.conf`.

Adding New Applications

If you add new ARMed applications to your system that use the default `slo` and `range` values from the `tran=*` line in your `ttd.conf` file, you don't need to do anything to incorporate these new transactions. (See the [Configuration File Keywords](#) section for descriptions of `tran`, `range`, and `slo`.) The new transactions will be picked up automatically. The `slo` and `range` values from the `tran` line in your `ttd.conf` file will be applied to the new transactions.

Adding New Transactions

After making additions to the `ttd.conf` file, you must perform the following steps to make the additions effective:

- Stop all applications.
- Execute the `ttd -hup -mi` command as root.

The above actions cause the `ttd.conf` file to be re-read and registers the new transactions, along with their `slo` and `range` values, with `ttd` and the `midaemon`. The re-read will not change the `slo` or `range` values for any transactions that were in the `ttd.conf` file prior to the re-read.

Changing the Range or SLO Values

If you need to change the SLO or range values of existing transactions in the `ttd.conf` file, you must do the following:

- Stop all ARMed applications.
- Stop the **oacore** collector using `ovpa stop`.
- Stop any usage of Glance.
- Stop the `ttd` by issuing the command `ttd -k`.
- Once you have made your changes to the `ttd.conf` file:
- Restart **oacore** using `ovpa start`.
- Restart your ARMed applications.

Configuration File Keywords

The `/var/opt/perf/ttd.conf` configuration file associates transaction names with transaction attributes that are defined by the keywords in Table 1.

Table 1: Configuration File Keywords

Keyword	Syntax	Usage
<code>tran</code>	<code>tran=<i>transaction_name</i></code>	Required
<code>slo</code>	<code>slo=<i>sec</i></code>	Optional
<code>range</code>	<code>range=<i>sec</i> [<i>,sec,...</i>]</code>	Optional

These keywords are described in more detail below.

tran

Use `tran` to define your transaction name. This name must correspond to a transaction that is defined in the `arm_getid` API call in your instrumented application. You must use the `tran` keyword before you can specify the optional attributes `range` or `slo`. `tran` is the only required keyword within the configuration file. A trailing asterisk (*) in the transaction name causes a wild card pattern match to be

performed when registration requests are made against this entry. Dashes can be used in a transaction name. However, spaces cannot be used in a transaction name.

The transaction name can contain a maximum of 128 characters. However, only the first 60 characters are visible in Performance Collection Component. GlancePlus can display 128 characters in specific screens.

The default `ttd.conf` file contains several entries. The first entries define transactions used by the Performance Collection Component data collector **oacore**, which is instrumented with ARM API calls. The file also contains the entry `tran=*`, which registers all other transactions in applications instrumented with ARM API or Transaction Tracker API calls.

range

Use `range` to specify the transaction performance distribution ranges. Performance distribution ranges allow you to distinguish between transactions that take different lengths of time to complete and to see how many successful transactions of each length occurred. The ranges that you define appear in the GlancePlus Transaction Tracking window.

Each value entered for `sec` represents the upper limit in seconds for the transaction time for the range. The value may be an integer or real number with a maximum of six digits to the right of the decimal point. On HP-UX, this allows for a precision of one microsecond (.000001 seconds). On other platforms, however, the precision is ten milliseconds (0.01 seconds), so only the first two digits to the right of the decimal point are recognized.

A maximum of ten ranges are supported for each transaction you define.

You can specify up to nine ranges. One range is reserved for an overflow range, which collects data for transactions that take longer than the largest user-defined range. If you specify more than nine ranges, the first nine ranges are used and the others are ignored.

If you specify fewer than nine ranges, the first unspecified range becomes the overflow range. Any remaining unspecified ranges are not used. The unspecified range metrics are reported as 0.000. The first corresponding unspecified count metric becomes the overflow count. Remaining unspecified count metrics are always zero (0).

Ranges must be defined in ascending order (see examples later in this chapter).

slo

Use `slo` to specify the service level objective (SLO) in seconds that you want to use to monitor your performance service level agreement (SLA).

As with the `range` keyword, the value may be an integer or real number, with a maximum of six digits to the right of the decimal point. On HP-UX, this allows for a precision of one microsecond (.000001 seconds). On other platforms, however, the precision is ten milliseconds (0.01 seconds), so only the first two digits to the right of the decimal point are recognized.

Note that even though transactions can be sorted with one microsecond precision on HP-UX, transaction times are displayed with 100 microsecond precision.

Configuration File Format

The `ttd.conf` file can contain two types of entries: general transactions and application-specific transactions.

General transactions should be defined in the `ttd.conf` file before any application is defined. These transactions will be associated with all the applications that are defined. The default `ttd.conf` file contains one general transaction entry and entries for the **oacore** collector that is instrumented with ARM API calls.

```
tran=* range=0.5, 1, 2, 3, 5, 10, 30, 120, 300 slo=5.0
```

Optionally, each application can have its own set of transaction names. These transactions will be associated *only* with that application. The application name you specify must correspond to an application name defined in the `arm_init` API call in your instrumented application. Each group of application-specific entries must begin with the name of the application enclosed in brackets. For example:

```
[AccountRec]
```

```
tran=acctOne range=0.01, 0.03, 0.05
```

The application name can contain a maximum of 128 characters. However, only the first 60 characters are visible in Performance Collection Component. Glance can display 128 characters in specific screens.

If there are transactions that have the same name as a “general” transaction, the transaction listed under the application will be used.

For example:

```
tran=abc range=0.01, 0.03, 0.05 slo=0.10
```

```
tran=xyz range=0.02, 0.04, 0.06 slo=0.08
```

```
tran=t* range=0.01, 0.02, 0.03
```

```
[AccountRec]
```

```
tran=acctOne range=0.04, 0.06, 0.08
```

```
tran=acctTwo range=0.1, 0.2
```

```
tran=t* range=0.03, 0.5
```

```
[AccountPay]
```

```
[GenLedg]
```

```
tran=GenLedgOne range=0.01
```

In the example above, the first three transactions apply to all of the three applications specified.

The application [AccountRec] has the following transactions: acctOne, acctTwo, abc, xyz, and t*. One of the entries in the general transaction set also has a wild card transaction named "t*". In this case, the "t*" transaction name for the AccountRec application will be used; the one in the general transaction set is ignored.

The application [AccountPay] has only transactions from the general transactions set.

The application [GenLedg] has transactions GenLedgOne, abc, xyz, and t*.

The ordering of transactions names makes no difference within the application.

For additional information about application and transaction names, see the section ["Specifying Application and Transaction Names"](#) on page 318 in this chapter.

Configuration File Examples

Example 1

```
tran=* range=0.5,1,2,3,5,10,30,12,30 slo=5.0
```

The "*" entry is used as the default if none of the entries match a registered transaction name. These defaults can be changed on each system by modifying the "*" entry. If the "*" entry is missing, a default set of registration parameters are used that match the initial parameters assigned to the "*" entry above.

Example 2

```
[MANufactr]
```

```
tran=MFG01 range=1,2,3,4,5,10 slo=3.0
```

```
tran=MFG02 range=1,2.2,3.3,4.0,5.5,10 slo=4.5
```

```
tran=MFG03
```

```
tran=MFG04 range=1,2.2,3.3,4.0,5.5,10
```

Transactions for the MANUFACT application, MFG01, MFG02, and MFG04, each use their own unique parameters. The MFG03 transaction does not need to track time distributions or service level objectives so it does not specify these parameters.

Example 3

```
[Financial]
```

```
tran=FIN01
```

```
tran=FIN02 range=0.1,0.5,1,2,3,4,5,10,20 slo=1.0
```

```
tran=FIN03 range=0.1,0.5,1,2,3,4,5,10,20 slo=2.0
```

Transactions for the Financial application, FIN02 and FIN03, each use their own unique parameters. The FIN01 transaction does not need to track time distributions or service level objectives so it does not specify these parameters.

Example 4

```
[PERSONL]
```

```
tran=PERS* range=0.1,0.5,1,2,3,4,5,10,20 slo=1.0
```

```
tran=PERS03 range=0.1,0.2,0.5,1,2,3,4,5,10,20 slo=0.8
```

The PERS03 transaction for the PERSONL application uses its own unique parameters while the remainder of the personnel transactions use a default set of parameters unique to the PERSONL application.

Example 5

```
[ACCOUNTS]
```

```
tran=ACCT_* slo=1.0
```

```
tran=ACCT_REC range=0.5,1,2,3,4,5,10,20 slo=2.0
```

```
tran=ACCT_PAY range=0.5,1,2,3,4,5,10,20 slo=2.0
```

Transactions for the ACCOUNTS application, ACCT_REC and ACCT_PAY, each use their own unique parameters while the remainder of the accounting transactions use a default set of parameters unique

to the accounting application. Only the accounts payable and receivable transactions need to track time distributions. The order of transaction names makes no difference within the application.

Overhead Considerations for Using ARM

The current versions of Performance Collection Component and GlancePlus contain modifications to their measurement interface that support additional data required for ARM 2.0. These modifications can result in increased overhead for performance management. You should be aware of overhead considerations when planning ARM instrumentation for your applications.

The overhead areas are discussed in the remainder of this chapter.

Guidelines

Here are some guidelines to follow when instrumenting your applications with the ARM API:

- The total number of separate transaction IDs should be limited to not more than 4,000. Generally, it is cheaper to have multiple instances of the same transaction than it is to have single instances of different transactions. Register *only* those transactions that will be actively monitored.
- Although the overhead for the `arm_start` and `arm_stop` API calls is very small, it can increase when there is a large volume of transaction instances. More than a few thousand `arm_start` and `arm_stop` calls per second on most systems can significantly impact overall performance.
- Request ARM correlators *only* when using ARM 2.0 functionality. (For more information about ARM correlators, see the “Advanced Topics” section in the [Application Response Measurement 2.0 API Guide](#). The overhead for producing, moving, and monitoring correlator information is significantly higher than for monitoring transactions that are not instrumented to use the ARM 2.0 correlator functionality.
- Larger string sizes (applications registering lengthy transaction names, application names, and user-defined string metrics) will impose additional overhead.

Disk I/O Overhead

The performance management software does not impose a large disk overhead on the system. Glance generally does not log its data to disk. Performance Collection Component's collector daemon, **oacore** generates disk database files.

CPU Overhead

A program instrumented with ARM calls will generally not run slower because of the ARM calls. This assumes that the rate of `arm_getid` calls is lower than one call per second, and the rate of `arm_start` and `arm_stop` calls is lower than a few thousand per second. More frequent calls to the ARM API should be avoided.

Most of the additional CPU overhead for supporting ARM is incurred inside of the performance tool programs and daemons themselves. The `midaemon` CPU overhead rises slightly but not more than two percent more than it was with ARM 1.0. If the `midaemon` has been requested to track per-transaction resource metrics, the overhead per transaction instance may be twice as high as it would be without tracking per-transaction resource metrics. (You can enable the tracking of per-transaction resource metrics by setting the `log_transaction=resource` flag in the `parm` file.) In addition, Glance and **oacore** CPU overhead will be slightly higher on a system with applications instrumented with ARM 2.0 calls. Only those applications that are instrumented with ARM 2.0 calls that make extensive use of correlators and/or user-defined metrics will have a significant performance impact on the `midaemon`, **oacore**, or Glance.

An `midaemon` overflow condition can occur when usage exceeds the available default shared memory. This results in:

- No return codes from the ARM calls once the overflow condition occurs.
- Display of incorrect metrics, including blank process names.
- Errors being logged in `status.mi` (for example, "out of space").

Memory Overhead

Programs that are making ARM API calls will not have a significant impact in their memory virtual set size, except for the space used to pass ARM 2.0 correlator and user-defined metric information. These buffers, which are explained in the [Application Response Measurement 2.0 API Guide](#), should not be a significant portion of a process's memory requirements.

There is additional virtual set size overhead in the performance tools to support ARM 2.0. The `midaemon` process creates a shared memory segment where ARM data is kept internally for use by Performance Collection Component and GlancePlus. The size of this shared memory segment has grown, relative to the size on releases with ARM 1.0, to accommodate the potential for use by ARM 2.0. By default on most systems, this shared memory segment is approximately 11 megabytes in size. This segment is not all resident in physical memory unless it is required. Therefore, this should not be a

significant impact on most systems that are not already memory-constrained. The memory overhead of `midaemon` can be tuned using special startup parameters (see the `midaemon` man page).

Chapter 23: Getting Started with Transactions

This chapter gives you the information you need to begin tracking transactions and your service level objectives. For detailed reference information, see ["How Transaction Tracking Works"](#) on page 316. See ["Transaction Tracking Examples"](#) on page 340 for examples.

Before you start

Performance Collection Component provides the `libarm.*` shared library in the following locations:

Platform	Path
IBM RS/6000	<code>/usr/lpp/perf/lib/</code>
Other UNIX platforms	<code>/opt/perf/lib/</code>

If you do not have Performance Collection Component installed on your system and if `libarm.*` doesn't exist in the path indicated above for your platform, see ["C Compiler Option Examples by Platform"](#) on page 353 at the end of this manual. See also "The ARM Shared Library (`libarm`)" section in the [Application Response Measurement 2.0 API Guide](#) for information on how to obtain it. For a description of `libarm`, see ["ARM Library \(`libarm`\)"](#) on page 349 at the end of this manual.

Setting Up Transaction Tracking

Follow the procedure below to set up transaction tracking for your application. These steps are described in more detail in the remainder of this section.

1. Define SLOs by determining what key transactions you want to monitor and the response level you expect (*optional*).
2. To monitor transactions in Performance Collection Component and Performance Manager, make sure that the Performance Collection Component `parm` file has transaction logging turned on. Then start or restart Performance Collection Component to read the updated `parm` file.

Editing the `parm` file is *not* required to see transactions in GlancePlus. However, `ttd` *must* be running in order to see transactions in GlancePlus. Starting GlancePlus will automatically start `ttd`.

3. Run the application that has been instrumented with ARM API calls that are described in this manual and the [Application Response Measurement 2.0 API Guide](#).
4. Use Performance Collection Component or Performance Manager to look at the collected transaction data, or use GlancePlus to view current data. If the data isn't visible in Performance Manager, close the data source and then reconnect to it.
5. Customize the configuration file, `ttd.conf`, to modify the way transaction data for the application is collected (*optional*).
6. After making additions to the `ttd.conf` file, you must perform the following steps to make the additions effective:
 - a. Stop all ARMed applications.
 - b. Execute the `ttd -hup -mi` command as root.

These actions re-read the `ttd.conf` file and registers new transactions along with their `slo` and range values with `ttd` and the `midaemon`. The re-read will not change the `slo` or range values for any transactions that were in the `ttd.conf` file prior to the re-read.

7. If you need to change the `slo` or range values of existing transactions in the `ttd.conf` file, do the following:
 - a. Stop all ARMed applications.
 - b. Stop the **oacore** collector using `ovpa stop`.
 - c. Stop all usage of Glance.
 - d. Stop `ttd` using `ttd -k`.

Once you have made your changes:

- a. Restart **oacore** using `ovpa start`.
- b. Start your ARMed applications.

Defining Service Level Objectives

Your first step in implementing transaction tracking is to determine the key transactions that are required to meet customer expectations and what level of transaction responsiveness is required. The level of responsiveness that is required becomes your service level objective (SLO). You define the service level objective in the configuration file, `ttd.conf`.

Defining service level objectives can be as simple as reviewing your Information Technology department's service level agreement (SLA) to see what transactions you need to monitor to meet your SLA. If you don't have an SLA, you may want to implement one. However, creating an SLA is not required in order to track transactions.

Modifying the Parm File

If necessary, modify the Performance Collection Component `parm` file to add transactions to the list of items to be logged for use with Performance Manager and Performance Collection Component. Include the `transaction` option in the `parm` file's `log` parameter as shown in the following example:

```
log global application process transaction device=disk
```

The default for the `log transaction` parameter is `no resource` and `no correlator`. To turn on resource data collection or correlator data collection, specify `log transaction=resource` or `log transaction=correlator`. Both can be logged by specifying `log transaction=resource, correlator`.

Before you can collect transaction data for use with Performance Collection Component and Performance Manager, the updated `parm` file must be activated as described below:

Performance Collection Component status	Command to activate transaction tracking
Running	<code>ovpa restart</code>
Not running	<code>ovpa start</code>

Collecting Transaction Data

Start up your application. The Transaction Tracking daemon, `ttd`, and the Measurement Interface daemon, `midaemon`, collect and synchronize the transaction data for your application as it runs. The data is stored in the `midaemon`'s shared memory segment where it can be used by Performance Collection Component or GlancePlus. See ["Monitoring Performance Data " on page 336](#) for information on using each of these tools to view transaction data for your application.

Error Handling

Due to performance considerations, not all problematic ARM or Transaction Tracker API calls return errors in real time. Some examples of when errors are not returned as expected are:

- calling `arm_start` with a bad `id` parameter such as an uninitialized variable
- calling `arm_stop` without a previously successful `arm_start` call

Performance Collection Component — To debug these situations when instrumenting applications with ARM calls, run the application long enough to generate and collect a sufficient amount of transaction data. Collect this data with Performance Collection Component, then use the `extract` command's `export` option to export transaction data. Examine the data to see if all transactions were logged as expected. Also, check the `/var/opt/perf/status.ttd` file for possible errors.

GlancePlus — To debug these situations when instrumenting applications with ARM calls, run the application long enough to generate a sufficient amount of transaction data, then use GlancePlus to see if all transactions appear as expected.

Limits on Unique Transactions

Depending on your particular system resources and kernel configuration, a limit may exist on the number of unique transactions allowed in your application. This limit is normally several thousand unique `arm_getid` calls.

The number of unique transactions may exceed the limit when the shared memory segment used by `midaemon` is full. If this happens, an overflow message appears in GlancePlus. Although no message appears in Performance Collection Component, data for subsequent new transactions won't be logged. Data for subsequent new transactions won't be visible in GlancePlus. Transactions that have already been registered will continue to be logged and reported. The `GBL_TT_OVERFLOW_COUNT` metric in GlancePlus reports the number of new transactions that could not be measured.

This situation can be remedied by stopping and restarting the `midaemon` process using the `-smdvss` option to specify a larger shared memory segment size. The current shared memory segment size can be checked using the `midaemon -sizes` command. For more information on optimizing the `midaemon` for your system, see the `midaemon` man page.

Customizing the Configuration File (optional)

After viewing the transaction data from your application, you may want to customize the transaction configuration file, `/var/opt/perf/ttd.conf`, to modify the way transaction data for the application is collected. This is optional because the default configuration file, `ttd.conf`, will work with all transactions defined in the application. If you do decide to customize the `ttd.conf` file, complete this task on the same systems where you run your application. You must be logged on as `root` to modify `ttd.conf`.

See ["How Transaction Tracking Works" on page 316](#) for information on the configuration file keywords – `tran`, `range`, and `slo`. Some examples of how each keyword is used are shown below:

```
tran=Example:   tran=answerid
                tran=answerid*
                tran=*

range=Example:  range=2.5,4.2,5.0,10.009

slo=Example:    slo=4.2
```

Customize your configuration file to include all of your transactions and each associated attribute. Note that the use of the `range` or `slo` keyword must be preceded by the `tran` keyword. An example of a `ttd.conf` file is shown below.

```
tran=*

tran=my_first_transaction slo=5.5

[answerid]
tran=answerid1 range=2.5, 4.2, 5.0, 10.009 slo=4.2

[orderid]
tran=orderid1 range=1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0
```

If you need to make additions to the `ttd.conf` file:

- Stop all ARMed applications.
- Execute the `ttd -hup -mi` command as root.

The above actions re-read the `ttd.conf` file and registers new transactions along with their `slo` and `range` values with `ttd` and the `midaemon`. The re-read will not change the `slo` or `range` value for any transactions that were in the `ttd.conf` file prior to the re-read,

If you need to change the `slo` or `range` values of existing transactions in the `ttd.conf` file, do the following:

1. Stop all ARMed applications.
2. Stop the **oacore** collector using `ovpa stop`.
3. Stop all usage of Glance.
4. Stop `ttd` using `ttd -k`.

Once you have made your changes:

1. Restart **oacore** using `ovpa start`.
2. Start your ARMed applications.

Monitoring Performance Data

You can use the following resource and performance management products to monitor transaction data – Performance Collection Component, Performance Manager, and GlancePlus.

... with Performance Collection Component

By collecting and logging data for long periods of time, Performance Collection Component gives you the ability to analyze your system's performance over time and to perform detailed trend analysis. Data from Performance Collection Component can be viewed with Performance Manager Agent or exported for use with a variety of other performance monitoring, accounting, modeling, and planning tools.

With Performance Collection Component's `extract` program, data can be exported for use with spreadsheets and analysis programs. Data can also be extracted for archiving and analysis.

Performance Collection Component and `ttd` must be running in order to monitor transaction data in Performance Collection Component. Starting Performance Collection Component using the `ovpa` script ensures that the `ttd` and `midaemon` processes that are required to view transaction data in GlancePlus are started in the right order.

... with Performance Manager

Performance Manager imports Performance Collection Component data and gives you the ability to translate that data into a customized graphical or numerical format. Using Performance Manager, you can perform analysis of historical trends of transaction data and you can perform more accurate forecasting.

You can select **TRANSACTION** from the Class List window for a data source in Performance Manager, then graph transaction metrics for various transactions. For more information, see Performance Manager online help, which is accessible from the Performance Manager Help menu. If you don't see the transactions you expect in Performance Manager, close the current data source and then reconnect to it.

... with GlancePlus

Monitoring systems with GlancePlus helps you identify resource bottlenecks and provides immediate performance information about the computer system. GlancePlus has a Transaction Tracking window that displays information about all transactions that you have defined and a Transaction Graph window that displays specific information about a single transaction. For example, you can see how each

transaction is performing against the SLO that you have defined. For more information about how to use GlancePlus, see the online help that is accessible from the Help menu.

Alarms

You can alarm on transaction data with the following resource and performance management products — Performance Collection Component, Performance Manager, and GlancePlus.

... with Performance Collection Component

In order to generate alarms with Performance Collection Component, you must define alarm conditions in its alarm definitions file, `aalarmdef`. You can set up Performance Collection Component to notify you of an alarm condition in various ways, such as sending an email message or initiating a call to your pager.

To pass a syntax check for the `aalarmdef` file, you must have data logged for that application name and transaction name in the log files, or have the names registered in the `ttd.conf` file.

There is a limitation when you define an alarm condition on a transaction that has a dash (–) in its name. To get around this limitation, use the `ALIAS` command in the `aalarmdef` file to redefine the transaction name.

... with GlancePlus

You can configure the Adviser Syntax to alarm on transaction performance. For example, when an alarm condition is met, you can instruct GlancePlus to display information to `stdout`, execute a UNIX command (such as `mailx`), or turn the Alarm button on the main GlancePlus window yellow or red. For more information about alarms in GlancePlus, choose **On This Window** from the Help menu in the Edit Adviser Syntax window.

Chapter 24: Transaction Tracking Messages

The error codes listed in Table 2 are returned and can be used by the application developer when instrumenting an application with Application Response Measurement (ARM) or Transaction Tracker API calls:

Table 2: Error codes

Error Code	Errno Value	Meaning
-1	EINVAL	Invalid arguments
-2	EPIPE	ttd (registration daemon) not running
-3	ESRCH	Transaction name not found in the ttd.conf file
-4	EOPNOTSUPP	Operating system version not supported

When an application instrumented with ARM or Transaction Tracker API calls is running, return codes from any errors that occur will probably be from the Transaction Tracking daemon, ttd. The Measurement Interface daemon, midaemon, does not produce any error return codes.

If a midaemon error occurs, see the `/var/opt/perf/status.mi` file for more information.

Chapter 25: Transaction Metrics

The ARM agent provided as a shared component of both the GlancePlus and Performance Collection Component, produces many different transaction metrics. To see a complete list of the metrics and their descriptions:

- For installed GlancePlus metrics, use the GlancePlus online help or see the *GlancePlus for HP-UX Dictionary of Performance Metrics* located:

On UNIX/Linux under `/<InstallDir>/paperdocs/gp/C/` as `gp-metrics.txt`.

`InstallDir` is the directory in which Performance Collection Component is installed.

- For installed Performance Collection Component metrics for specific platforms, see the platform's *Operations Agent Dictionary of Operating System Performance Metrics* files located:

On UNIX/Linux under `/<InstallDir>/paperdocs/ovpa/C/` as `met<platform>.txt`.

On Windows under `%ovinstalldir%paperdocs\ovpa\C` as `met<platform>.txt`.

Chapter 26: Transaction Tracking Examples

This chapter contains a pseudocode example of how an application might be instrumented with ARM API calls, so that the transactions defined in the application can be monitored with Performance Collection Component or GlancePlus. This pseudocode example corresponds with the real time order processing scenario described in ["What is Transaction Tracking?" on page 310](#)

Several example transaction configuration files are included in this chapter, including one that corresponds with the real time order processing scenario.

Pseudocode for Real Time Order Processing

This pseudocode example includes the ARM API calls used to define transactions for the real time order processing scenario described in ["What is Transaction Tracking?" on page 310](#). This routine would be processed *each time* an operator answered the phone to handle a customer order. The lines containing the ARM API calls are highlighted with bold text in this example.

```
routine answer calls()
{
*****
* Register the transactions if first time in      *
*****

    if (transactions not registered)
    {
        appl_id = arm_init("Order Processing Application","*", 0,0,0)
        answer_phone_id = arm_getid(appl_id,"answer_phone","1st tran",0,0,0)
        if (answer_phone_id < 0)
            REGISTER OF ANSWER_PHONE FAILED - TAKE APPROPRIATE ACTION
        order_id = arm_getid(appl_id,"order","2nd tran",0,0,0)
        if (order_id < 0)
            REGISTER OF ORDER FAILED - TAKE APPROPRIATE ACTION
        check_id = arm_getid(appl_id,"check_db","3rd tran",0,0,0)
```

```

    if (check_id < 0)
        REGISTER OF CHECK DB FAILED - TAKE APPROPRIATE ACTION
    update_id = arm_getid(appl_id,"update","4th tran",0,0,0)
    if (update_id < 0)
        REGISTER OF UPDATE FAILED - TAKE APPROPRIATE ACTION
} if transactions not registered
*****
* Main transaction processing loop
*****
    while (answering calls)
    {
        if (answer_phone_handle = arm_start(answer_phone_id,0,0,0) < -1)
            TRANSACTION START FOR ANSWER_PHONE NOT REGISTERED
*****
* At this point the answer_phone transaction has      *
* started.  If the customer does not want to order, *
* end the call; otherwise, proceed with order.      *
*****
        if (don't want to order)
            arm_stop(answer_phone_handle,ARM_FAILED,0,0,0)
            GOOD-BYE - call complete
        else
        {
*****
* They want to place an order - start an order now *
*****
            if (order_handle = arm_start(order_id,0,0,0) < -1)
                TRANSACTION START FOR ORDER FAILED
            take order information: name, address, item, etc.
*****
* Order is complete - end the order transaction    *

```

```

*****
    if (arm_stop(order_handle,ARM_GOOD,0,0,0) < -1)
        TRANSACTION END FOR ORDER FAILED
*****
* order taken - query database for availability      *
*****
    if (query_handle = arm_start(query_id,0,0,0) < -1)
        TRANSACTION QUERY DB FOR ORDER NOT REGISTERED
    query the database for availability
*****
* database query complete - end query transaction  *
*****
    if (arm_stop(query_handle,ARM_GOOD,0,0,0) < -1)
        TRANSACTION END FOR QUERY DB FAILED
*****
* If the item is in stock, process order, and      *
* update inventory.                               *
*****
    if (item in stock)
        if (update_handle = arm_start(update_id,0,0,0) < -1)
            TRANSACTION START FOR UPDATE NOT REGISTERED
        update stock
*****
* update complete - end the update transaction    *
*****
    if (arm_stop(update_handle,ARM_GOOD,0,0,0) < -1)
        TRANSACTION END FOR ORDER FAILED
*****
* Order complete - end the call transaction      *
*****
    if (arm_stop(answer_phone_handle,ARM_GOOD,0,0,0) < -1)

```

```
                TRANSACTION END FOR ANSWER_PHONE FAILED
    } placing the order
    GOOD-BYE - call complete
    sleep("waiting for next phone call...zzz...")
    }         while answering calls
    arm_end(appl_id, 0,0,0)
    }         routine answer calls
```

Configuration File Examples

This section contains some examples of the transaction configuration file, `/var/opt/perf/ttd.conf`. For more information on the `ttd.conf` file and the configuration file keywords, see ["How Transaction Tracking Works" on page 316](#)

Example 1 (for Order Processing Pseudocode Example)

```
# The "*" entry below is used as the default if none of the
# entries match a registered transaction name.
```

```
tran=* range=0.5,1,1.5,2,3,4,5,6,7 slo=1
tran=answer_phone* range=0.5,1,1.5,2,3,4,5,6,7 slo=5
tran=order* range=0.5,1,1.5,2,3,4,5,6,7 slo=5
tran=query_db* range=0.5,1,1.5,2,3,4,5,6,7 slo=5
```

Example 2

```
# The "*" entry below is used as the default if none of the
# entries match a registered transaction name.
```

```
tran=* range=1,2,3,4,5,6,7,8 slo=5
```

```
# The entry below is for the only transaction being
# tracked in this application. The "*" has been inserted
# at the end of the tran name to catch any possible numbered
# transactions. For example "First_Transaction1",
# "First_Transaction2", etc.

tran=First_Transaction* range=1,2.2,3.3,4.0,5.5,10 slo=5.5
```

Example 3

```
# The "*" entry below is used as the default if none of the
# entries match a registered transaction name.
```

```
tran=*
tran=Transaction_One range=1,10,20,30,40,50,60 slo=30
```

Example 4

```
tran=FactoryStor* range=0.05, 0.10, 0.15 slo=3
# The entries below shows the use of an application name.
# Transactions are grouped under the application name. This
# example also shows the use of less than 10 ranges and
# optional use of "slo."
```

```
[Inventory]
tran=In_Stock range=0.001, 0.004, 0.008
tran=Out_Stock range=0.001, 0.005
tran>Returns range=0.1, 0.3, 0.7
```

```
[Pers]
tran=Acctg range=0.5, 0.10, slo=5
```



```
tran=Time_Cards range=0.010, 0.020
```

Chapter 27: Advanced Features

This chapter describes how Performance Collection Component uses the following ARM 2.0 API features:

- data types
- user-defined metrics
- **oacore** instrumentation

How Data Types are Used

The table below describes how data types are used in Performance Collection Component. It is a supplement to “Data Type Definitions” in the “Advanced Topics” section of the [Application Response Measurement 2.0 API Guide](#).

Table 3: Data type usage in Performance Collection Component

ARM_Counter32	Data is logged as a 32-bit integer.
ARM_Counter64	Data is logged as a 32-bit integer with type casting.
ARM_CntrDivr32	Makes the calculation and logs the result as a 32-bit integer.
ARM_Gauge32	Data is logged as a 32-bit integer.
ARM_Gauge64	Data is logged as a 32-bit integer with type casting.
ARM_GaugeDivr32	Makes the calculation and logs the result as a 32-bit integer.
ARM_NumericID32	Data is logged as a 32-bit integer.
ARM_NumericID64	Data is logged as a 32 bit integer with type casting.
ARM_String8	Ignored.
ARM_String32	Ignored.

Performance Collection Component does not log string data. Because Performance Collection Component logs data every five minutes, and what is logged is the summary of the activity for that interval, it cannot summarize the strings provided by the application.

Performance Collection Component logs the Minimum, Maximum, and Average for the first six usable user-defined metrics. If your ARM-instrumented application passes a Counter32, a String8, a NumericID 32, a Gauge32, a Gauge64, a Counter64, a NumericID64, a String32, and a GaugeDivr32, Performance Collection Component logs the Min, Max, and Average over the five-minute interval for the Counter32, NumericID32, Gauge32, Gauge64, NumericID32 and NumericID64 as 32-bit integers. The String8 and String32 are ignored because strings cannot be summarized in the Performance Collection Component. The GaugeDivr32 is also ignored because only the first six usable user-defined metrics are logged. (For more examples, see the next section, [User-Defined Metrics](#))

User-Defined Metrics

This section is a supplement to “Application-Defined Metrics” under “Advanced Topics” in the [Application Response Measurement 2.0 API Guide](#). It contains some examples about how Performance Collection Component handles user-defined metrics (referred to as application-defined metrics in ARM). The examples in Table 4 show what is logged if your program passes the following data types.

Table 4: Examples of What is Logged with Specific Program Data Types

...what your program passes in	...what is logged
EXAMPLE 1 String8 Counter32 Gauge32 CntrDivr32	Counter32 Gauge32 CntrDivr32
EXAMPLE 2 String32 NumericID32 NumericID64	NumericID32 NumericID64
EXAMPLE 3 NumericID32 String8 NumericID64 Gauge32 String32	NumericID32 NumericID64 Gauge32

Table 4: Examples of What is Logged with Specific Program Data Types, continued

...what your program passes in	...what is logged
Gauge64	Gauge64
EXAMPLE 4 String8 String32	(nothing)
EXAMPLE 5 Counter32 Counter64 CntrDivr32 Gauge32 Gauge64 NumericID32 NumericID64	Counter32 Counter64 CntrDivr32 Gauge32 Gauge64 NumericID32

Because Performance Collection Component cannot summarize strings, no strings are logged.

In example 1, only the counter, gauge, and counter divisor are logged.

In example 2, only the numerics are logged.

In example 3, only the numerics and gauges are logged.

In example 4, nothing is logged.

In example 5, because only the first six user-defined metrics are logged, NumericID64 is not logged.

Chapter 28: Transaction Libraries

This appendix discusses:

- Application Response Measurement library (libarm)
- C compiler option examples by platform
- Application Response Measurement NOP library (libarmNOP)
- Using Java wrappers

ARM Library (libarm)

With Performance Collection Component and GlancePlus, the environment is set up to make it easy to compile and use the ARM facility. The libraries needed for development are located in `/opt/perf/lib/`. See the next section in this appendix for specific information about compiling.

The library files listed in Table 5 exist on an HP-UX 11.11 and beyond Performance Collection Component and GlancePlus installation:

Table 5: HP-UX 11.11 and Beyond Performance Collection Component and GlancePlus Library Files

<code>/opt/perf/lib/</code>	<code>libarm.0</code>	HP-UX 10.X compatible shared library for ARM (not thread safe). If you execute a program on HP-UX 11 that was linked on 10.20 with <code>-larm</code> , the 11.0 loader will automatically reference this library.
	<code>libarm.1</code>	HP-UX 11 compatible shared library (thread safe). This will be referenced by programs that were linked with <code>-larm</code> on HP-UX releases. If a program linked on 10.20 references this library, (for example, if it was not linked with <code>-L /opt/perf/lib</code> , it may abort with an error such as <code>"/usr/lib/dld.sl: Unresolved symbol: _thread_once (code) from libtt.sl"</code> .
	<code>libarm.sl</code>	A symbolic link to <code>libarm.1</code>
	<code>libarmNOP.sl</code>	"No-operation" shared library for ARM (the

		API calls succeed but do nothing; used for testing and on systems that do not have Performance Collection Component installed.
/opt/perf/examples/arm	libarmjava.sl	32-bit shared library for ARM.
/opt/perf/examples/arm/arm64	libarmjava.sl	64-bit shared library for ARM.
/opt/perf/lib/pa20_64/	Note that these files will be referenced automatically by programs compiled on HP-UX 11 with the +DD64 compiler option.	
	libarm.sl	64-bit shared library for ARM.
	libarmNOP.sl	64-bit "no-operation" shared library for ARM (the API calls succeed but do nothing; used for testing and on systems that do not have Performance Collection Component installed).

The additional library files listed in Table 6 exist on an IA64 HP-UX installation:

Table 6: HP-UX IA64 Library Files

/opt/perf/lib/hpux32/	libarm.so.1	IA64/32-bit shared library for ARM.
/opt/perf/lib/hpux64/	libarm.so.1	IA64/64-bit shared library for ARM.
/opt/perf/examples/arm	libarmjava.so	32-bit shared library for ARM.
/opt/perf/examples/arm/arm64	libarmjava.so	64-bit shared library for ARM.

Because the ARM library makes calls to HP-UX that may change from one version of the operating system to the next, programs should link with the shared library version, using `-larm`. Compiling an application that has been instrumented with ARM API calls and linking with the archived version of the ARM library (`-Wl, -a archive`) is not supported. (For additional information, see ["Transaction Tracking Daemon \(ttt\)" on page 319](#) in Chapter 2.

The library files that exist on an AIX operating system with Performance Collection Component and GlancePlus installation are as follows.

Table 7: AIX Library Files

/usr/lpp/perf/lib/	libarm.a	32-bit shared ARM library (thread safe). This library is referenced by programs linked with <code>-larm</code> .
/usr/lpp/perf/lib	libarmNOP.a	32-bit shared library for ARM. This library is used for testing on systems that do not

		have Performance Agent/Performance Collection Component installed.
/usr/lpp/perf/lib64/	libarm.a	64-bit shared ARM library (thread safe). This library is referenced by programs linked with -larm.
/usr/lpp/perf/lib64	libarmNOP.a	64-bit shared library for ARM. This library is used for testing on systems that do not have Performance Agent/Performance Collection Component installed.
/usr/lpp/perf/examples/arm	libarmjava.a	32-bit shared library for ARM
/usr/lpp/perf/examples/arm/arm64	libarmjava.a	64-bit shared library for ARM.
/usr/lpp/perf/lib/	libarmns.a	32-bit archived ARM library. Functionality wise this is same as 32 bit libarm.a.
/usr/lpp/perf/lib64/	libarmns.a	64-bit archived ARM library. Functionality wise this is same as 64 bit libarm.a.

The library files that exist on a Solaris operating system with Performance Collection Component and GlancePlus installation are as follows.

Table 8: Solaris Library Files for 32-bit programs

/opt/perf/lib/	libarm.so	32-bit shared ARM library (thread safe). This library is referenced by programs linked with -larm.
	libarmNOP.so	32-bit shared library for ARM. This library is used for testing on systems that do not have Performance Collection Component installed.

Table 9: Solaris Library Files for Sparc 64-bit programs

/opt/perf/lib/sparc_64/	libarm.so	64-bit shared ARM library (thread safe). This library is referenced by programs linked with -larm.
	libarmNOP.so	64-bit shared library for ARM This library is used for testing on systems that do not have Performance agent/Performance Collection Component installed.
/opt/perf/examples/arm	libarmjava.so	32-bit shared library for ARM.
/opt/perf/examples/arm/arm64	libarmjava.so	64-bit shared library for ARM.

Table 10: Solaris Library Files for x86 64-bit programs

/opt/perf/lib/x86_64/	libarm.so	64-bit shared ARM library (thread safe). This library is referenced by programs linked with -larm.
	libarmNOP.so	64-bit shared library for ARM This library is used for testing on systems that do not have Performance agent installed.
/opt/perf/examples/arm/arm64	libarmjava.so	32-bit shared library for ARM.
/opt/perf/examples/arm/arm64	libarmjava.so	64-bit shared library for ARM.

Note: You must compile 64-bit programs using `-xarch=generic64` command-line parameter along with the other parameters provided for 32-bit programs.

The library files that exist on a Linux operating system with Performance Collection Component and GlancePlus installation are as follows.

Table 11: Linux Library Files

/opt/perf/lib/	libarm.so	32-bit shared ARM library (thread safe). This library is referenced by programs linked with -larm.
	libarmNOP.so	32-bit shared library for ARM. This library is used for testing on systems that do not have Performance Collection Component installed.
/opt/perf/lib/	libarm.so	64-bit shared ARM library (thread safe). This library is referenced by programs linked with -larm.
	libarmNOP.so	64-bit shared library for ARM. This library is used for testing on systems that do not have Performance Collection Component installed.
/opt/perf/examples/arm	libarmjava.so	32-bit shared library for ARM.
/opt/perf/examples/arm/arm64	libarmjava.so	64-bit shared library for ARM.

Note: For Linux 2.6 IA 64 bit 32 bit libarm.so and libarmjava.so are not implemented.

C Compiler Option Examples by Platform

The `arm.h` include file is located in `/opt/perf/include/`. For convenience, this file is accessible via a symbolic link from `/usr/include/` as well. This means that you do not need to use “`-I/opt/perf/include/`” (although you may). Likewise, `libarm` resides in `/opt/perf/lib/` but is linked from `/usr/lib/`. You should always use “`-L/opt/perf/lib/`” when building ARMed applications.

- For Linux:

The following example shows a compile command for a C program using the ARM API.

```
cc myfile.c -o myfile -I /opt/perf/include -L -Xlinker -rpath -Xlinker /opt/perf/lib
```

- For 64-bit programs on Linux:

```
cc -m64 myfile.c -o myfile -I /opt/perf/include -L -Xlinker -rpath -Xlinker /opt/perf/lib64
```

- For HP-UX:

For HP-UX releases 11.2x on IA64 platforms, change the `-L` parameter from `-L/opt/perf/lib` to `-L/opt/perf/lib/hpux32` for 32-bit IA ARMed program compiles, and to `-L/opt/perf/lib/hpux64` for 64-bit IA program compiles using ARM.

The following example shows a compile command for a C program using the ARM API.

```
cc myfile.c -o myfile -I /opt/perf/include -L /opt/perf/lib -larm
```

- For Sun Solaris:

The following example works for Performance Collection Component and GlancePlus on Sun Solaris:

```
cc myfile.c -o myfile -I /opt/perf/include -L /opt/perf/lib -larm -lnsl
```

- For 64-bit Sparc programs on Sun Solaris:

The following example works for Performance Collection Component and 64-bit programs on Sun Solaris:

```
cc -xarch=generic64 myfile.c -o myfile -I /opt/perf/include -L /opt/perf/lib/sparc_64 -larm -lnsl
```

- For 64-bit x86 programs on Sun Solaris:

The following example works for Performance agent and 64-bit programs on Sun Solaris:

```
cc -xarch=generic64 myfile.c -o myfile -I /opt/perf/include -L  
/opt/perf/lib/x86_64 -larm -lnsl
```

- For IBM AIX:

The file placement on IBM AIX differs from the other platforms (`/usr/lpp/perf/` is used instead of `/opt/perf/`), therefore the example for IBM AIX is different from the examples of other platforms:

```
cc myfile.c -o myfile -I /usr/lpp/perf/include -L /usr/lpp/perf/lib -larm
```

- For 64-bit programs on IBM AIX:

The following example works for Performance agent and 64-bit programs on IBM AIX:

```
cc -q64 myfile.c -o myfile -I /usr/lpp/perf/include -L /usr/lpp/perf/lib64 -larm
```

Note: For C++ compilers, the `-D_PROTOTYPES` flag may need to be added to the compile command in order to reference the proper declarations in the `arm.h` file.

ARM NOP Library

The “no-operation” library (named `libarmNOP.*` where `*` is `sl`, `so`, or `a`, depending on the OS platform) is shipped with Performance Collection Component and Glance. This shared library does nothing except return valid status for every ARM API call. This allows an application that has been instrumented with ARM to run on a system where Performance Collection Component or GlancePlus is not installed.

To run your ARM-instrumented application on a system where Performance Collection Component or GlancePlus is not installed, copy the NOP library and name it `libarm.sl` (`libarm.so`, or `libarm.a` depending on the platform) in the appropriate directory (typically, `/<InstallDir>/lib/`). When Performance Collection Component or GlancePlus is installed, it will overwrite this NOP library with the correct functional library (which is not removed as the other files are). This ensures that instrumented programs will not abort when Performance Collection Component or GlancePlus is removed from the system.

Using the Java Wrappers

The Java Native Interface (JNI) wrappers are functions created for your convenience to allow the Java applications to call the ARM2.0 API. These wrappers (`armapi.jar`) are included with the ARM sample programs located in the `/<InstallDir>/examples/arm/` directory. `InstallDir` is the directory in which Performance Collection Component is installed.

Examples

Examples of the Java wrappers are located in the `<InstallDir>/examples/arm/` directory. This location also contains a README file, which explains the function of each wrapper.

Setting Up an Application (arm_init)

To set up a new application, make a new instance of `ARMApplication` and pass the name and the description for this API. Each application needs to be identified by a unique name. The `ARMApplication` class uses the C – function `arm_init`.

Syntax:

```
ARMApplication myApplication =new ARMApplication("name","description")
```

Setting Up a Transaction (arm_getid)

To set up a new transaction, you can choose whether or not you want to use user-defined metrics (UDMs). The Java wrappers use the C – function `arm_getid`.

Setting Up a Transaction With UDMs

If you want to use UDMs, you must first define a new `ARMTranDescription`. `ARMTranDescription` builds the Data Buffer for `arm_getid`. (See also the `jprimeudm.java` example.)

Syntax:

```
ARMTranDescription myDescription =  
new ARMTranDescription("transactionName","details");
```

If you don't want to use details, you can use another constructor:

Syntax:

```
ARMTranDescription myDescription =  
new ARMTranDescription("transactionName");
```

Adding the Metrics

Metric 1-6:

Syntax:

```
myDescription.addMetric(metricPosition, metricType, metricDescription);
```

Parameters:

metricPosition: 1-6

metricType: ARMConstants.ARM_Counter32

ARMConstants.ARM_Counter64 ARMConstants.ARM_CntrDivr32

ARMConstants.ARM_Gauge32 ARMConstants.ARM_Gauge64

ARMConstants.ARM_GaugeDivr32 ARMConstants.ARM_NumericID32

ARMConstants.ARM_NumericID64 ARMConstants.ARM_String8

Metric 7:

Syntax:

```
myDescription.addStringMetric("description");
```

Then you can create the Transaction:

Syntax:

```
myApplication.createTransaction(myDescription);
```

Setting the Metric Data

Metric 1-6:

Syntax:

```
myTransaction.setMetricData(metricPosition, metric);
```

Examples for "Metric"

```
ARMGauge32Metric metric = new ARMGauge32Metric(start);
```

```
ARMCounter32Metric metric = new ARMCounter32Metric(start);
```

```
ARMCntrDivr32Metric metric = new ARMCntrDivr32Metric(start, 1000);
```

Metric 7:

Syntax:

```
myTransaction.setStringMetricData(text);
```

Setting Up a Transaction Without UDMs

When you set up a transaction without UDMs, you can immediately create the new transaction. You can choose whether or not to specify details.

With Details

Syntax:

```
ARMTransaction myTransaction =  
myApplication.createTransaction("Transactionname", "details");
```

Without Details

Syntax:

```
ARMTransaction myTransaction =  
myApplication.createTransaction("Transactionname");
```

Setting Up a Transaction Instance

To set up a new transaction instance, make a new instance of `ARMTransactionInstance` with the method `createTransactionInstance()` of `ARMTransaction`.

Syntax:

```
ARMTransactionInstance myTranInstance =  
myTransaction.createTransactionInstance();
```

Starting a Transaction Instance (arm_start)

To start a transaction instance, you can choose whether or not to use correlators. The following methods call the C – function `arm_start` with the relevant parameters.

Starting the Transaction Instance Using Correlators

When you use correlators, you must distinguish between getting and delivering a correlator.

Requesting a Correlator

If your transaction instance wants to request a correlator, the call is as follows (see also the `jcorrelators.java` example).

Syntax:

```
int status = myTranInstance.startTranWithCorrelator();
```

Passing the Parent Correlator

If you already have a correlator from a previous transaction and you want to deliver it to your transaction, the syntax is as follows:

Syntax

```
int status = startTran(parent);
```

Parameter

`parent` is the delivered correlator. In the previous transaction, you can get the transaction instance correlator with the method `getCorrelator()`.

Requesting and Passing the Parent Correlator

If you already have a correlator from a previous transaction and you want to deliver it to your transaction and request a correlator, the syntax is as follows:

Syntax:

```
int status = myTranInstance.startTranWithCorrelator(parent);
```

Parameter:

`parent` is the delivered correlator. In the previous transaction, you can get the transaction instance correlator with the method `getCorrelator()`.

Retrieving the Correlator Information

You can retrieve the transaction instance correlator using the `getCorrelator()` method as follows:

Syntax:

```
ARMTranCorrelator parent = myTranInstance.getCorrelator();
```

Starting the Transaction Instance Without Using Correlators

When you do not use correlators, you can start your transaction instance as follows:

Syntax:

```
int status = myTranInstance.startTran();
```

`startTran` returns a unique handle to `status`, which is used for the update and stop.

Updating Transaction Instance Data

You can update the UDMs of your transaction instance any number of times between the start and stop. This part of the wrappers calls the C – function `arm_update` with the relevant parameters.

Updating Transaction Instance Data With UDMs

When you update the data of your transaction instance with UDMs, first, you must set the new data for the metric. For example,

```
metric.setData(value) for ARM_Counter32 ARM_Counter64, ARM_Gauge32, ARM_Gauge64, ARM_NumericID32, ARM_NumericID64
```

```
metric.setData(value,value) for ARM_CntrDivr32 and , ARM_GaugeDivr32
```

```
metric.setData(string) for ARM_String8 and ARM_String32
```

Then you can set the metric data to new (like the examples in the ["Setting the Metric Data" on page 356](#) section) and call the update:

Syntax:

```
myTranInstance.updateTranInstance();
```

Updating Transaction Instance Data Without UDMs

When you update the data of your transaction instance without UDMs, you just call the update. This sends a “heartbeat” indicating that the transaction instance is still running.

Syntax:

```
myTranInstance.updateTranInstance();
```

Providing a Larger Opaque Application Private Buffer

If you want to use the second buffer format, you must pass the byte array to the update method. (See the [Application Response Measurement 2.0 API Guide](#).)

Syntax:

```
myTranInstance.updateTranInstance(byteArray);
```

Stopping the Transaction Instance (arm_stop)

To stop the transaction instance, you can choose whether or not to stop it with or without a metric update.

Stopping the Transaction Instance With a Metric Update

To stop the transaction instance with a metric update, call the method `stopTranInstanceWithMetricUpdate`.

Syntax:

```
myTranInstance.stopTranInstanceWithMetricUpdate  
transactionCompletionCode);
```


Parameter:

The transaction Completion Code can be:

ARMConstants. ARM_GOOD.	Use this value when the operation ran normally and as expected.	ARMConstants. ARM_GOOD.
ARMConstants.ARM_ ABORT.	Use this value when there is a fundamental failure in the system.	ARMConstants.ARM_ ABORT.
ARMConstants.ARM_ FAILED.	Use this value in applications where the transaction worked properly, but no result was generated.	ARMConstants.ARM_ FAILED.

These methods use the C – function `arm_stop` with the requested parameters.

Stopping the Transaction Instance Without a Metric Update

To stop the transaction instance without a metric update, you can use the method `stopTranInstance`.

Syntax:

```
myTranInstance.stopTranInstance(transactionCompletionCode);
```

Using Complete Transaction

The Java wrappers can use the `arm_complete_transaction` call. This call can be used to mark the end of a transaction that has lasted for a specified number of nanoseconds. This enables the real time integration of transaction response times measured outside of the ARM agent.

In addition to signaling the end of a transaction instance, additional information about the transaction (UDMs) can be provided in the optional data buffer.

(See also the `jcomplete.java` example.)

Using Complete Transaction With UDMs:

Syntax:

```
myTranInstance.completeTranWithUserData(status,responseTime;
```

Parameters:

status	<ul style="list-style-type: none">• <code>ARMConstants.ARM_GOOD</code> Use this value when the operation ran normally and as expected.• <code>ARMConstants.ARM_ABORT</code> Use this value when there was a fundamental failure in the system.• <code>ARMConstants.ARM_FAILED</code> Use this value in applications where the transaction worked properly, but no result was generated.
responseTime	This is the response time of the transaction in nanoseconds.

Using Complete Transaction Without UDMs:

Syntax:

```
myTranInstance.completeTran(status,responseTime);
```

Further Documentation

For further information about the Java classes, see the doc folder in the `<InstallDir>/examples/arm/` directory, which includes html-documentation for every Java class. Start with `index.htm`.

Part VI: Troubleshooting

This section describes the solutions or workarounds for the common problems encountered while working with the Operations Agent.

Operations Monitoring Component

- *Problem:* If OM manages a large number of nodes (more than 1024), you may experience communication problems between OM and managed nodes. You can also see this problem when the Operations Agent is installed on the Performance Manager server that communicates with a large number of managed nodes (more than 1024).

Solution:

To avoid this problem, go to the management server (if OM manages more than 1024 nodes) or the Performance Manager server (if the agent is installed on an Performance Manager server that communicates with more than 1024 nodes), and then perform the following configuration steps:

Log on as root or administrator.

Run the following command:

On 32-bit versions of Windows:

```
%ovinstalldir%bin\ovconfchg -ns xpl.net -set SocketPoll true
```

On 64-bit versions of Windows:

```
%ovinstalldir%bin\win64\ovconfchg -ns xpl.net -set SocketPoll true
```

On HP-UX/Linux/Solaris:

```
/opt/OV/bin/ovconfchg -ns xpl.net -set SocketPoll true
```

Restart the agent:

On 32-bit versions of Windows:

```
%ovinstalldir%bin\opcagt -stop
```

```
%ovinstalldir%bin\opcagt -start
```

On 64-bit versions of Windows:

```
%ovinstalldir%bin\win64\opcagt -stop
```

```
%ovinstalldir%bin\win64\opcagt -start
```

On HP-UX/Linux/ Solaris:

```
/opt/OV/bin/opcagt -stop
```

```
/opt/OV/bin/opcagt -start
```

- *Problem:* On the Windows Server 2008 node, the `opcmsga` process does not function, and the `ovccommand` shows the status of the `opcmsga` process as aborted.

Solution:

Set the `OPC_RPC_ONLY` variable to `TRUE` by running the following command:

```
ovconfchg -ns eaagt -set OPC_RPC_ONLY TRUE
```

- *Problem:* On Windows nodes, Perl scripts do not work from the policies.

Cause: Perl scripts available within the policies require the `PATH` configuration variable to include the directory where Perl (supplied with the Operations Agent) is available.

Solution:

- a. Run the following command to set the `PATH` configuration variable to the Perl directory:

```
ovconfchg -ns ctrl.env -set PATH "%ovinstalldir%nonOV\perl\bin"
```

- b. Restart the agent by running the following commands:

- i. `ovc -kill`

- ii. `ovc -start`

- *Problem:* Changes do not take effect after changing the variable values through the `ovconfchg` command.

Cause 1:

The variable requires the agent to be restarted.

Solution 1:

Restart the agent by running the following commands:

- a. `ovc -kill`

- b. `ovc -start`

Cause 2:

`ConfigFile` policies deployed on the node sets the variable to a certain value.

Solution 2:

If the deployed ConfigFile policies include commands to set the configuration variables to certain values, your changes through the `ovconfchg` command will not take effect. You must either remove the ConfigFile policies from the node, or modify the policies to include the commands that set the variables to the desired values.

Cause 3:

The profile or job file available on the node override your changes.

Solution 3: Open the profile or job file on the node and make sure they do not include conflicting settings for the variables.

- **Problem:** When a particular Message Interceptor (`msgi`) policy is deployed and a message that matches the condition in this policy is triggered, the `opcmsgi` process stops responding with high CPU utilization (it consumes 25 percent on 4-CPU's server, so 100 percent of a single CPU) and it does not process anymore messages.

Cause:

The `opcmsgi` process stops responding and does not process anymore messages because of the following pattern:

```
<*><@.variable><*>
```

Solution:

Replace the `<*><@.variable><*>` pattern with `<*><@.variable><S><*>`.

For example:

Original pattern:

```
^\\tM_Type: <@.SiSMonType><*>SiS: <[http://<@.SiSIP>:<@>].SiSURL><*>Monitor:
<@.SiSMonName><*>Group: <@.SiSGrpName><*>Status: <@.SiSMonStatus><*>URL:
<@.SiSMonURL><*>Match Content: <@.SiSMonMatch><*>Monitor Description: DOTCOM_
BACKEND: {<@.OMTargetNode>} {<@.OMMsgGrp>} {<*.OMApp1>} {<*.OMObj>} {<*.OMSvcID>}
{<@.SiSEMSCode>} {<*.OMCIs>} {<*.OMMsgText>}
```

Replaced Pattern:

```
^\\tM_Type: <@.SiSMonType><S><*>SiS: <
[http://<@.SiSIP>:<@>].SiSURL><S><*>Monitor: <@.SiSMonName><S><*>Group:
<@.SiSGrpName><S><*>Status: <@.SiSMonStatus><S><*>URL: <@.SiSMonURL><S><*>Match
Content: <@.SiSMonMatch><S><*>Monitor Description: DOTCOM_BACKEND:
{<@.OMTargetNode>} {<@.OMMsgGrp>} {<*.OMApp1>} {<*.OMObj>} {<*.OMSvcID>}
{<@.SiSEMSCode>} {<*.OMCIs>} {<*.OMMsgText>}
```

For example:

Original Pattern:

```
^\tm_Type: <@.SiSMonType><*>SiS: <[http://<@.SiSIP>:<@>].SiSURL><*>Monitor:
<@.SiSMonName><*>Group: <@.SiSGrpName><*>Status: <@.SiSMonStatus><*>URL:
<@.SiSMonURL><*>Match Content: <@.SiSMonMatch><*>Monitor Description: DOTCOM_
AUTONOMY:{<@.OMTargetNode>}{<@.OMMsgGrp>}{<*.OMApp1>}{<*.OMObj>}{<*.OMSvcID>}
{<@.SiSEMCode>}{<*.OMCIs>}{<*.OMMsgText>}
```

Replaced Pattern:

```
^\tm_Type: <@.SiSMonType><S><*>SiS: <
[http://<@.SiSIP>:<@>].SiSURL><S><*>Monitor: <@.SiSMonName><S><*>Group:
<@.SiSGrpName><S><*>Status: <@.SiSMonStatus><S><*>URL: <@.SiSMonURL><S><*>Match
Content: <@.SiSMonMatch><S><*>Monitor Description: DOTCOM_AUTONOMY:
{<@.OMTargetNode>}{<@.OMMsgGrp>}{<*.OMApp1>}{<*.OMObj>}{<*.OMSvcID>}
{<@.SiSEMCode>}{<*.OMCIs>}{<*.OMMsgText>}
```

- **Problem:** On Operations Agent, pattern matching reports wrong results.

Cause:

On Operations Agent, pattern matching may report wrong results if there are more than 148 OR (|) operators in the pattern.

Note: In pattern matching, OR (|) operator is a special character that separates two expressions by matching the string in both the expressions.

For example:

```
[ab|c]d
```

In the above pattern, the OR <|> operator matches the string **abd** and the string **cd**.

Solution:

Ensure that the number of OR (|) operators does not exceed 148.

- **Problem:** The opcmna process is automatically restarted after you run a schedule task policy with an embedded perl script on the node and the following message appears in the OM console:

```
(ctrl-208) Component 'opcmna' with pid 6976 exited with exit value '-
1073741819'. Restarting component.
```

Cause:

References of exit (0) in the embedded perl script cause opcmna to restart.

Solution:

Do not use `exit (0)` in the embedded perl script.

- *Problem:* On Red Hat Enterprise Linux system, `ovbbccb` cannot access the time zone file and writes the date information in UTC(GMT) format rather than actual time zone set for the system. This wrong time zone information is logged in the `System.txt` file.

Solution:

To establish the correct time zone, follow these steps:

- a. Run the following command to verify the location of the zoneinfo file:

```
ls /usr/share/zoneinfo/
```

- b. Run the following command to verify the location of the localtime file:

```
ls -l /etc/localtime
```

Note: Local time determines the current time zone if no "TZ" environment variable is set.

- c. Create the following directory:

```
mkdir -p /var/opt/OV/usr/share/lib
```

- d. Copy the files recursively from one folder to another:

```
cp -rp /usr/share/zoneinfo /var/opt/OV/usr/share/lib
```

- e. Create the directory named `etc` under `/var/opt/OV/` by running the following command:

```
mkdir /var/opt/OV/etc
```

- f. Copy the `localtime` file into the directory:

```
cp /etc/localtime /var/opt/OV/etc
```

- g. Restart the agent by running the following commands:

- i. `ovc -kill`

- ii. `ovc -start`

Note: `ovbbccb` uses the current time zone when it later records an entry into the `System.txt` file.

- *Problem:* `ovcodautl -ds scope -o process -flat -rawonly` is giving timeout error after upgrading Operations Agent from 11.14 to 12.01.

Cause: Reporting tools like Performance Manager will not be able to graph all the process data selected if the number of days are more.

Solution: Reduce the number of metrics or/and the duration and increase the timeout to a higher value.

Workaround: Run the following command to extract the data:

```
extract -xp -p
```

Performance Collection Component

- *Problem:* The following error appears in the `status.midaemon` file on the HP-UX 11.11 system:
`mi_shared - MI initialization failed (status 28)`

Cause: Large page size of the `midaemon` binary.

Solution: To resolve this, follow these steps:

a. Log on to the system as the root user.

b. Run the following command to stop the Operations Agent:

```
/opt/OV/bin/opcagt -stop
```

c. Run the following command to take a backup of `midaemon`:

```
cp /opt/perf/bin/midaemon /opt/perf/bin/midaemon.backup
```

d. Run the following command to reduce the page size to 4K for the `midaemon` binary:

```
chattr +pi 4K /opt/perf/bin/midaemon
```

e. Run the following command to start the Operations Agent:

```
/opt/OV/bin/opcagt -start
```

- *Problem:* After installing the Operations Agent, the following error message appears in the `System.txt` file if the tracing mechanism is enabled:

```
Scope data source initialization failed
```

Solution: Ignore this error.

- *Problem:* The following error message appears in the OM console:

```
CODA: GetDataMatrix returned 76='Method ScopeDataView::CreateViewEntity failed
```

Cause: This message appears if you use the `PROCESS` object with the `SCOPE` data source in Measurement Threshold policies where the source is set to Embedded Performance Component.

Solution: Use the service/process monitoring policy instead.

- *Problem: Data analysis products, such as the Performance Manager or the Operations Bridge Reporter, fail to retrieve data from agent's datastore and show the following error:*

Error occurred while retrieving data from the data source

Cause: The data access utility of the agent reads all the records of a data class against a single query from a client program. Queries are sent to the agent by data analysis clients like the Performance Manager. When a data class contains a high volume of records, the data access utility fails to process the query.

Solution: To avoid this issue, configure the data access utility to transfer data records to the client in multiple chunks. Follow these steps:

- a. Log on to the agent node with as root or administrator.
- b. Run the following command:

On Windows:

```
%ovinstalldir%bin\ovconfchg -ns coda -set DATAMATRIX_VERSION 1
```

On HP-UX/Linux/Solaris:

```
/opt/OV/bin/ovconfchg -ns coda -set DATAMATRIX_VERSION 1
```

On AIX:

```
/usr/lpp/OV/bin/ovconfchg -ns coda -set DATAMATRIX_VERSION 1
```

For each query, the agent's data access utility now breaks the data into chunks of five records, and then sends the data to the client program. Breaking the data into chunks enhances the performance of data transfer process.

You can control the number of records that the agent can send to the client with every chunk. The DATAMATRIX_ROWCOUNT variable (available under the coda namespace) enables you to control this number (the default value is five).

Decreasing the value of the DATAMATRIX_ROWCOUNT variable may marginally increase the data transfer rate when you have very large volumes of data into the datastore.

When the DATAMATRIX_ROWCOUNT variable is set to 0, the Operations Agent reverts to its default behavior of sending data records without chunking.

However, it is recommended that you do not change the default setting of the DATAMATRIX_ROWCOUNT variable.

- c. Restart the agent for the changes to take effect:

```
ovc -restart coda
```

- *Problem: hpsensor, perfd and oacore memory utilization is high on AIX 6.1 and 7.1.*

Solution: Install the patch provided by IBM to resolve this issue. For more information, see:

<https://www-01.ibm.com/support/docview.wss?uid=isg1IV61121>

<https://www-01.ibm.com/support/docview.wss?uid=isg1IV63105>

RTMA

- *Problem:* The following error appears in the `status.perfd` file on the HP-UX 11.11 system:

```
mi_shared - MI initialization failed (status 28)
```

Cause: Large page size of the `perfd` binary.

Solution: To resolve this, follow these steps:

- a. Log on to the system as the root user.
- b. Run the following command to stop the Operations Agent:

```
/opt/0V/bin/opcagt -stop
```

- c. Run the following command to take a backup of `perfd`:

```
cp /opt/perf/bin/perfd /opt/perf/bin/perfd.backup
```

- d. Run the following command to reduce the page size to 4K for the `perfd` binary:

```
chattr +pi 4K /opt/perf/bin/perfd
```

- e. Run the following command to start the Operations Agent:

```
/opt/0V/bin/opcagt -start
```

GlancePlus

Problem: GlancePlus does not show all LPAR instances hosted on an AIX frame.

Cause: The LPAR where you installed the Operations Agent cannot communicate with other LPARs hosted on the AIX frame.

Solution: Make sure that the LPAR where you installed the Operations Agent can communicate with all other LPARs hosted on the AIX frame.

Run the following command on the LPAR that hosts the Operations Agent to check its connectivity with other LPARs:

```
xmpeek -l<hostname>
```

In this instance, <hostname> is the host name of an LPAR.

hpsensor

- *Problem:* hpsensor reporting high resident memory usage.

Cause: Internal STL data structures allocated are not cleaned even after deleting. This issue is mainly seen on HPUX platform.

Solution: To resolve this issue, follow these steps:

- a. Log on to the system as an administrator.
- b. Go to the following directory:
 - *On Windows:* %OvDataDir%\hpcs\
 - *On Unix/Linux:* /var/opt/OV/hpcs/
- c. Open the **hpcs.conf** file and set the following configuration variable to **true** under the **hpcs.runtime** namespace:

```
MemMap=<true>
```

Other

- *Problem:* The fs_type metrics reports an autofs system as NFS (Network File System).

Cause: On a Linux system where Operations Agent 12.05 is running, an autofs system is reported as NFS if it is on a kernel 2.6 or lower versions.

Solution: To solve this issue, ensure that autofs systems are on kernel 3.0 or higher versions.

- *Problem :* The **oacore** process stops after logging the following error message in the System.txt file:

```
Database is in an inconsistent state.
```

Cause: If the managed class information is present without the corresponding database file, the **oacore** process stops after logging the error message in the System.txt file.

Note: Example of a System.txt message for Global class:

```
0: INF: Thu Aug 20 16:14:28 2015: oacore (21580/139960775325472): Collection intervals: Process = 60 Global = 300 DataFile Rollover% = 20.
0: ERR: Thu Aug 20 16:14:28 2015: oacore (21580/139960775325472): Database is in an inconsistent state. No database found for Class Scope::Global
0: INF: Thu Aug 20 16:14:28 2015: oacore (21580/139960775325472): (oacore-84) oacore. oacore Server stopped.
```

Solution: To resolve this issue, you must remove the datasource or specific classes from the datasource.

Use the `oadbutil.pl` tool to remove the datasource or specific classes from the datasource.

Syntax:

- `oadbutil.pl -d <datasource name>`
Removes the datasource along with its classes.
- `oadbutil.pl -d <datasource name> -c <class name>`
Removes meta data and records only for the specified class (Information about the datasource is retained).

For example:

```
oadbutil.pl -d Scope -c Global
```

In this instance :

<datasource name> is Scope

<class name> is Global

By default the data collector does not create a model to log default performance metric class and datasource (Scope) into the Metrics Datastore. After removing a default performance metric class or datasource, follow the steps to recreate the model:

- a. Run the following command to stop the **oacore** process:

```
ovc -stop oacore
```

- b. Run the following command recreate a class:

```
ovconfchg -ns oacore -set UPDATED_MODEL_AVAILABLE true
```

- c. Run the following command to start the **oacore** process:

```
ovc -start oacore
```

Note: If a custom class (submitted through Perl based scriptable APIs, or DSI) is removed, it

| is recreated automatically in subsequent submission.

Send documentation feedback

If you have comments about this document, you can [contact the documentation team](#) by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on User Guide (Operations Agent 12.05)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to docfeedback@hpe.com.

We appreciate your feedback!