HP OpenView Operations

Tracing Concepts and User's Guide

Software Version: A.08.10

HP-UX and Sun Solaris Management Servers

Manufacturing Part Number: None

September 2004

Legal Notices

Warranty.

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

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

Restricted Rights Legend.

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

Hewlett-Packard Company
United States of America

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

Copyright Notices.

Trademark Notices.

Adobe® is a trademark of Adobe Systems Incorporated.

Microsoft® is a U.S. registered trademark of Microsoft Corporation.

UNIX® is a registered trademark of the Open Group.

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

Contents

Contents

1 OpenView Tracing Fundamentals

Troubleshooting and Tracing

To help you investigate the cause of problems, OpenView provides problem tracing. Trace log files can help you pinpoint when and where problems occur (for example, if processes or programs abort, performance is greatly reduced, or unexpected results appear).

You can activate tracing for specific management server or agent processes by adding a statement to the `opcsvinfo` or `opcinfo` file. To simplify the interpretation of the trace logfile, you can activate tracing for specific functional areas by specifying one or more functional areas in the trace statement.

Installation

The OpenView tracing tools are installed during the HP OpenView product installation. To support dynamic trace configuration without additional steps the trace server is configured to start when the system starts. On Windows this is done by installing the trace server `OVTrace` as a Windows service called `OpenView Trace Service`. On UNIX this is done by adding `ovtrcd` to the startup script.

The files are placed in a location where trace-enabled applications can find the tracing library. For native applications, `OvXpl.dll` should be where the applications can find it (usually somewhere in the system search PATH). For Java applications, `xpl.jar` is added to the application's classpath.

The OpenView tracing tools are normally located under the HP OpenView directory structure. They can be found under the `/bin` directory corresponding to the HP OpenView product and platform.

UNIX Trace Server Installation

On UNIX systems, the trace server is started by adding the `S900VTrcSrv` start-up script to the following directory:

HP-UX `/sbin/rc3.d`

Solaris `/etc/rc3.d`

These files are links to the `OVTrcSrv` start-up script in the `init.d` directory that starts the `ovtrcd` executable located in the `/opt/OV/bin` directory.

The trace server can be started, stopped, or restarted if necessary using the following commands from the root account:

WARNING **If the trace server is stopped and restarted the tracing state for all applications is lost and can not be reloaded without restarting all applications to be traced.**

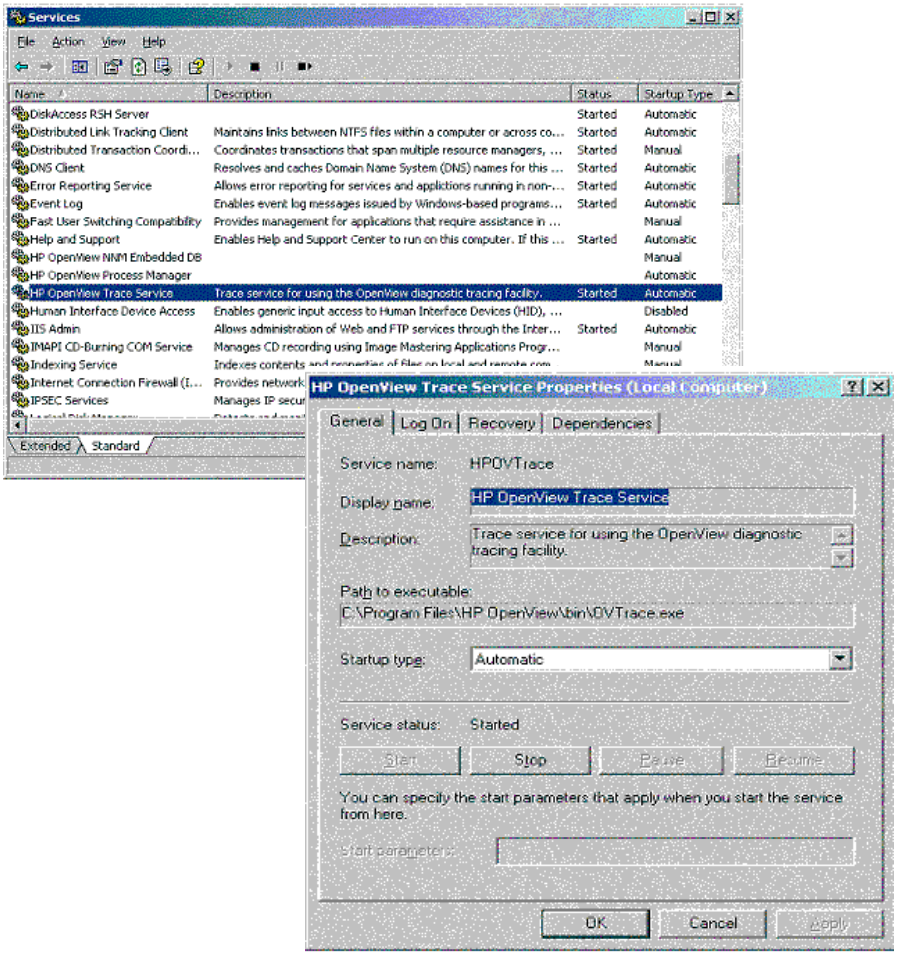
HP-UX	Start Trace Server: **`/sbin/rc3.d/S900OVTrcSrv start`** Stop Trace Server: **`/sbin/rc3.d/S900OVTrcSrv stop`** Restart Trace Server: **`/sbin/rc3.d/S900OVTrcSrv restart`**
Solaris	Start Trace Server: **`/etc/rc3.d/S90OVTrcSrv start`** Stop Trace Server: **`/etc/rc3.d/S90OVTrcSrv stop`** Restart Trace Server: **`/etc/rc3.d/S90OVTrcSrv restart`**

Windows Trace Server Installation

On Windows systems, the trace server is automatically installed and started by installing the `OVTrace.exe` as a Windows service called `HP OpenView Trace Service` (see Figure 1 and Figure 2).

The Trace Server can be started, stopped, or configured using the options available within the Services dialog.

Figure 1-1 HP OpenView Trace Service



Trace Information

The `ovtrcd` logs information into the following directories:

UNIX *<OVDataDir>*`/log/trc.log`

Windows *<ProgramFilesDir>*`\HP OpenView\data\log`

The `OVTraceCfg.dat` file is created under the `/root` directory by the local `ovtrcd`, after the trace-enabled applications have been started. This file contains the names of all applications and their associated component and category names.

How to Trace

The following section outline a general methodology that can be applied in most case where you need to enable tracing within an HP OpenView product. You need to use these steps as a general guideline and may need to make modifications to the procedure as necessary to fit the actual problem. The procedure assumes that a problem has been identified and that there is the need to capture application tracing information.

Overview of Tracing Procedures

1. Determine the applications (process / daemon) for which you need to capture trace information.
2. Determine if the applications identified are trace-enabled applications.
3. Determine if the trace-enable application requires any pre-configuration steps to enable OpenView Tracing.
4. Determine which components and categories within each application need to be traced.
5. Determine which attribute flags associated with each component/category combination need to be set.
6. Determine the most suitable tracing configurations for the problem being experienced. Tracing can be static or dynamic, configured locally or remotely, written directly to a file, or transmitted to a local or remote trace server.
7. Create the trace configuration file to match the tracing configuration selected.
8. Depending on the trace configuration selected, enable either static or dynamic tracing.
9. Execute the application specific commands necessary to duplicate the problem that initiated the requirement for tracing output. When the desired behavior has been duplicated the tracing can be stopped or disabled.

10. Depending upon the trace configuration selected, disable either static or dynamic tracing.
11. Gather the trace configuration file and the trace output files. Evaluate the trace messages or package the files for transfer to support for evaluation.

2 Configuring OpenView Tracing

Setting Up OpenView Tracing

To set up tracing of a problem, complete the following steps:

1. Determine the applications (process/daemon) for which you need to capture trace information. The success of this step is greatly influenced by the experience of the investigating support engineer.
2. Determine if the applications identified as being the root of the problems being experienced are trace-enabled applications, that is they incorporate OpenView Tracing. This procedure can only be used for trace-enabled applications. The non-trace-enable applications will need to use existing tracing mechanisms to capture trace output. In some cases a combination of both OpenView Tracing and existing tracing mechanisms may be needed to capture all the tracing information required for a given support problem.

   UNIX Systems

   On UNIX systems, look into the `OVTraceCfg.dat` file created under the `/var/opt/OV/datafiles/xpl/` directory by the local trace server.

   This file contains the names of all tracing-enabled applications that have registered with the `ovtrcd` process. The file also includes the components and categories defined for each trace-enabled application. Examine the `OVTraceCfg.dat` file and identify the trace-enabled applications and the components and categories defined for each application.

   The following is an extract from an `OVTraceCfg.dat` file:

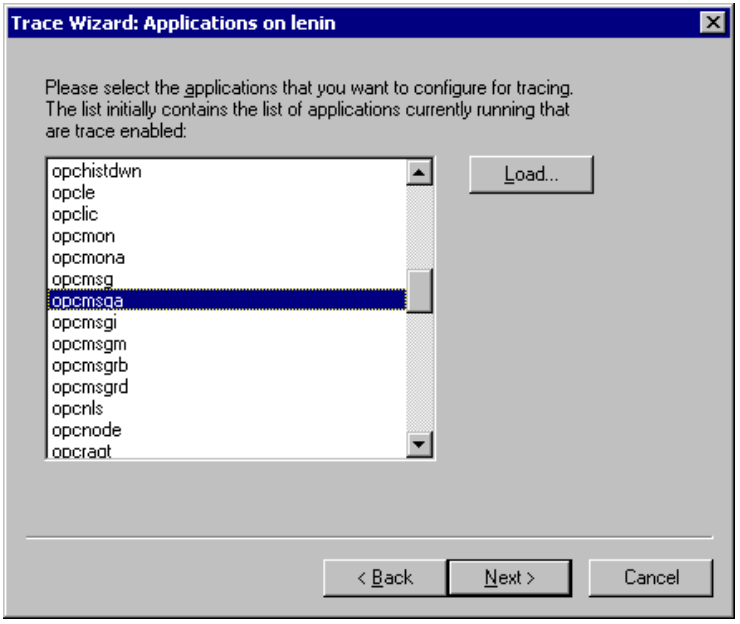
```
opcmsgm                 ← Application
   opc.db               ← Component
            Trace       ← Category
opcmsgm                 ← Application
   opc.lic              ← Component
            Trace       ← Category
opcmsgm                 ← Application
   opc.init             ← Component
            Trace       ← Category
```

```
opcactm                    ← Application
   xpl.runtime             ← Component
            Trace          ← Category
opcactm                    ← Application
   sec.core.auth           ← Component
            Proc           ← Category
……                         ← Continues for other Application,
                             Component, and Categories
                             defined.
……
```

Windows Systems

On Windows systems, you can install and use the TraceMon GUI to connect to the system running the trace server. The connection can be either local or remote. Once a connection has been established to the machine where the target trace server is running, the TraceMon GUI displays the names of the trace-enabled applications running on the machine (see Figure 2-1).

Figure 2-1 Applications Dialog



CAUTION It is recommended that not all applications are simultaneously selected. The volume of trace information generated can quickly become too much to analyze efficiently.

3. Determine if the trace-enabled application requires any additional pre-configuration steps to enable OpenView tracing. In general, this should not be necessary, but the implementation of OpenView tracing in some OpenView products does require additional pre-configuration steps. For example, the `NNM/ET` processes require the process `lrf` file be modified to output the OpenView trace messages.

NOTE Refer to OV product-specific reference documents to determine which applications within OpenView products require pre-configuration steps.

4. Determine which components and categories within each application need to be traced.

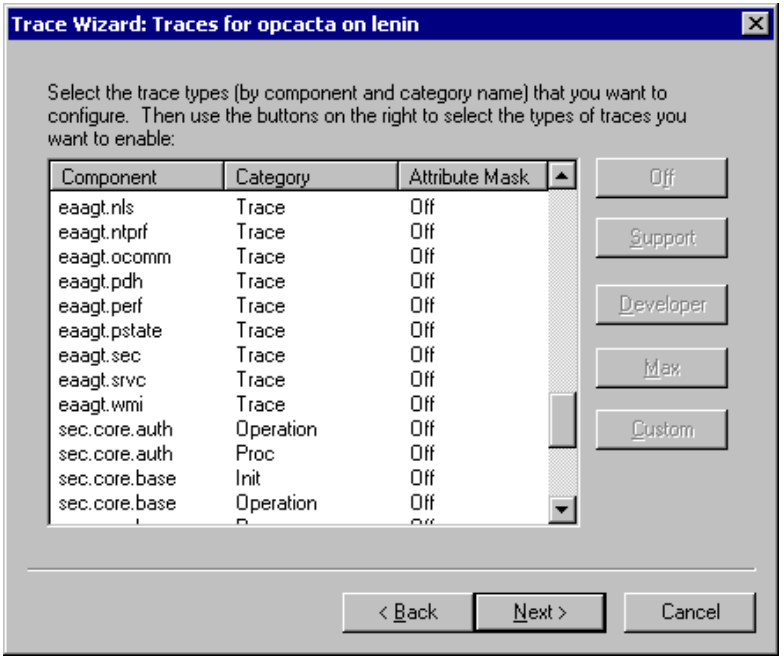
   UNIX Systems

   On UNIX systems, look into the `OVTraceCfg.dat` file created under the `/var/opt/OV/datafiles/xpl/` directory by the local trace server.

   Windows Systems

   On Windows systems, use the TraceMon GUI to connect to the system running the trace server. Once a connection has been established to the machine where the target trace server is running, the TraceMon GUI displays the names of the trace-enabled applications and their associated components and categories (see Figure 2-2).

Figure 2-2 Traces for OvNamespaceSec Dialog



Once you have identified the components and categories that are available, determine which ones relate to the problem for which you want to establish tracing. In general, the components and categories names are self-explanatory.

The product support teams often have supplemental support documentation that provides additional information on the combinations of applications, components, and categories that should be logically or functionally grouped together for a given support problem, for example, the *OVO for Windows Tracemon Usage Guide*.

CAUTION It is recommended that not all applications are simultaneously selected. The volume of trace information generated can quickly become too much to analyze efficiently.

5. Determine which attribute flags associated with each component/category combination need to be set. The recommended starting points are the usual support attributes: `Info`, `Warn`, and `Error`, unless directed to set the `Verbose` or `Developer` attributes (See Table 2-1).

CAUTION It is strongly recommend to avoid setting all attributes (`Max`), as this can significantly degrade system performance.

Table 2-1 Trace Attribute Flag

Attribute Group	Attribute Keywords	Recommendations
`Support`	`Info`	**Recommended for normal tracing activities** The `Support` setting should be used for all front line support tracing. The `Verbose` attribute can be use in conjunction with the other `Support` attributes.
	`Warn`	
	`Error`	
	`Verbose` (Optional)	

Table 2-1 Trace Attribute Flag

Attribute Group	Attribute Keywords	Recommendations
Developer	Info	**Limited to a 2nd, 3rd, or CPE level support tracing activities** The Developer attribute is targeted for trace messages that require the source code to be interpreted. The Developer setting should be used when source level detailed trace messages are requested. The Verbose attribute can be use in conjunction with the other Developer attributes
	Warn	
	Error	
	Developer	
	Verbose (Optional)	
Max	Info	**Not recommend for normal tracing activities** The Location and Stack attributes can greatly degrade system performance due to the overhead required to capture the tracing information. The Max attribute setting should be used with caution.
	Warn	
	Error	
	Developer	
	Verbose	
	Location	
	Stack	

The attribute keywords listed above are also used within a Trace Configuration File on the TRACE line. (See Example 1).

WARNING **Do not set the Location or Stack attributes or the use the Max option provide in the TraceMon GUI, unless absolutely necessary. These attribute setting can greatly degrade system performance.**

6. Determine which tracing configuration is most suitable for the tracing situation. Tracing can be static or dynamic, configured locally or remotely, written directly to a file, or transmitted to a local or remote trace server.

Static Tracing

Static Tracing is enabled during application start-up and permits application start-up traces to be captured. Once the application has started, the tracing configuration can not be modified. Static tracing can only be disabled by stopping the application. Unless the trace configuration file is renamed or moved, the tracing will resume upon the next application start-up. Static tracing can be configured in two ways:

- Using the `OVTrace.tcf` standard trace configuration file name and locating the file in the application start-up directory or in the `/var/opt/OV/datafiles/xpl/` directory.
- Defining an environment variable named `TRACE_CONFIG_FILE` to specify the name and location of the trace configuration file. When using this method, you can determined the configuration file name and location.

Dynamic Tracing

Dynamic tracing is enabled after the application has started and is dynamically configurable. Dynamic tracing can be configured using either one of the configuration clients, `ovtrccfg`, located in `/opt/OV/support/`, or `TraceMon` (Windows GUI) and a trace configuration file. Using the `TraceMon` tool, you can also dynamically set trace configurations through configuration dialogs.

Dynamic tracing can be configured locally or remotely. The configuration clients, `ovtrccfg` and `TraceMon` can configure a trace server that is running on the local machine or on a remote machine. Using the options available within these tools, you can dynamically configure tracing on a system. If you have direct access to the system, the local configuration option can be used. If you do not have direct access to the system, the remote configuration must be used.

TIP

If you have access to a Windows system with the `TraceMon` graphical tool installed, it may be easier to use the GUI to remotely configure the tracing configuration. The GUI provides additional features that allow for the configuration of multiple applications, sort and filter the trace output, and to save the trace configuration settings.

NOTE If working across a firewall, the tracing messages can only cross a firewall if port 5051 (TCP) is open.

The tracing output from both static and dynamic tracing can be written directly to a file or transmitted to a local or remote trace server. If the trace output is transmitted to a trace server (either local or remote) the monitor clients `ovtrcmon`, located in `/opt/OV/support/`, or `TraceMon` (Windows GUI) tools can be use to monitor the trace output. These monitor clients are capable of writing the trace output to a file or directly to standard out.

7. Create the trace configuration file. if required, to match the tracing configuration selected. See following example. Locate the trace configuration file in a directory that can be accessed by either the application or the configuration client. If using the static tracing model, specify the trace configuration file name using the `TRACE_CONFIG_FILE` environment variable. If using the dynamic tracing model, the trace configuration file can be located locally or remotely.

   Example - Sample Trace Configuration File

   This sample trace configuration file enables tracing on two applications, `opcmsgrb` and `opcmsgm`. The sink is configured as a socket with the system `supnode1` as the target server. The components selected are `opc.msg`, `opc.act` and `opc.int`. The associated category selected is `Trace`. The tracing attributes are set to the Support defaults of `Info`, `Warn`, and `Error` for all, with the `Verbose` attribute set on the `opc.msg` components.

```
TCF Version 3.2
APP: "opcmsgrb"
SINK: Socket "supnode1" "node=10.1.111.20;"
TRACE: "opc.actn" "Trace" Info Warn Error
TRACE: "opc.int" "Trace" Info Warn Error
TRACE: "opc.msg" "Trace" Info Warn Error Verbose
APP: "opcmsgm"
SINK: Socket "supnode1" "node=10.1.111.20;"
TRACE: "opc.actn" "Trace" Info Warn Error
TRACE: "opc.msg" "Trace" Info Warn Error Verbose
```

TIP

If possible use the `TraceMon` (Windows GUI) tool to create the trace configuration file.

8. Depending upon the trace configuration selected, enable either static or dynamic tracing. Refer to the appropriate following section for specific steps to enable static and dynamic tracing.
   - “Enabling Static Tracing” on page 23.
   - “Enabling Dynamic Tracing” on page 26

Enabling Static Tracing

To enable static tracing, complete the following steps:

1. Verify that the trace server is running.

   When using static tracing and the sink configuration is defined to a socket, the trace server must be started and running before the application is started. If the application is started before the trace server is started, tracing is not be possible. If the sink configuration is defined to a file, the trace server is not required to be running.

   UNIX Execute a command such as:

   ```
   ps -ef | grep ovtrcd
   ```

   to verify that the trace server process is running. The information returned should be similar to the following:

   ```
   root@ supnode1: ps -ef | grep ovtrcd
   root 18750     1  0  Mar  5  ?         0:00
   /opt/OV/bin/ovtrcd
   ```

   Windows Open the `Services` dialog and verify the state of the service named `HP OpenView Trace Service` is `Started`.

2. If using a configuration file named `OVTrace.tcf`, verify that the file is located in the application start-up directory, usually the `/root` directory on UNIX systems. If you are not using the `OVTrace.tcf` file, verify that the `TRACE_CONFIG_FILE` environment variable is used to specify the name and location of the trace configuration file.

3. Stop the target trace-enabled applications if it is running, using the required application specific commands.

4. Start the trace monitor client for the required tracing configuration if the application sink was set to `Socket`.

   Use the `ovtrcmon` monitor client on either the local or remote system.

NOTE

The TraceMon client will not work in this situation since a static configuration was used.

If the Sink was set to File, a monitor client is not required.

There are a number of options available with the ovtrcmon command. Refer to the ovtrcmon documentation for the complete option details.

To monitor trace messages, execute one of the following commands or a similar command using additional ovtrcmon command options:

To monitor trace messages from supnode1 and output traces to a file in binary format:

```
ovtrcmon -server supnode1-tofile /tmp/traceout.trc
```

To monitor trace messages from supnode1, display verbose format, output directly to standard out.

```
ovtrcmon -server supnode1 -verbose
```

To monitor trace messages from supnode1 , display short format, and redirect standard out to a file.

```
ovtrcmon -server supnode1 -short > /tmp/traces.trc
```

5. Start the target trace-enabled applications using the required application specific commands.
6. Execute 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.
7. Stop the target trace-enabled applications using the required application specific commands.
8. Collect the trace configuration file and the trace output files. Evaluate the trace messages or package the files for transfer to support for evaluation. If the trace output was written directly to a file (Sink to File), there may be multiple versions of the trace output files. The Sink to File configuration option Maxfiles allows for multiple trace output files. These files have an extension of .001 - .100 added to the filename.

9. Reconfigure the application to disable tracing before the application is restarted. If using the `OVTrace.tcf` file, the configuration file should be removed or renamed to prevent the application from reading the trace configuration file the next time the application starts. If using the `TRACE_CONFIG_FILE` environment variable, the value should be disabled or the specified configuration file should be removed or renamed.

WARNING **Tracing will restart the next time the application starts unless tracing is disabled.**

Enabling Dynamic Tracing

Dynamic tracing has a variety of implementation options including, local or remote configuration and local or remote monitoring. The procedure outlined below covers the general sequence of steps required to enable dynamic tracing. However, they make no attempt to cover all the possible configuration combinations. You must know which system, either local or remote is to be used to execute the commands.

1. Place the trace configuration file in a location to which the client configuration tool has access. The location can be a local or network directory.

   - The trace configuration file should not be named `OVTrace.tcf`, since this is a reserved filename.
   - The `TRACE_CONFIG_FILE` environment variable must not be defined.

2. Verify that the `ovtrcd` is running.

   The trace server must be started and running before the application is started. If the application is started before the trace server is started, tracing is not be possible.

   UNIX Execute a command such as:

   ```
   ps -ef | grep ovtrcd
   ```

   to verify that the trace server process is running. The information returned should be similar to the following:

   ```
   root@ supnode1: ps -ef | grep ovtrcd
   root 18750     1  0  Mar  5  ?        0:00
   /opt/OV/bin/ovtrcd
   ```

   Windows Open the `Services` dialog and verify the state of the service named `HP OpenView Trace Service` is `Started`.

3. Verify the targeted trace-enabled applications are running and that they were started after the trace server was started. If the trace server was started after the trace-enabled applications, tracing will not be possible.

4. Make a trace configuration request using either the `ovtrccfg` or `TraceMon` (Window GUI) configuration clients.

   If using the `ovtrccfg` configuration client, execute command:

   `ovtrccfg -server <server-name> <trace-config-file-name>`

   Windows If using the `TraceMon` GUI configuration client, start a new Trace Wizard and select the option to load a configuration file using the following steps.

   a. Start the `TraceMon` tool.

   b. From the `File` menu, select the `Trace Wizard` option, then select `Next`.

   c. Select the `Configure local applications by loading a saved configuration` option.

   d. Locate and select the trace configuration file from the `Open` dialog.

   This starts new tracing windows with the configuration setting from the selected trace configuration file.

5. Make a trace monitor request using either the `ovtrcmon` or `TraceMon` (Window GUI) monitor clients.

   There are a number of options available with the `ovtrcmon` command. Refer to the `ovtrcmon` documentation for the complete option details.

   To monitor trace messages, execute one of the following commands or a similar command using additional `ovtrcmon` command options:

   To monitor trace messages from `supnode1` and output traces to a file in binary format:

   `ovtrcmon -server supnode1-tofile /tmp/traceout.trc`

   To monitor trace messages from `supnode1`, display verbose format, output directly to standard out.

   `ovtrcmon -server supnode1 -verbose`

   To monitor trace messages from `supnode1` , display short format, and redirect standard out to a file.

   `ovtrcmon -server supnode1 -short > /tmp/traces.trc`

Windows If using the TraceMon GUI configuration client, start a new Trace Wizard and select the option to load a configuration file using the following steps.

a. Start the TraceMon tool.

b. From the File menu, select the Trace Wizard option, then select Next.

c. Select the Configure local applications by loading a saved configuration option.

d. Locate and select the trace configuration file from the Open dialog.

This starts new tracing windows with the configuration setting from the selected trace configuration file. The TraceMon tool can also be used to dynamically configure tracing using the displayed configuration dialogs.

6. Execute 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.

7. Stop or disable tracing using either the ovtrccfg or TraceMon configuration clients.

   If using the ovtrccfg configuration client, execute command:

   ```
   ovtrccfg -server <server-name> off
   ```

   Windows If using the TraceMon GUI configuration client, stop tracing using the following steps:

   - Select the configuration window associated with the tracing.
   - From the File menu, select the Close option.

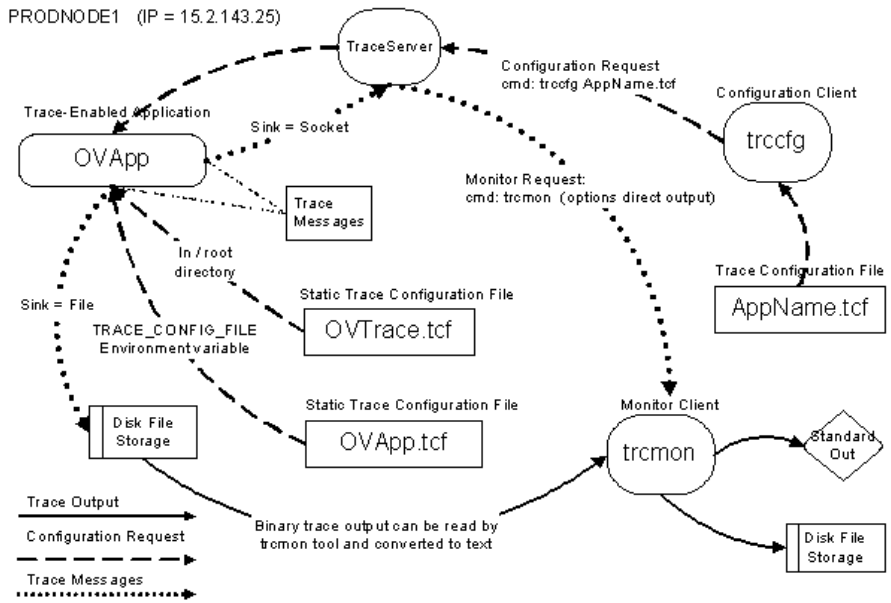
8. Collect the trace configuration file and the trace output files. Evaluate the trace messages or package the files for transfer to support for evaluation. If the trace output was written directly to a file (Sink to File), there may be multiple versions of the trace output files. The Sink to File configuration option Maxfiles allows for multiple trace output files. These files have an extension of .001 - .100 added to the filename.

Configuration Diagrams

Local Tracing Options

Figure 2-3 shows the static and dynamic tracing configurations available when limiting tracing to one particular node (not using any of the remote tracing capabilities).

Figure 2-3 Local Tracing Options



Dynamic Tracing: Starting with the `ovtrccfg` Tool

- Figure 2-3 illustrates that tracing can be configured using the `ovtrccfg` tool, which reads a Trace Configuration File named `AppName.tcf`.
- This trace configuration request is sent through the local trace server to the Trace-enabled application.

- The Trace-enabled application sends the trace messages to the local trace server (This assumes that the sink is configured as `Socket` to the local system).
- The `ovtrcmon` tool is used to monitor the trace messages, and can be configured to output the trace messages to standard out or to a disk file.

Static Tracing: Starting with the OVApp Process

Figure 2-3 also illustrates two ways static tracing can be configured to start when the `OVApp` starts:

- Creating a trace configuration file named `OVTrace.tcf` and placing this configuration file in the `/root` directory.
  - Creating a trace configuration file named `OVApp.tcf` (or an alternative file name) and defining the `TRACE_CONFIG_FILE` environment variable to reference this file.
  - The configuration file is read by the `OVApp` process during startup and will enable tracing within the `OVApp` application.
- The trace output can be output directly to a file, using the `Sink to File` configuration option, or it can be output to a trace server using the `Sink to Socket` configuration option.
- The `ovtrcmon` tool is used to monitor the trace messages directly from the trace server if the `Sink to Socket` option is used, or the `ovtrcmon` tool can read the binary trace output file (created by the `OVApp` application). In both cases the trace output can be written to standard out or to a disk file.

Additional Notes

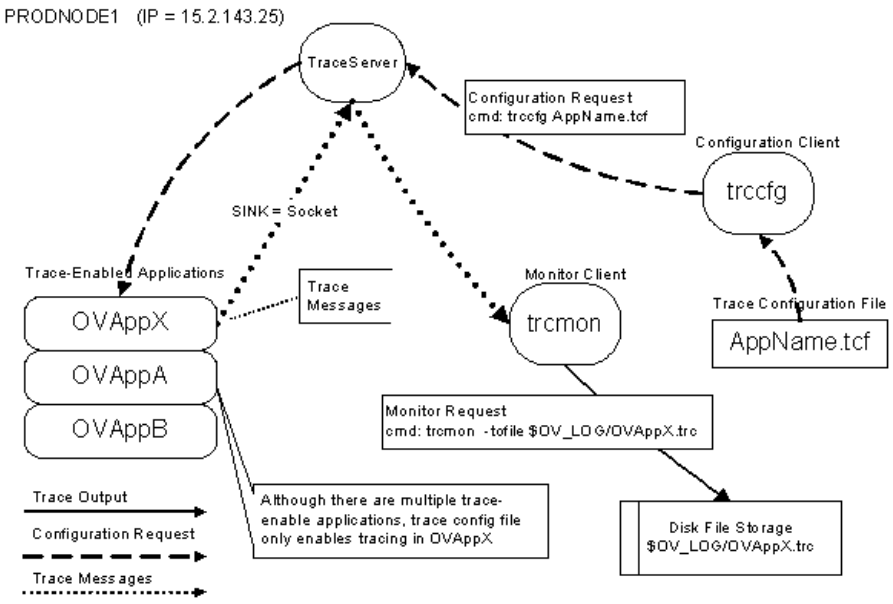
- There will be multiple trace-enabled applications and there will be other non trace-enabled applications running on the same system.
- The trace configuration file can specify more than one trace-enabled application.
- There should only be one `ovtrcd` process running on the system.

Local Dynamic Tracing Options

The tracing configuration objectives for local dynamic tracing are:

- Enable dynamic tracing on `OVAppX`. Use the localhost `ovtrcd` process as the `Sink`, use the trace configuration file named `AppName.tcf`.
- Tracing is monitored using the `ovtrcmon` tool.
- Trace messages are output in binary format to the file named `$OV_LOG/OVAppX.trc`

Figure 2-4 Local Dynamic Tracing Options



AppName.tcf File Contents

```
TCF Version 3.2
APP: "OVAppX"
SINK: Socket "PRODNODE1" "node=10.1.143.25;"
TRACE: "Comp1-Name" "Parms" Error Info Warn
TRACE: "Comp2-Name" "Init" Info Verbose
```

Establishing Local Dynamic Tracing

Figure 2-4 illustrates how local dynamic tracing can be configured. The steps you must execute are as follows:

1. Verify that the local `ovtrcd` process is running with the command:

   `ps -ef | grep ovtrcd`

2. Verify that the `OVAppX` process is running, and that it was started after the `ovtrcd` process.

3. Create the `AppName.tcf trace` configuration file as shown above. Substitute the actual `Application`, `Component`, and `Category` names.

4. Make a trace configuration request using the `ovtrccfg` tool with the command:

   `ovtrccfg AppName.tcf`

NOTE The local trace server is being used, therefore the `-server` option is not required.

5. Make a trace monitor request using `ovtrcmon` tool with the command:

   `ovtrcmon -tofile $OV_LOG/OVAppX.trc`

6. Execute application-specific commands to duplicate the problem or situation for which you need tracing information. When these actions are completed tracing can be disabled.

7. Disable tracing using the `ovtrccfg` tool with the command:

   `ovtrccfg off`

   The application will continue to run, only the tracing will stop.

8. Stop the monitoring of trace messages. Use **`Ctrl-C`** to kill the `ovtrcmon` tool.

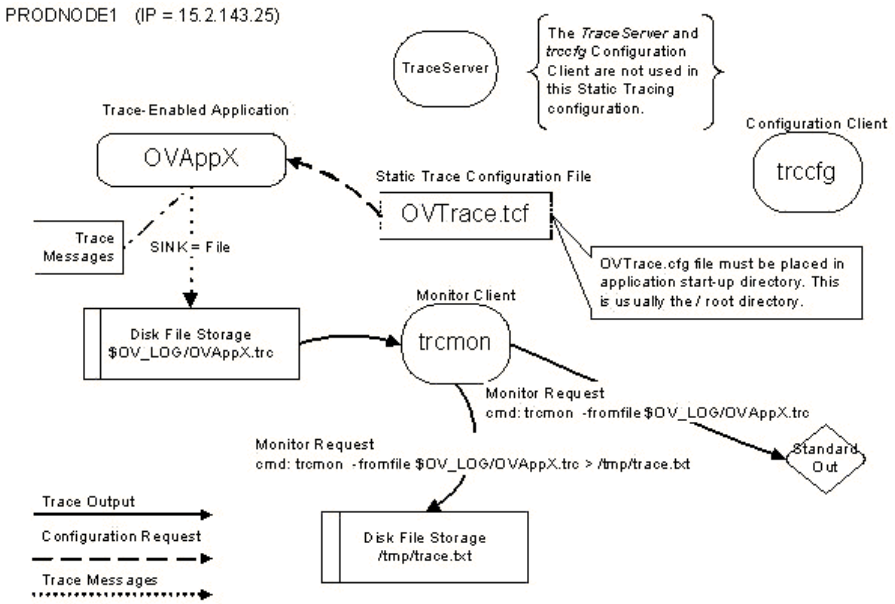
9. Examine the trace output or send the trace output files to your support organization.

Local Static Tracing Options

The tracing configuration objectives for local dynamic tracing are:

- Enable static tracing on `OVAppX` to capture startup trace messages, by placing a trace configuration file named `OVTrace.tcf` in the `/root` directory.
- Trace output is monitored using the `ovtrcmon` tool which reads the binary trace output file created. The trace output is converted to text, written to a text file and viewed on the standard out device.
- Trace messages are output in binary format to the file named `$OV_LOG/OVAppX.trc`

Figure 2-5 Local Static Tracing Options



AppName.tcf File Contents

```
TCF Version 3.2
APP: "OVAppX"
SINK: File "/var/opt/OV/share/log/OVAppX.trc"
                          "force=0;maxfiles=10;maxsize=100;"
TRACE: "Comp1-Name" "Parms" Error Info Warn
TRACE: "Comp2-Name" "Init" Info Verbose
```

Establishing Local Static Tracing

Figure 2-5 illustrates how local static tracing can be configured. The steps you must execute are as follows:

1. Create the `OVTrace.tcf` trace configuration file as shown above and place the file in the `/root` directory. Substitute the actual `Application`, `Component`, and `Category` names.
2. Stop the `OVAppX` process if it is running, using the appropriate application commands.
3. Restart the `OVAppX` process using the appropriate application commands.
4. Execute application-specific commands to duplicate the problem or situation for which you need tracing information. When these actions are completed tracing can be disabled.
5. Stop the `OVAppX` process using the appropriate application commands.
6. Make a trace monitor request using `ovtrcmon` tool to read from the trace output file, convert the trace messages to text, and redirect the output to a text file or directly to standard out. Use the command:

   To redirect to a text file, use the command:

   `ovtrcmon -fromfile $OV_LOG/OVAppX.trc > /tmp/trace.txt`

   To redirect to standard out, use the command:

   `ovtrcmon -fromfile $OV_LOG/OVAppX.trc`
7. Examine the trace output or send the trace output files to your support organization.

Trace Configuration Files

Trace configuration files are ASCII text files that can be viewed or modified using a standard text editor. The tracemon GUI can also be used to save a trace configuration file.

Trace configuration file syntax takes the following form. Details are explained in the following sections.

Line Type	Keyword Format & Parameters
Syntax Version	TCF Version 3.2
Application	APP: "Application-name"
Sink	SINK: File "Output-name" "force=[1/0]; maxfiles=[1..100];maxsize=[0..1000];"
	SINK: Socket "node" "node=<node name>;"
Trace	TRACE: "Component-name" "Category-name" <keyword list>

Syntax Version Line Details

The first line specifies that this is a trace configuration file and also specifies the syntax version of the file. It is case sensitive and must be specified exactly as shown below:

Format TCF Version 3.2

Application Line Details

The application line defines the name of the application to be traced. It must start with APP followed by a colon (:) and a space ( ). The application name should be in double quotes ("..."). The APP line should be immediately followed by a SINK line and then zero or more TRACE lines. Repeat this pattern for each application that you want to trace.

Format APP: "Application-name"

Example APP: "dbmanager"

Sink Line Details

The sink line specific the target to which the tracing information is directed. It must begin with `SINK` followed by a colon (`:`) and a space ( ). The arguments on the line should be separated by spaces. The `SINK` line has three arguments.

The first argument is the type of sink and must be one of the two keywords `File` or `Socket` and they should not be in quotes.

The second argument is the sink name and must be in double quotes (`"..."`). If the sink type is `File`, then this argument is the name of the file. If the sink type is `Socket`, then this argument is the name of the system running the trace server to which you want the application to send the trace messages.

The third argument is the sink options which must also be in double quotes (`"..."`), and each option must be followed by a semi-colon (;).

Format

```
SINK: File "Output-name" "force=[1/0];
maxfiles=[1..100];maxsize=[0..1000];"
```

or

```
SINK: Socket "node" "node=<node name>;"
```

Examples

```
SINK: File "C:\\TEMP\\Output.trc"
"force=0;maxfiles=10;maxsize=100;"
```

```
SINK: Socket "bigfoot" "node=10.1.115.98;"
```

File Sink Type Options

For the sink type `File`, the options are:

- `force=n`
- `maxfiles=n`
- `maxsize=n`

force The `force` option is followed by an integer value `n` that is either zero or non-zero. If the value is zero, trace output is buffered until the buffer is full before flushing the output to disk. This can speed up performance, but it also means if the application crashes, then the last traces may not get written to disk. If the value is non-zero, then the tracing subsystem forces the trace output physically to the disk after each trace event is written. The default is `force=0`.

maxfiles The `maxfiles` option is followed by an integer value between 1 and 100, and allows you to specify the number of historic trace log files to be retained. Each time an application starts to trace to the file, a backup is made of the previous file (if any) by adding ".001" to the name and renaming the file. If there was a ".001" file already, then it is renamed to ".002" and so on. the same backup scheme is in effect if the current log file reaches the maximum size.

maxsize The maxsize option is followed by an integer value (from 0 to 1000) which specifies the maximum amount of disk space in megabytes to be used for each trace output file. If the last block of data written to the trace output file causes the file to be larger than the specified maximum, then the next output will cause the current output file to be closed and backed up and a new output file to be created. A value of 0 is a special case that lets the file grow until you run out of disk space.

Socket Sink Type Options

For the sink type `Socket`, one option is supported: `node=node-name`.

The value of `node-name` is the communication path to the the system where the trace server is running and to where the trace output must be sent. It can be a DNS name, or IP address. If you want to create a configuration file that sends the output to the local Trace Server, regardless of what machine you copy the configuration file to, you must set the sink name to localhost and remove any `node=` option from the options string.

Trace Line Details

The trace line must begin with `TRACE` followed by a colon (`:`) and a space ( ). The arguments on the line must be separated by spaces.

The first argument is the trace component name and it must be in double quotes (`"..."`).

The second argument is the trace category name and it should also be in in double quotes (`"..."`). If you are using one of the standard categories in the code, it is mapped to the string value which you specify here). For the exact mapping of standard category constants to string values, see the language-appropriate documentation (C++, Java).

Format
```
TRACE: "Component-name" "Category-name"
<keyword list>
```

Example `TRACE: "database" "Parms" Error Info Warn Developer`

You can use "*" as the component name, category name, or both. This is useful when using applications in the mode where they read their configuration information directly from a file.

NOTE Configuration files that use this feature cannot be loaded into `tracemon`, or `ovtrccfg`.

How this works requires a little explanation. When an application tries to determine the settings for `component A` and `category B`, it first looks to see if the configuration contains an explicit trace definition for this pair. If the trace definition is there, it uses these settings. If it is not, then it looks to see if there is a configuration for `component A` and category *. If there is, it uses these settings. If there is not, then if looks to see if there is a configuration for `component *` and `category *`. If there is, it uses those settings. If not, then the trace is not activated.

The remaining parameters are a variable list of keyword options. At least one of the keywords: `Error`, `Info`, or `Warn` must be in the list. The supported keywords are:

Keyword	Attribute Description
`Error`	Enable traces marked as errors.
`Warn`	Enable traces marked as warnings.
`Info`	Enable traces marked as information.
`Developer`	Enable traces aimed at developers. In general, developer trace messages are not targeted at the front line support engineers, since the trace messages often require access to the source code before they can be effectively interpreted.
`Verbose`	Enable traces that produce very detailed output. Verbose trace messages can be both support and developer focused.

`Location`	Include source and line number information in the trace output if possible. The location trace message are not targeted at the front line support engineers, since the trace messages require access to the source code to be interpreted.
`Stack`	Include call stack information in the trace output if possible. The stack trace message are not targeted at the front line support engineers, since the trace messages require access to the source code to be interpreted.

Sample Trace Configuration File

SINK to File

```
TCF Version 3.2
APP: "dbmanager"
SINK: File "C:\\TEMP\\Output.trc" "force=0;maxfiles=10;maxsize=100;"
TRACE: "DbManager" "Parms" Error Info Warn Developer
TRACE: "DbManager" "Init" Info Verbose
TRACE: "DbManager" "Proc" Errore
```

SINK to Socket

```
TCF Version 3.2
APP: "nodedisc"
SINK: Socket "mgtstation" "node=10.1.112.99;"
TRACE: "Discovery" "Event" Error Info Warn Developer
TRACE: "Discovery" "Operation" Error Info Warn Developer
TRACE: "Discovery" "Trace" Error Info Warn Developer
```

SINK to File

```
TCF Version 3.2
APP: "ovet_disco"
SINK: Socket "bigfoot" "node=10.1.118.88;"
TRACE: "OvXplLog" "Trace" Info Warn Error Developer Verbose Location Stack
TRACE: "OvXplThread" "Trace" Info Warn Error Developer Verbose Location
Stack
TRACE: "OvXplIo" "Trace" Info Warn Error Developer Verbose Location Stack
TRACE: "OvDbil" "Event" Info Warn Error Developer Verbose Location Stack
TRACE: "OvDbil" "Proc" Info Warn Error Developer Verbose Location Stack
TRACE: "OvDbil" "Parms" Info Warn Error Developer Verbose Location Stack
TRACE: "OvDbil" "ResMgmt" Info Warn Error Developer Verbose Location Stack
```

3 Configuring Tracing For OVO

Tracing OpenView Products

OpenView tracing is the mechanism for tracing the latest OpenView products and will be incorporated into all future OpenView products.

In earlier version of OVO, tracing is enabled and disabled by setting parameter values within the `opcsvinfo` and `opcinfo` configuration files (See Example 3-1). In addition to controlled enabling of tracing, it also specifies which trace areas are enabled, and which on which processes tracing was enabled. The `opcsvinfo` file is read by the server processes and the `opcinfo` file was read by the agent processes.

Example 3-1 opcsvinfo Configuration File

```
Enable tracing for the  message/action flow and
initialization  and debug.
Generate trace output only for opcmsga and opcacta.
Enable debug output only for opcmsga.

OPC_TRACE TRUE
OPC_TRACE_AREA MSG,ACTN,INIT,DEBUG
OPC_TRC_PROCS opcmsga,opcacta
OPC_DBG_PROCS opcmsga
```

OpenView Tracing implements a hierarchy of elements starting with `Applications`, `Components`, `Categories` and `Attributes`. In OpenView Tracing terminology, the processes defined by `OPC_TRC_PROCS` and `OPC_DBG_PROCS` are referred to as `Applications`. The `TRACE AREAS` defined by the `OPC_TRACE_AREA` parameter are referred to as `subcomponents`. `Component` and `Attribute` elements were not part of the tracing configuration for OVO versions before OVO 8.0.

Component = *<component name>*

Trace area =*<sub-component>*

Category = `Trace`

To configure the same type of trace configuration using OpenView Tracing, you create a Trace Configuration File (See Example 2), enable tracing using the `ovtrccfg` tool, and monitor the trace messages using the `trcmon` tool.

Example 3-2 OpenView Trace Configuration File

```
TCF Version 3.2
APP: "opcmsga"
SINK: Socket "prodnode" "node=10.1.221.22;"
TRACE: "eaagt.actn" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.debug" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.init" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.msg" "Trace" Info Warn Error Developer Verbose
APP: "opcacta"
SINK: Socket "prodnode" "node=10.1.221.22;"
TRACE: "eaagt.actn" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.init" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.msg" "Trace" Info Warn Error Developer Verbose
```

OVO 8.0 Trace-Enabled Applications

All OVO 8.0 processes use OpenView Tracing (See Table 1). The OVO 8.0 Trace Enabled processes can be divided into three groups: 1) The Server processes; 2) The Agent processes; 3) The processes that link with a lower level component which implemented XPL Tracing. There are no pre-configuration steps required to enable tracing within OVO 8.0.

This was accomplished by either adding XPL Tracing into the OVO code base or by incorporating core functionality from a foundation component and linking with the corresponding library. In the case where XPL Tracing was added to the OVO code base, the existing tracing was converted to XPL Tracing. In cases where functionality from a foundation component was added, the XPL Tracing incorporated into these foundation components is pulled into OVO.

Table 3-1 OVO 8.0 Trace-enabled Applications on Management Server and Managed Nodes

Platform	Application Name
UNIX	coda
	codautil
	ctrlconfupd
	logdump
	opc_getmsg
	opc_ip_addr
	opccrpt
	opcnls
	ovas
	ovbbccb
	ovc
	ovcd
	ovcert
	ovcm
	ovconfchg
	ovconfd
	ovconfget
	ovcoreid
	ovcreg
	ovcs
	ovdeploy
	ovpolicy

Table 3-2 OVO 8.0 Trace-enabled Applications on Management Server

Platform	Application Name
UNIX	opc
	opc_dbinit
	opc_dflt_lang
	opc_rexec
	opcactm
	opcagtdbcfg
	opcagtutil
	opcauddwn
	opcbbcdist
	opccfgupld
	opccsacm
	opccsad
	opcctlm
	opcdbck
	opcdbinst
	opcdbmsgmv
	opcdbpwd
	opcdispm
	opcdistm
	opcforwm
	opchbp
	opchistdwn
	opcmsgm

Table 3-2 OVO 8.0 Trace-enabled Applications on Management Server

Platform	Application Name
UNIX	opcmsgrb
	opcmsgrd
	opcnode
	opcragt
	opcservice
	opcsvcm
	opcsvreg
	opcsw
	opcttnsm
	opcuiadm
	opcuiopadm
	opcuiwww
	ovoareqhdlr
	ovoareqsdr

Table 3-3 OVO 8.0 Trace-enabled Applications on Managed Nodes

Platform	Application Name
UNIX	opcacta
	opceca
	opcecaas
	opcle
	opcmon
	opcmona
	opcmsg
	opcmsga
	opcmsgi
	opctrapi

Server and Agent Applications

OVO Specific and OpenView Components

There are many components and sub-components defined for each application. The most important are `eaagt` and `opc`. Table 3-4 lists the OpenView Tracing Components which are defined for the Server and Agent processes.

Table 3-4 OVO 8.0 Server and Agent Components

OVO Component Name	Component Description
`eaagt`	Event Action Agent
`opc`	Management Server Control

Table 3-5 lists the components defined for the shared components which have been incorporated into the product.

Table 3-5 OV Shared Components

Application Name	Component and Subcomponent Names
Embedded Performance Agent	coda
	coda.dataaccess
	coda.kmdatamatrix
	coda.localmesa
	coda.logger
	coda.mesa
	coda.mesainstances
	coda_mesametricrdr
	coda.mesarea
	coda.prospector

Table 3-5 OV Shared Components

Application Name	Component and Subcomponent Names
Deployment Component	depl
Certificate Server Adapter	CSA-CertRequestImpl
	CSA-CertReqContainer
	CSA-Database
	Csa-Log
	Csa-Main
	csa.ovcmwrap
	Csa-RpcServer
	CSA-UpdateHandler
Control Component	ctrl.action
	ctrl.autoshutdown
	ctrl.component
	ctrl.controller
	ctrl.main
	ctrl.monitor
	ctrl.monitorproxy
	ctrl.ovc
	ctrl.process
	ctrl.rpcclient
	ctrl.rpcserver
	ctrl.soap
	ctrl.xml

Table 3-5 OV Shared Components

Application Name	Component and Subcomponent Names
Black Box Communication	bbc.cb
	bbc.fx
	bbc.fx.client
	bbc.fx.server
	bbc.http
	bbc.http.client
	bbc.http.dispatcher
	bbc.http.output
	bbc.http.server
	bbc.messenger
	bbc.rpc
	bbc.rpc.server
	bbc.soap
Configuration Management Component	conf.cluster
	conf.cluster.clioutputs
	conf.config
	conf.message
	conf.ovconfd
	conf.ovpolicy
	conf.policy

Table 3-5 OV Shared Components

Application Name	Component and Subcomponent Names
Security Core Component	sec.cm.client
	sec.cm.server
	sec.core.auth
	sec.core.base
	sec.core.ssl
Cross Platform Library	xpl.cfgfile
	xpl.config
	xpl.io
	xpl.log
	xpl.msg
	xpl.net
	xpl.runtime
	xpl.thread
	xpl.thread.mutex

OVO Specific and XPL Standard Categories

OVO trace areas are designated by OpenView categories. In addition, a number of the OpenView standard categories are used by both OVO processes and the lower level OpenView components used by OVO.

Table 3-6 lists the OpenView tracing categories which are defined for the `eaagt` and `opc` components.

NOTE These categories are referred to as `areas` in version of OVO before OVO 8.0.

Table 3-6 OVO 8.0 opc and eaagt Sub-components

Sub-Component Name	Sub-Component Description
OVO Specific Tracing Categories	
actn	Actions
agtid	IP independence using AgentID
alive	Agent alive checking
api	Configuration API
apm	Cluster APM
audit	Auditing
db	Database (dblib)
debug	Debug
dist	Distribution
fct	Function (control flow)
gui	Motif Userinterface
init	Initialization (e.g. err init, conf init)
inst	Installation

Table 3-6 OVO 8.0 opc and eaagt Sub-components

Sub-Component Name	Sub-Component Description
int	Internal
lic	Licensing
memerr	Problems with Memory allocation
memory	Rest of memory allocation
misc	Miscellaneous
mon	Monitor
msg	Message flow
name	Name resolution
nls	National Language Support (character set conversion,...)
ntprf	NT Performance trace
ocomm	Openagent communication
pdh	Performance data helper
perf	Performance
pstate	Policy and Source state changes
sec	Security
srvc	Service
wmi	Conversion of LE-Templates to WMI-Templates

Table 3-6 OVO 8.0 opc and eaagt Sub-components

Sub-Component Name	Sub-Component Description
Generic XPL Tracing Categories	
Trace	Generic traces
Proc	Procedure traces
Operation	Operational traces
Init	Initialization
Cleanup	Cleanup operation
Event	Event
Parms	Parameters
ResMgmt	Resource Management

NNM Pre-Configuration Requirements

There are no pre-configuration steps required to enable OpenView Tracing in OVO 8.0.

If NNM/ET is installed, some of the NNM processes require a pre-configuration step. The required steps are summarized below:

- The NNM/ET applications, these have names starting with: `ovet_`, require their associated `lrf` file be modified to include the hidden `-debug 4` option to enable tracing.
- The ECS Correlation Composer applications require the ECS and PMD tracing be configured to enable OpenView tracing.

Tracing OVO Processes

The following sample procedure provides an example of how to setup OpenView tracing on OVO processes. The example makes the following configuration assumptions:

- The `opcmsga` and `opcmsgm` processes running on a UNIX system must be traced.
- The `ovtrccfg` trace configuration client will be used to make configuration changes.
- The trace configuration file must be named: `$OV_CONF/OVOTrace.tcf`
- The `trcmon` trace monitor client will be used to monitor the traces.
- The trace output must be written to a file named: `$OV_LOG/OVOTrace.trc`

To setup OpenView tracing on OVO processes:

1. Identify the OVO processes that you want to trace. (The following example uses the `opcmsga` and `opcmsgm` processes).
2. Create a trace configuration file named `OvoTrace.tcf`. Locate the file in the `$OV_CONF` directory.

   This sample trace configuration file (See Example 3-3) enables tracing on the two OVO applications, `opcmsga` and `opcmsgm`. The Sink is configured as a socket with the machine `supnode1` as the target server. The components selected are the `opc` and `eaagt`. All the associated sub-components are selected except for the `DEBUG`

sub-components. This would correspond to selecting `All Areas except DEBUG`. The tracing attributes are set to the `Support` defaults of `Info`, `Warn`, and `Error` for all, with the `Verbose` attribute added to each component/sub-component combination entry.

Example 3-3 Trace Configuration File $OV_CONF/OVOTrace.tcf

```
TCF Version 3.2
APP: "opcmsgm"
SINK: Socket "supnode1" "node=10.111.1.21;"
TRACE: "opc.actn" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.agtid" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.alive" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.api" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.audit" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.db" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.dist" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.fct" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.gui" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.init" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.inst" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.int" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.lic" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.mem" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.memerr" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.misc" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.mon" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.msg" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.name" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.nls" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.ntprf" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.ocomm" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.pdh" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.perf" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.pstate" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.sec" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.srvc" "Trace" Info Warn Error Developer Verbose
TRACE: "opc.wmi" "Trace" Info Warn Error Developer Verbose
APP: "opcmsga"
SINK: Socket "supnode1" "node=10.111.1.21;"
TRACE: "eaagt.actn" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.agtid" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.alive" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.api" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.audit" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.db" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.dist" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.fct" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.gui" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.init" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.inst" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.int" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.lic" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.mem" "Trace" Info Warn Error Developer Verbose
```

```
TRACE: "eaagt.memerr" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.misc" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.mon" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.msg" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.name" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.nls" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.ntprf" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.ocomm" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.pdh" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.perf" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.pstate" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.sec" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.srvc" "Trace" Info Warn Error Developer Verbose
TRACE: "eaagt.wmi" "Trace" Info Warn Error Developer Verbose
```

If you have access to a Windows system with the `TraceMon` tool installed, it can be used to connect to the remote trace server to identify the application, component, and category names and to view the attributes. Refer to Example 3-1 and Example 3-2 for screen shots of associated dialogs from TraceMon GUI. Using the features provided within the TraceMon GUI tool, the required configuration setting can be selected and the configuration file saved.

Figure 3-1 TraceMon Applications Dialog for OVO Applications

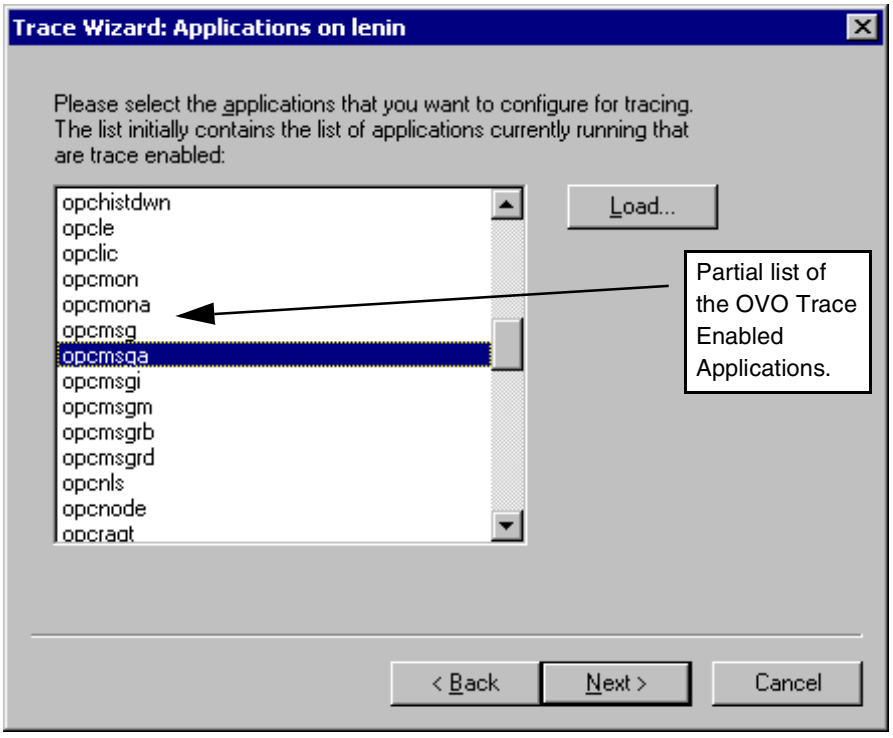
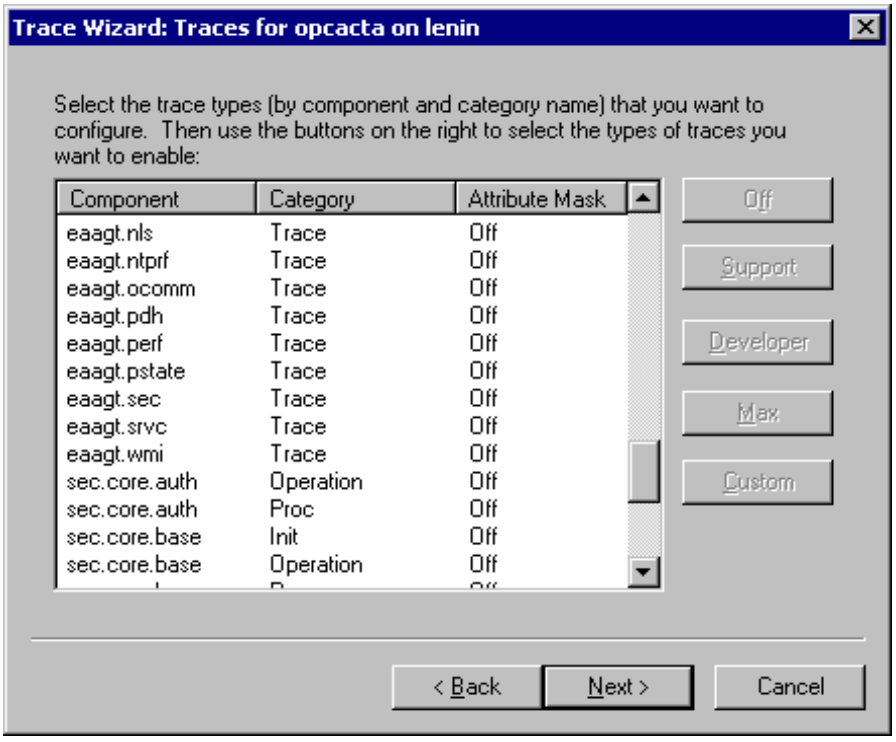


Figure 3-2 TraceMon Trace Dialog for OVO Applications



3. Verify the that the trace server is running on the system by executing the command:

   `ps -ef | grep ovtrcd`

   If the process is running, the information returned should be of the following form:

   ```
   root@ supnode1: ps -ef | grep ovtrcd
   root 18750  1  0  Mar  5  ?0:00 /opt/OV/bin/ovtrcd
   ```

4. Verify that the applications being traced, `opcmsga` and `opcmsgm`, are running on the system.

   To verify a process is running, execute commands of the following form:

   `ovstatus -c opcmsga opcmsgm`

The information returned should be of the following form:

```
root@ supnode1: ovstatus -c opcmsga opcmsgm

Name  PID  State Last Message(s)

opcmsga 15422  RUNNING  Initialization complete.

opcmsgm 26605  RUNNING  OVO Server Initialization
Complete.
```

5. Use the `ovtrccfg` configuration client to set the tracing configuration, using the command:

   `$OV_BIN/ovtrccfg -server supnode1 $OV_CONF/OvoTrace.tcf`

6. Use the `trcmon` monitor client to monitor the trace messages generated from the `opcmsga` and `opcmsgm` applications. To monitor the trace server running on the `supnode1` system and output the trace messages in binary format to the `$OV_LOG/OvoTrace.trc file`, enter the command:

   `$OV_BIN/trcmon -server supnode1 -tofile $OV_LOG/OvoTrace.trc`

7. Provided that the the processes to be traced are running (`opcmsga` and `opcmsgm` in our example), they should now be generating trace messages. Once enough trace information has been captured, stop the tracing. To Stop tracing, enter the command:

   `$OV_BIN/ovtrccfg off`

8. View the trace output using the `trcmon` monitor client. The trace output can be read from the binary trace file created using the `trcmon -fromfile` option. This option reads in a binary trace file and converts it to text. The converted trace messages can be sent directly to standard out or can be redirected to trace text file.

   To convert the binary trace file to text and send the output to standard out, enter the following command:

   `$OV_BIN/trcmon -fromfile $OV_LOG/OvoTrace.trc`

   To redirect the converted trace messages to a text file, enter the following command:

   `$OV_BIN/trcmon -fromfile $OV_LOG/OvoTrace.trc \`
   `> /tmp/trc.text`

   The binary `$OV_LOG/OvoTrace.trc` can be viewed from within the TraceMon Windows tool, where additional filtering can be done.

9. If analysis of the trace output is inconclusive, additional tracing can be done to capture more trace information. If needed, the trace configuration file can be modified to include or remove applications, components, categories or attributes.

Index