



## Trace Management

---

- [Information About Trace Management, on page 1](#)
- [How to Configure Conditional Debugging, on page 4](#)
- [Configuration Examples for Trace Management, on page 7](#)

### Information About Trace Management

The tracing functionality logs internal events. Trace files are automatically created and saved on the persistent storage device of specific platforms.

If the device has issues, the contents of the trace files are useful to troubleshoot the issue. The trace file outputs provide logs that are used to locate and solve the issue, and helps to get a detailed view of system actions and operations.

To view the recent trace information for a specific process, use the **show logging [process | Profile | process-helper]** command. The **process** keyword uses the first few letters of the name of a process and provides trace logs of the process that starts or matches with the entered string, the **profile** keyword lists the predefined set of process names, and the **profile-helper** keyword displays the available names.

To change the verbosity in a trace message output, you can adjust the trace level of processes using the **set platform software trace level** command. You can choose the **all** keyword to adjust the trace level for all the processes listed or you can select a specific process. When you select a specific process, there's also the option to adjust the trace level for a specific module, or you can use the **all-modules** keyword to adjust all the modules of processes.

### Introduction to Binary Tracing

Binary tracing is helpful in gathering trace information with a minimal impact on performance. In binary tracing, the tracing is always on for the system components and a basic level of trace is collected on all the time; thus, the data necessary for troubleshooting a problem has been captured the first time it occurs.

### Introduction to Conditional Debugging and Radioactive Tracing

The Conditional Debugging feature allows you to enable debugging and logging for specific features based on the set of conditions you define. This feature is useful in systems where many features are supported.

The Conditional debug allows granular debugging in a network that is operating at a large scale with a large number of features. It allows you to observe detailed debugs for granular instances within the system. This

type of debugging is useful when we need to debug only a particular session among thousands of sessions. It's also possible to specify multiple conditions.

A condition refers to a feature or identity, where an identity could be an interface, IP Address, or a MAC address and so on.

Conditional debugging is in contrast to the general debug command, that produces its output without discriminating on the feature objects that are being processed. General debug command consumes numerous system resources and impacts the system performance.

Radioactive tracing provides the ability to form a chain of execution for operations of interest across the system, at an increased verbosity level. This provides a way to print conditionally debug information (up to DEBUG Level or a specified level) across threads, processes, and function calls.

Radioactive Tracing when coupled with Conditional Debugging, provides a single debug command to debug all execution contexts related to the condition. You can execute this command without being aware of the various control flow processes of the feature within the box and without having to issue debugs at these processes individually.

## Tracing Levels

Trace level determines the types of traces outputted. Each trace message is assigned a trace level. If the trace level of a process or its module is set as greater than or equal to the level as the trace message, the trace message is displayed otherwise, it's skipped. For example, the default trace level is **Notice** level, so all traces with the **Notice** level and below the notice level are included while the traces above the **Notice** level are excluded.

The following table shows the available tracing levels, and provides descriptions of the message that are displayed with each tracing level. The tracing levels listed in the table are from the lowest to the highest order. The default trace level is **Notice**.

**Table 1: Tracing Levels and Descriptions**

Tracing Level	Description
Fatal	The message stating the process is aborted.
Emergency	The message is regarding an issue that makes the system unusable.
Alert	The message indicating that an action must be taken immediately.
Critical	The message is regarding a critical event causing loss of important functions.
Error	The message is regarding a system error.
Warning	The message is regarding a system warning.
Notice	The message is regarding a significant event.
Informational	The message is useful for informational purposes only.
Debug	The message provides debug-level output.

Tracing Level	Description
Verbose	All possible trace messages are sent.
Noise	All possible trace messages for the module are logged.  The noise level is always equal to the highest possible tracing level. Even if a future enhancement to tracing introduces a higher tracing level, the noise level will become equal to the level of that new enhancement.

## Payload Filter

This feature is used to filter trace messages. Trace messages contain actual debug information such as text strings, special characters, and variable arguments (strings), integers, long, IPv4/IPv6/MAC addresses, and so on. Using the payload feature, the trace messages can be filtered based on the selected criteria and without string operations.

You can use the following set and show commands to configure a payload filter and to view the applied filters.

**Table 2: Set Commands for Payload Filter**

<b>set platform software btrace-manager ... utm-pf enable</b>	Enables and disables the payload filtering feature.
<b>set platform software btrace-manager ... utm-pf disable</b>	
<b>set platform software btrace-manager ... consumer-name &lt;input&gt; create</b>	Creates and deletes consumer/stream.
<b>set platform software btrace-manager ... consumer-name &lt;input&gt; delete</b>	
<b>set platform software btrace-manager ... consumer-name &lt;input&gt; filter &lt;input&gt; add</b>	Applies and removes filter on stream/consumer
<b>set platform software btrace-manager ... consumer-name &lt;input&gt; filter &lt;input&gt; remove</b>	

**Table 3: Show Commands for Payload Filter**

<b>#show platform software btrace-manager ... utm-pf</b>	Shows the current status of the payload feature and other additional details
<b>show platform software btrace-manager ... utm-pf consumer-name &lt;input&gt; all-filters</b>	Shows all filters currently applied on consumer/stream.
<b>show platform software btrace-manager ... utm-pf consumer-name &lt;input&gt; all-luids</b>	Shows all or selected LUID of consumer for the applied filter.
<b>show platform software btrace-manager ... utm-pf consumer-name &lt;input&gt; filter &lt;input&gt;</b>	

<b>show platform software btrace-manager ... utm-pf message</b>	Shows consumer/stream messages.
---	---------------------------------

# How to Configure Conditional Debugging

## Conditional Debugging and Radioactive Tracing

Radioactive Tracing when coupled with Conditional Debugging, provides a single debug command to debug all execution contexts related to the condition. You can execute this command without being aware of the various control flow processes of the feature within the box and without having to issue debugs at these processes individually.

## Configuring Conditional Debugging

Follow the steps to configure conditional debugging:

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>enable</b> <b>Example:</b> Device> <b>enable</b>	Enables privileged EXEC mode. <ul style="list-style-type: none"> <li>• Enter your password if prompted.</li> </ul>
<b>Step 2</b>	<b>debug platform condition mac {mac-address}</b> <b>Example:</b> Device# <b>debug platform condition mac bc16.6509.3314</b>	Configures conditional debugging for the MAC Address specified.
<b>Step 3</b>	<b>debug platform condition start</b> <b>Example:</b> Device# <b>debug platform condition start</b>	Starts conditional debugging (this step starts radioactive tracing if there's a match on one of the preceding conditions).
<b>Step 4</b>	<b>show platform condition OR show debug</b> <b>Example:</b> Device# <b>show platform condition</b> Device# <b>show debug</b>	Displays the current conditions set.
<b>Step 5</b>	<b>debug platform condition stop</b> <b>Example:</b> Device# <b>debug platform condition stop</b>	Stops conditional debugging (this step stops radioactive tracing).

	Command or Action	Purpose
<b>Step 6</b>	<b>request platform software trace archive</b> [last {number} days] [target {crashinfo:   flashinfo:}]  <b>Example:</b> <pre># request platform software trace archive last 2 days</pre>	(Optional) Displays historical logs of merged tracefiles on the system. Filter on any combination of number of days or location.
<b>Step 7</b>	<b>show platform software trace</b> [filter-binary   level   message]  <b>Example:</b> <pre>Device# show platform software trace message</pre>	(Optional) Displays logs merged from the latest trace file. Filter on any combination of application condition, trace module name, and trace level. <ul style="list-style-type: none"> <li>• <b>filter-binary</b> - Filter the modules to be collated</li> <li>• <b>level</b> - Show trace levels</li> <li>• <b>message</b> - Show trace message ring contents</li> </ul> <p><b>Note</b> On the device:</p> <ul style="list-style-type: none"> <li>• Available from IOS console in addition to linux shell.</li> <li>• Generates a file with merged logs</li> <li>• Displays merged logs only from staging area.</li> </ul>
<b>Step 8</b>	<b>clear platform condition all</b>  <b>Example:</b> <pre>Device# clear platform condition all</pre>	Clears all conditions.

### What to do next



**Note** The commands **request platform software trace filter-binary** and **show platform software trace filter-binary** work in a similar way. The only difference is:

- **request platform software trace filter-binary** - Sources the data from historical logs.
- **show platform software trace filter-binary** – Sources the data from the flash Temp directory.

The `mac_log <..date..>` is the important file, as it provides messages for the MAC that is being debugged. The command **show platform software trace filter-binary** also generates the same flash files, and also prints the `mac_log` on the screen.

## Collecting Trace Files

To collect trace files from a device, follow these steps:

1. To request the tracelogs for a specific time period (For example: Five days), use the command:

```
Device# request platform software trace archive last 5 day
```

2. The system generates a tar ball (.gz file) of the tracelogs in the location **/flash**:

## Copying Archived Trace Files

The following is an example of the trace file for a switching device:

```
Device# dir crashinfo:/tracelogs
Directory of crashinfo:/tracelogs/

50664 -rwx 760 Sep 22 2015 11:12:21 +00:00 plogd_F0-0.bin_0.gz
50603 -rwx 991 Sep 22 2015 11:12:08 +00:00 fed_pmanlog_F0-0.bin_0.9558.20150922111208.gz
50610 -rw- 11 Nov 2 2015 00:15:59 +00:00 timestamp
50611 -rwx 1443 Sep 22 2015 11:11:31 +00:00
auto_upgrade_client_sh_pmanlog_R0-.bin_0.3817.20150922111130.gz
50669 -rwx 589 Sep 30 2015 03:59:04 +00:00 cfgwr-8021_R0-0.bin_0.gz
50612 -rwx 1136 Sep 22 2015 11:11:46 +00:00 reflector_803_R0-0.bin_0.1312.20150922111116.gz
50794 -rwx 4239 Nov 2 2015 00:04:32 +00:00 IOSRP_R0-0.bin_0.14239.20151101234827.gz
50615 -rwx 131072 Nov 2 2015 00:19:59 +00:00 linux_iosd_image_pmanlog_R0-0.bin_0
--More--
```

You can copy the trace files using one of the following options:

```
Device# copy crashinfo:/tracelogs ?
crashinfo: Copy to crashinfo: file system
flash: Copy to flash: file system
ftp: Copy to ftp: file system
http: Copy to http: file system
https: Copy to https: file system
null: Copy to null: file system
nvram: Copy to nvram: file system
rcp: Copy to rcp: file system
running-config Update (merge with) current system configuration
scp: Copy to scp: file system
startup-config Copy to startup configuration
syslog: Copy to syslog: file system
system: Copy to system: file system
tftp: Copy to tftp: file system
tmpsys: Copy to tmpsys: file system
```

The general syntax for copying onto a TFTP server is as follows:

```
Device# copy source: tftp:
Device# copy crashinfo:/tracelogs/IOSRP_R0-0.bin_0.14239.20151101234827.gz tftp:
Address or name of remote host []? 2.2.2.2
Destination filename [IOSRP_R0-0.bin_0.14239.20151101234827.gz]?
```




---

**Note** It's important to clear the generated report or archive files off the device so that there's flash space available for tracelog and other purposes.

---

## Configuring Payload Filter

To configure a payload filter, you must create a consumer and add the relevant payload filter data.

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>enable</b> <b>Example:</b> <pre>Device&gt; enable</pre>	Enables privileged EXEC mode. <ul style="list-style-type: none"> <li>• Enter your password if prompted.</li> </ul>
<b>Step 2</b>	<b>set platform software btrace-manager utm-pf enable</b> <b>Example:</b> <pre>Device# set platform software btrace-manager chassis active r0 utm-pf enable  Device# set platform software btrace-manager chassis active r0 utm-pf disable</pre>	Enables or disables the payload filter.
<b>Step 3</b>	<b>set platform software btrace-manager {consumer-name} create</b> <b>Example:</b> <pre>Device# set platform software btrace-manager chassis active r0 consumer-name utm_pf_test create</pre>	Creates a consumer name.
<b>Step 4</b>	<b>set platform software btrace-manager consumer {consumer-name} filter {input} add</b> <b>Example:</b> <pre>Device# set platform software btrace-manager chassis active r0 consumer-name utm_pf_test filter "Failed to retrieve an interface" add</pre>	Add a filter data.

## Configuration Examples for Trace Management

The following is an output example of the *show platform condition* command.

The following is a sample of the *debug platform condition stop* command.

```
Device# debug platform condition stop
Conditional Debug Global State: Stop
```

The following is an example of the **show logging** command for the **ios** process.

```

Device# show logging process ios
Logging display requested on 2022/10/27 09:32:06 (PDT) for Hostname: [vwlc_1_9222], Model:
[C9800-CL-K9], Version: [X.X.X], SN: [9ZY0U03YBM0], MD_SN: [9ZY0U03YBM0]

Displaying logs from the last 0 days, 0 hours, 10 minutes, 0 seconds
executing cmd on chassis 1 ...
Unified Decoder Library Init .. DONE
Found 1 UTF Streams

2022/10/27 09:31:52.835197577 {iosrp_R0-0}{1}: [parser_cmd] [26471]: (note): id=
console@console:user= cmd: 'show logging process ios' SUCCESS 2022/10/27 08:31:48.762 PST
2022/10/27 09:31:59.651965736 {iosrp_R0-0}{1}: [parser_cmd] [26471]: (note): id=
console@console:user= cmd: 'show logging process ios internal' SUCCESS 2022/10/27 08:31:56.485
PST
=====
===== Unified Trace Decoder Information/Statistics =====
=====
----- Decoder Input Information -----
=====
Num of Unique Streams .. 1
Total UTF To Process ... 1
Total UTM To Process ... 75403
UTM Process Filter ..... ios
MRST Filter Rules ..... 4
=====
----- Decoder Output Information -----
=====
First UTM TimeStamp ..... 2022/10/27 02:21:47.048461994
Last UTM TimeStamp ..... 2022/10/27 09:32:04.919540850
UTM [Skipped / Rendered / Total] .. 75401 / 2 / 75403
UTM [ENCODED] ..... 75266
UTM [PLAIN TEXT] ..... 94
UTM [DYN LIB] ..... 0
UTM [MODULE ID] ..... 0
UTM [TDL TAN] ..... 43
UTM [APP CONTEXT] ..... 0
UTM [MARKER] ..... 0
UTM [PCAP] ..... 0
UTM [LUID NOT FOUND] ..... 0
=====

```

The following is an example of the **show logging profile wireless** command.

```

Device# show logging profile wireless
Logging display requested on 2023/03/13 09:07:09 (UTC) for Hostname: [FABRIEK], Model:
[C8300-1N1S-4T2X], Version: [X.X.X], SN: [FDO24190V85], MD_SN: [FDO2451M13G]

Displaying logs from the last 0 days, 0 hours, 10 minutes, 0 seconds
executing cmd on chassis local ...
Unified Decoder Library Init .. DONE
Found 1 UTF Streams

2023/03/13 08:57:34.084609935 {iosrp_R0-0}{255}: [parser_cmd] [3793]: (note): id=
10.68.219.145@vty0:user= cmd: 'show logging profile wireless level info' SUCCESS 2023/03/13
08:57:31.376 UTC
2023/03/13 09:07:03.562290152 {iosrp_R0-0}{255}: [parser_cmd] [3793]: (note): id=
10.68.219.145@vty0:user= cmd: 'show logging profile wireless internal ' SUCCESS 2023/03/13
08:58:51.922 UTC
=====
===== Unified Trace Decoder Information/Statistics =====
=====
----- Decoder Input Information -----
=====

```



The following is an example of the **show logging process-helper** command.

## Trace Management

```
UTM [PCAP] ..... 0
UTM [LUID NOT FOUND] ..... 0
UTM Level [EMERGENCY / ALERT / CRITICAL / ERROR] .. 0 / 0 / 0 / 0
UTM Level [WARNING / NOTICE / INFO / DEBUG] ..... 0 / 1 / 0 / 0
UTM Level [VERBOSE / NOISE / INVALID] ..... 0 / 0 / 0
=====
```