C H A P T E R   **5**

VoIP Debug Commands

This chapter documents debug commands that are new or specific to the Cisco 1750 router. All other commands used with this feature are documented in the D*ebug Command Reference* chapter for the Cisco IOS Release12.0T.

- **debug voip ccapi error**
- **debug voip ccapi inout**
- **debug vpm all**
- **debug vpm dsp**
- **debug vpm port**
- **debug vpm signal**
- **debug vpm spi**
- **debug vtsp all**
- **debug vtsp dsp**
- **debug vtsp session**
- **debug vtsp stats**

Using Debug Commands

Debug commands are provided for most of the configurations in this document. You can use the debug commands to troubleshoot any configuration problems that you might be having on your network. Debug commands provide extensive, informative displays to help you interpret any possible problems.

Table 5-1 contains important information about debug commands.

Caution   Debugging is assigned a high priority in your router CPU process, and it can render your router unusable. For this reason, use debug commands only to troubleshoot specific problems. The best time to use debug commands is during periods of low network traffic and few users to decrease the likelihood that the debug command processing overhead affects network users.

(DRAFT LABEL) ALPHA DRAFT - CISCO CONFIDENTIAL

Table 5-1   Important Information About Debug Commands

About	Information
Additional documentation	You can find additional information and documentation about the debug commands in the *Debug Command Reference* document on the Cisco IOS software documentation CD-ROM that came with your router.
	If you are not sure where to find this document on the CD-ROM, use the Search function in the Verity Mosaic browser that comes with the CD-ROM.
Disabling debugging	To turn off any debugging, enter the **undebug all** command.
Telnet sessions	If you want to use debug command during a telnet session with your router, you must first enter the **terminal monitor** command.

debug voip ccapi error

Use the **debug voip ccapi error** EXEC command to trace error logs in the call control application programming interface (API). Use the **no** form of this command to disable debugging output.

> **[no] debug voip ccapi error**

Usage Guidelines

The **debug voip ccapi error** EXEC command traces the error logs in the call control API. When there are insufficient resources, error logs are generated during normal call processing. They are also generated when there are problems in the underlying network-specific code, the higher call session application, or the call control API itself.

This debug command shows error events or unexpected behavior in system software. In most cases, no events are generated.

debug voip ccapi inout

Use the **debug voip ccapi inout** EXEC command to trace the execution path through the call control application programming interface (API). Use the **no** form of this command to disable debugging output.

> **[no] debug voip ccapi inout**

Usage Guidelines

The **debug voip ccapi inout** EXEC command traces the execution path through the call control API, which serves as the interface between the call session application and the underlying network-specific software. You can use the output from this command to understand how calls are being handled by the router.

This command shows how a call flows through the system. Using this debug level, you can see the call setup and teardown operations performed on both the telephony and network call legs.

Sample Display

The following output shows the call setup indicated and accepted by the router:

```
router# debug voip ccapi inout
cc_api_call_setup_ind (vdbPtr=0x60BFB530, callInfo={called=, calling=, fdest=0},
callID=0x60BFAEB8)
cc_process_call_setup_ind (event=0x60B68478)
sess_appl: ev(14), cid(1), disp(0)
ccCallSetContext (callID=0x1, context=0x60A7B094)
ccCallSetPeer (callID=0x1, peer=0x60C0A868, voice_peer_tag=2, encapType=1,
dest-pat=14085231001, answer=)
ccCallSetupAck (callID=0x1)
```

The following output shows the caller entering DTMF digits until a dial-peer is matched:

```
cc_api_call_digit (vdbPtr=0x60BFB530, callID=0x1, digit=4, mode=0)
sess_appl: ev(8), cid(1), disp(0)
ssa: cid(1)st(0)oldst(0)cfid(-1)csize(0)in(1)fDest(0)
cc_api_call_digit (vdbPtr=0x60BFB530, callID=0x1, digit=1, mode=0)
sess_appl: ev(8), cid(1), disp(0)
ssa: cid(1)st(0)oldst(0)cfid(-1)csize(0)in(1)fDest(0)
cc_api_call_digit (vdbPtr=0x60BFB530, callID=0x1, digit=0, mode=0)
sess_appl: ev(8), cid(1), disp(0)
ssa: cid(1)st(0)oldst(0)cfid(-1)csize(0)in(1)fDest(0)
cc_api_call_digit (vdbPtr=0x60BFB530, callID=0x1, digit=0, mode=0)
sess_appl: ev(8), cid(1), disp(0)
ssa: cid(1)st(0)oldst(0)cfid(-1)csize(0)in(1)fDest(0)
cc_api_call_digit (vdbPtr=0x60BFB530, callID=0x1, digit=1, mode=0)
sess_appl: ev(8), cid(1), disp(0)
ssa: cid(1)st(0)oldst(0)cfid(-1)csize(0)in(1)fDest(0)
ccCallProceeding (callID=0x1, prog_ind=0x0)
ssaSetupPeer cid(1), destPat(14085241001), matched(8), prefix(), peer(60C0E710)
```

The following output shows the call setup over the IP network to the remote router:

```
ccCallSetupRequest (peer=0x60C0E710, dest=, params=0x60A7B0A8 mode=0, *callID=0x60B6C110)
ccIFCallSetupRequest: (vdbPtr=0x60B6C5D4, dest=, callParams={called=14085241001,
calling=14085231001, fdest=0, voice_peer_tag=104}, mode=0x0)
ccCallSetContext (callID=0x2, context=0x60A7B2A8)
```

The following output shows the called party is alerted, a codec is negotiated, and voice path is cut through:

```
cc_api_call_alert(vdbPtr=0x60B6C5D4, callID=0x2, prog_ind=0x8, sig_ind=0x1)
sess_appl: ev(6), cid(2), disp(0)
ssa: cid(2)st(1)oldst(0)cfid(-1)csize(0)in(0)fDest(0)-cid2(1)st2(1)oldst2(0)
ccCallAlert (callID=0x1, prog_ind=0x8, sig_ind=0x1)
ccConferenceCreate (confID=0x60B6C150, callID1=0x1, callID2=0x2, tag=0x0)
cc_api_bridge_done (confID=0x1, srcIF=0x60B6C5D4, srcCallID=0x2, dstCallID=0x1,
disposition=0, tag=0x0)
cc_api_bridge_done (confID=0x1, srcIF=0x60BFB530, srcCallID=0x1, dstCallID=0x2,
disposition=0, tag=0x0)
cc_api_caps_ind (dstVdbPtr=0x60B6C5D4, dstCallId=0x2,srcCallId=0x1, caps={codec=0x7,
fax_rate=0x7F, vad=0x3})
cc_api_caps_ind (dstVdbPtr=0x60BFB530, dstCallId=0x1,srcCallId=0x2, caps={codec=0x4,
fax_rate=0x2, vad=0x2})
cc_api_caps_ack (dstVdbPtr=0x60BFB530, dstCallId=0x1,srcCallId=0x2, caps={codec=0x4,
fax_rate=0x2, vad=0x2})
cc_api_caps_ack (dstVdbPtr=0x60B6C5D4, dstCallId=0x2,srcCallId=0x1, caps={codec=0x4,
fax_rate=0x2, vad=0x2})
sess_appl: ev(17), cid(1), disp(0)
ssa: cid(1)st(3)oldst(0)cfid(1)csize(0)in(1)fDest(0)-cid2(2)st2(3)oldst2(1)
```

The following output shows that the call is connected and voice is active:

```
cc_api_call_connected(vdbPtr=0x60B6C5D4, callID=0x2)
sess_appl: ev(7), cid(2), disp(0)
ssa: cid(2)st(4)oldst(1)cfid(1)csize(0)in(0)fDest(0)-cid2(1)st2(4)oldst2(3)
ccCallConnect (callID=0x1)
```

The following output shows how the system processes voice statistics and monitors voice quality during the call:

```
ccapi_request_rt_packet_stats (requestorIF=0x60B6C5D4, requestorCID=0x2,
     requestedCID=0x1, tag=0x60A7C598)
cc_api_request_rt_packet_stats_done (requestedIF=0x60BFB530, requestedCID=0x1,
     tag=0x60A7A4C4)
ccapi_request_rt_packet_stats (requestorIF=0x60B6C5D4, requestorCID=0x2,
     requestedCID=0x1, tag=0x60A7C598)
cc_api_request_rt_packet_stats_done (requestedIF=0x60BFB530, requestedCID=0x1,
     tag=0x60C1FE54)
ccapi_request_rt_packet_stats (requestorIF=0x60B6C5D4, requestorCID=0x2,
     requestedCID=0x1, tag=0x60A7C598)
cc_api_request_rt_packet_stats_done (requestedIF=0x60BFB530, requestedCID=0x1,
     tag=0x60A7A5F4)
ccapi_request_rt_packet_stats (requestorIF=0x60B6C5D4, requestorCID=0x2,
     requestedCID=0x1, tag=0x60A7C598)
cc_api_request_rt_packet_stats_done (requestedIF=0x60BFB530, requestedCID=0x1,
     tag=0x60A7A6D8)
ccapi_request_rt_packet_stats (requestorIF=0x60B6C5D4, requestorCID=0x2,
     requestedCID=0x1, tag=0x60A7C598)
cc_api_request_rt_packet_stats_done (requestedIF=0x60BFB530, requestedCID=0x1,
     tag=0x60A7ACBC)
```

The following output shows that disconnection is generated from the calling party and that call legs are torn down and disconnected:

```
cc_api_call_disconnected(vdbPtr=0x60BFB530, callID=0x1, cause=0x10)
sess_appl: ev(9), cid(1), disp(0)
ssa: cid(1)st(5)oldst(3)cfid(1)csize(0)in(1)fDest(0)-cid2(2)st2(5)oldst2(4)
ccConferenceDestroy (confID=0x1, tag=0x0)
cc_api_bridge_done (confID=0x1, srcIF=0x60B6C5D4, srcCallID=0x2, dstCallID=0x1,
disposition=0 tag=0x0)
cc_api_bridge_done (confID=0x1, srcIF=0x60BFB530, srcCallID=0x1, dstCallID=0x2,
disposition=0 tag=0x0)
sess_appl: ev(18), cid(1), disp(0)
ssa: cid(1)st(6)oldst(5)cfid(-1)csize(0)in(1)fDest(0)-cid2(2)st2(6)oldst2(4)
ccCallDisconnect (callID=0x1, cause=0x10 tag=0x0)
ccCallDisconnect (callID=0x2, cause=0x10 tag=0x0)
cc_api_call_disconnect_done(vdbPtr=0x60B6C5D4, callID=0x2, disp=0, tag=0x0)
sess_appl: ev(10), cid(2), disp(0)
ssa: cid(2)st(7)oldst(4)cfid(-1)csize(0)in(0)fDest(0)-cid2(1)st2(7)oldst2(6)
cc_api_call_disconnect_done(vdbPtr=0x60BFB530, callID=0x1, disp=0, tag=0x0)
sess_appl: ev(10), cid(1), disp(0)
ssa: cid(1)st(7)oldst(6)cfid(-1)csize(1)in(1)fDest(0)
```

debug vpm all

Use the **debug vpm all** EXEC command to enable debugging on all virtual voice-port module (VPM) areas. Use the **no** form of this command to disable debugging output.

[no] debug vpm all

Usage Guidelines

The **debug vpm all** EXEC command enables all of the **debug vpm** commands: **debug vpm spi**, **debug vpm signal**, and **debug vpm dsp**. For more information or sample output, refer to the individual commands in this chapter.

debug vpm dsp

Use the **debug vpm dsp** EXEC command to show messages from the digital signal processor (DSP) on the virtual voice-port module (VPM) to the router. Use the **no** form of this command to disable debugging output.

> **[no] debug vpm dsp**

Usage Guidelines

The **debug vpm dsp** command shows messages from the DSP on the VPM to the router; this command can be useful if you suspect that the VPM is not functional. It is a simple way to check if the VPM is responding to off-hook indications and to evaluate timing for signaling messages from the interface.

Sample Display

The following output shows the DSP timestamp and the router timestamp for each event and, for SIG_STATUS, the state value shows the state of the ABCD bits in the signaling message. This sample shows a call coming in on a foreign exchange office (FXO) interface.

The router waits for ringing to terminate before accepting the call. State=0x0 indicates ringing; state 0x4 indicates not ringing:

```
router# debug vpm dsp
ssm_dsp_message: SEND/RESP_SIG_STATUS: state=0x0 timestamp=58172 systime=40024
ssm_dsp_message: SEND/RESP_SIG_STATUS: state=0x4 timestamp=59472 systime=40154
ssm_dsp_message: SEND/RESP_SIG_STATUS: state=0x4 timestamp=59589 systime=40166
```

The following output shows the digits collected:

```
vcsm_dsp_message: MSG_TX_DTMF_DIGIT: digit=4
vcsm_dsp_message: MSG_TX_DTMF_DIGIT: digit=1
vcsm_dsp_message: MSG_TX_DTMF_DIGIT: digit=0
vcsm_dsp_message: MSG_TX_DTMF_DIGIT: digit=0
vcsm_dsp_message: MSG_TX_DTMF_DIGIT: digit=0
```

This shows the disconnect indication and the final call statistics reported by the DSP (which are then populated in the call history table):

```
ssm_dsp_message: SEND/RESP_SIG_STATUS: state=0xC timestamp=21214 systime=42882
vcsm_dsp_message: MSG_TX_GET_TX_STAT: num_tx_pkts=1019 num_signaling_pkts=0
num_comfort_noise_pkts=0 transmit_durtation=24150 voice_transmit_duration=20380
fax_transmit_duration=0
```

debug vpm port

Use the **debug vpm port** EXEC command to limit the debug output to a particular port. Use the **no** form of this command to disable debugging output.

[no] debug vpm port *slot-number*/*port*

Syntax Description

slot-number	Slot number in the router where the VIC is installed. Valid entries are from 0 to 2, depending on the slot where it has been installed.
port	Voice port. Valid entries are 0 or 1.

Usage Guidelines

Use the **debug vpm port** command to limit the debug output to a particular port. The debug output can be quite voluminous for a single port. A six-port chassis might create problems. Use this debug command with any or all of the other debug modes.

Examples

The following example shows **debug vpm dsp** messages only for port 0/0:

```
debug vpm dsp
debug vpm port 0/0
```

The following example shows the **debug vpm signal** messages only for ports 0/0 and 0/1:

```
debug vpm signal
debug vpm port 0/0
debug vpm port 0/1
```

The following example shows how to turn off debugging on a port:

```
no debug vpm port 0/0
```

The following example shows no output because port level debugs work in conjunction with other levels:

```
debug vpm port 0/0
```

Execution of **no debug all** turns off all port level debugging. It is usually a good idea to turn off all debugging and then, one by one, to enter the debug commands you are interested in. This helps to avoid confusion about which ports you are actually debugging.

debug vpm signal

Use the **debug vpm signal** EXEC command to collect debug information only for signaling events. Use the **no** form of this command to disable debugging output.

[no] debug vpm signal

Usage Guidelines

The **debug vpm signal** EXEC command collects debug information only for signaling events. This command can also be useful in resolving problems with signaling to a PBX.

(DRAFT LABEL) ALPHA DRAFT - CISCO CONFIDENTIAL

Sample Display

The following output shows that a ring is detected and that the router waits for the ringing to stop before accepting the call:

```
router# debug vpm signal
ssm_process_event: [1/0, 0.2, 15] fxols_onhook_ringing
ssm_process_event: [1/0, 0.7, 19] fxols_ringing_not
ssm_process_event: [1/0, 0.3, 6]
ssm_process_event: [1/0, 0.3, 19] fxols_offhook_clear
```

The following output shows that the call is connected:

```
ssm_process_event: [1/0, 0.3, 4] fxols_offhook_proc
ssm_process_event: [1/0, 0.3, 8] fxols_proc_voice
ssm_process_event: [1/0, 0.3, 5] fxols_offhook_connect
```

The following output confirms a disconnect from the switch and release with higher layer code:

```
ssm_process_event: [1/0, 0.4, 27] fxols_offhook_disc
ssm_process_event: [1/0, 0.4, 33] fxols_disc_confirm
ssm_process_event: [1/0, 0.4, 3] fxols_offhook_release
```

debug vpm spi

Use the **debug vpm spi** EXEC command to trace how the virtual voice-port module (VPM) serial peripheral interface (SPI) interfaces with the call control application programming interface (API). Use the **no** form of this command to disable debugging output.

[no] debug vpm spi

Usage Guidelines

The **debug vpm spi** EXEC command traces how the virtual voice-port module SPI interfaces with the call control API. This debug command displays information about how each network indication and application request is handled.

This debug level shows the internal workings of the voice telephony call state machine.

Sample Display

The following output shows that the call is accepted and presented to a higher layer code:

```
router# debug vpm spi
sp_set_sig_state: [1/0] packet_len=14 channel_id=129 packet_id=39 state=0xC timestamp=0x0
vcsm_process_event: [1/0, 0.5, 1] act_up_setup_ind
```

The following output shows that the higher layer code accepts the call, requests addressing information, and starts DTMF and dial-pulse collection. This also shows that the digit timer is started.

```
vcsm_process_event: [1/0, 0.6, 11] act_setup_ind_ack
dsp_voice_mode: [1/0 packet_len=22 channel_id=1 packet_id=73 coding_type=1
voice_field_size=160 VAD_flag=0 echo_length=128 comfort_noise=1 fax_detect=1
dsp_dtmf_mode: [1/0] packet_len=12 channel_id=1 packet_id=65 dtmf_or_mf=0
dsp_CP_tone_on: [1/0] packet_len=32 channel_id=1 packet_id=72 tone_id=3 n_freq=2
freq_of_first=350 freq_of_second=440 amp_of_first=4000 amp_of_second=4000 direction=1
on_time_first=65535 off_time_first=0 on_time_second=65535 off_time_second=0
dsp_digit_collect_on: [1/0] packet_len=22 channel_id=129 packet_id=35 min_inter_delay=550
max_inter_delay=3200 mim_make_time=18 max_make_time=75 min_brake_time=18
max_brake_time=75
vcsm_timer: 46653
```

The following output shows the collection of digits one by one until the higher level code indicates it has enough. The input timer is restarted with each digit, and the device waits in idle mode for connection to proceed.

```
vcsm_process_event: [1/0, 0.7, 25] act_dcollect_digit
dsp_CP_tone_off: [1/0] packet_len=10 channel_id=1 packet_id=71
vcsm_timer: 47055
vcsm_process_event: [1/0, 0.7, 25] act_dcollect_digit
dsp_CP_tone_off: [1/0] packet_len=10 channel_id=1 packet_id=71
vcsm_timer: 47079
vcsm_process_event: [1/0, 0.7, 25] act_dcollect_digit
dsp_CP_tone_off: [1/0] packet_len=10 channel_id=1 packet_id=71
vcsm_timer: 47173
vcsm_process_event: [1/0, 0.7, 25] act_dcollect_digit
dsp_CP_tone_off: [1/0] packet_len=10 channel_id=1 packet_id=71
vcsm_timer: 47197
vcsm_process_event: [1/0, 0.7, 25] act_dcollect_digit
dsp_CP_tone_off: [1/0] packet_len=10 channel_id=1 packet_id=71
vcsm_timer: 47217
vcsm_process_event: [1/0, 0.7, 13] act_dcollect_proc
dsp_CP_tone_off: [1/0] packet_len=10 channel_id=1 packet_id=71
dsp_digit_collect_off: [1/0] packet_len=10 channel_id=129 packet_id=36
dsp_idle_mode: [1/0] packet_len=10 channel_id=1 packet_id=68
```

The following output shows that the network voice path cuts through:

```
vcsm_process_event: [1/0, 0.8, 15] act_bridge
vcsm_process_event: [1/0, 0.8, 20] act_caps_ind
vcsm_process_event: [1/0, 0.8, 21] act_caps_ack
dsp_voice_mode: [1/0] packet_len=22 channel_id=1 packet_id=73 coding_type=6
voice_field_size=20 VAD_flag=1 echo_length=128 comfort_noise=1 fax_detect=1
```

The following output shows that the called-party end of the connection is connected:

```
vcsm_process_event: [1/0, 0.8, 8] act_connect
```

(DRAFT LABEL) ALPHA DRAFT - CISCO CONFIDENTIAL

The following output shows the voice quality statistics collected periodically:

```
vcsm_process_event: [1/0, 0.13, 17]
dsp_get_rx_stats: [1/0] packet_len=12 channel_id=1 packet_id=87 reset_flag=0
vcsm_process_event: [1/0, 0.13, 28]
vcsm_process_event: [1/0, 0.13, 29]
vcsm_process_event: [1/0, 0.13, 32]
vcsm_process_event: [1/0, 0.13, 17]
dsp_get_rx_stats: [1/0] packet_len=12 channel_id=1 packet_id=87 reset_flag=0
vcsm_process_event: [1/0, 0.13, 28]
vcsm_process_event: [1/0, 0.13, 29]
vcsm_process_event: [1/0, 0.13, 32]
vcsm_process_event: [1/0, 0.13, 17]
dsp_get_rx_stats: [1/0] packet_len=12 channel_id=1 packet_id=87 reset_flag=0
vcsm_process_event: [1/0, 0.13, 28]
vcsm_process_event: [1/0, 0.13, 29]
vcsm_process_event: [1/0, 0.13, 32]
```

The following output shows that the disconnection indication is passed to higher level code. The call connection is torn down, and final call statistics are collected.

```
vcsm_process_event: [1/0, 0.13, 4] act_generate_disc
vcsm_process_event: [1/0, 0.13, 16] act_bdrop
dsp_CP_tone_off: [1/0] packet_len=10 channel_id=1 packet_id=71
vcsm_process_event: [1/0, 0.13, 18] act_disconnect
dsp_get_levels: [1/0] packet_len=10 channel_id=1 packet_id=89
vcsm_timer: 48762
vcsm_process_event: [1/0, 0.15, 34] act_get_levels
dsp_get_tx_stats: [1/0] packet_len=12 channel_id=1 packet_id=86 reset_flag=1
vcsm_process_event: [1/0, 0.15, 31] act_stats_complete
dsp_CP_tone_off: [1/0] packet_len=10 channel_id=1 packet_id=71
dsp_digit_collect_off: [1/0] packet_len=10 channel_id=129 packet_id=36
dsp_idle_mode: [1/0] packet_len=10 channel_id=1 packet_id=68
vcsm_timer: 48762
dsp_set_sig_state: [1/0] packet_len=14 channel_id=129 packet_id=39 state=0x4
timestamp=0x0
vcsm_process_event: [1/0, 0.16, 5] act_wrelease_release
dsp_CP_tone_off: [1/0] packet_len=10 channel_id=1 packet_id=71
dsp_idle_mode: [1/0] packet_len=10 channel_id=1 packet_id=68
dsp_get_rx_stats: [1/0] packet_len=12 channel_id=1 packet_id=87 reset_flag=1
```

debug vtsp all

Use the **debug vtsp all** EXEC command to show debugging information for all of the **debug vtsp** commands. Use the **no** form of this command to disable debugging output.

> **[no] debug vtsp all**

Usage Guidelines

The **debug vtsp all** command enables the following debug voice telephony service provider (vtsp) commands: **debug vtsp session**, **debug vtsp error**, and **debug vtsp dsp**. For more information or sample output, refer to the individual commands in this chapter.

debug vtsp dsp

Use the **debug vtsp dsp** EXEC command to show messages from the digital signal processor (DSP) on the V.Fast Class (VFC) modem to the router. Use the **no** form of this command to disable debugging output.

> **[no] debug vtsp dsp**

Usage Guidelines

The **debug vtsp dsp** command shows messages from the DSP on the VFC to the router; this command is useful if you suspect that the VFC is not functional. It is a simple way to check if the VFC is responding to off-hook indications.

Sample Display

The following output shows the collection of DTMF digits from the DSP:

```
router# debug vtsp dsp
*Nov 30 00:44:34.491: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT: digit=3
*Nov 30 00:44:36.267: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT: digit=1
*Nov 30 00:44:36.571: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT: digit=0
*Nov 30 00:44:36.711: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT: digit=0
*Nov 30 00:44:37.147: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT: digit=2
```

debug vtsp session

Use the **debug vtsp session** EXEC command to trace how the router interacts with the digital signal processor (DSP) based on the signaling indications from the signaling stack and requests from the application. Use the **no** form of this command to disable debugging output.

> **[no] debug vtsp session**

Usage Guidelines

The **debug vtsp session** command displays information about how each network indication and application request is processed, signaling indications, and DSP control messages.

This debug level shows the internal workings of the voice telephony call state machine.

Sample Display

The following output shows that the call has been accepted and that the system is now checking for incoming dial-peer matches:

```
router# debug vtsp session
*Nov 30 00:46:19.535: vtsp_tsp_call_accept_check (sdb=0x60CD4C58,
calling_number=408 called_number=1): peer_tag=0
*Nov 30 00:46:19.535: vtsp_tsp_call_setup_ind (sdb=0x60CD4C58,
tdm_info=0x60B80044, tsp_info=0x60B09EB0, calling_number=408 called_number=1):
peer_tag=1
```

(DRAFT LABEL) ALPHA DRAFT - CISCO CONFIDENTIAL

The following output shows that a DSP has been allocated to process the call and indicate to the higher layer code:

```
*Nov 30 00:46:19.535: vtsp_do_call_setup_ind:
*Nov 30 00:46:19.535: dsp_open_voice_channel: [0:D:12] packet_len=12
channel_id=8737 packet_id=74 alaw_ulaw_select=0 transport_protocol=2
*Nov 30 00:46:19.535: dsp_set_playout_delay: [0:D:12] packet_len=18
channel_id=8737 packet_id=76 mode=1 initial=60 min=4 max=200 fax_nom=300
*Nov 30 00:46:19.535: dsp_echo_canceller_control: [0:D:12] packet_len=10
channel_id=8737 packet_id=66 flags=0x0
*Nov 30 00:46:19.539: dsp_set_gains: [0:D:12] packet_len=12 channel_id=8737
packet_id=91 in_gain=0 out_gain=0
*Nov 30 00:46:19.539: dsp_vad_enable: [0:D:12] packet_len=10 channel_id=8737
packet_id=78 thresh=-38
*Nov 30 00:46:19.559: vtsp_process_event: [0:D:12, 0.3, 13] act_setup_ind_ack
```

The following output shows that the higher layer code has accepted the call, placed the DSP in dual tone multifrequency (DTMF) mode, and collected digits:

```
*Nov 30 00:46:19.559: dsp_voice_mode: [0:D:12] packet_len=20 channel_id=8737
packet_id=73 coding_type=1 voice_field_size=160 VAD_flag=0 echo_length=64
comfort_noise=1 fax_detect=1
*Nov 30 00:46:19.559: dsp_dtmf_mode: [0:D:12] packet_len=10 channel_id=8737
packet_id=65 dtmf_or_mf=0
*Nov 30 00:46:19.559: dsp_cp_tone_on: [0:D:12] packet_len=30 channel_id=8737
packet_id=72 tone_id=3 n_freq=2 freq_of_first=350 freq_of_second=440
amp_of_first=4000 amp_of_second=4000 direction=1 on_time_first=65535
off_time_first=0 on_time_second=65535 off_time_second=0
*Nov 30 00:46:19.559: vtsp_timer: 278792
*Nov 30 00:46:22.059: vtsp_process_event: [0:D:12, 0.4, 25] act_dcollect_digit
*Nov 30 00:46:22.059: dsp_cp_tone_off: [0:D:12] packet_len=8 channel_id=8737
packet_id=71
*Nov 30 00:46:22.059: vtsp_timer: 279042
*Nov 30 00:46:22.363: vtsp_process_event: [0:D:12, 0.4, 25] act_dcollect_digit
*Nov 30 00:46:22.363: dsp_cp_tone_off: [0:D:12] packet_len=8 channel_id=8737
packet_id=71
*Nov 30 00:46:22.363: vtsp_timer: 279072
*Nov 30 00:46:22.639: vtsp_process_event: [0:D:12, 0.4, 25] act_dcollect_digit
*Nov 30 00:46:22.639: dsp_cp_tone_off: [0:D:12] packet_len=8 channel_id=8737
packet_id=71
*Nov 30 00:46:22.639: vtsp_timer: 279100
*Nov 30 00:46:22.843: vtsp_process_event: [0:D:12, 0.4, 25] act_dcollect_digit
*Nov 30 00:46:22.843: dsp_cp_tone_off: [0:D:12] packet_len=8 channel_id=8737
packet_id=71
*Nov 30 00:46:22.843: vtsp_timer: 279120
*Nov 30 00:46:23.663: vtsp_process_event: [0:D:12, 0.4, 25] act_dcollect_digit
*Nov 30 00:46:23.663: dsp_cp_tone_off: [0:D:12] packet_len=8 channel_id=8737
packet_id=71
*Nov 30 00:46:23.663: vtsp_timer: 279202
```

The following output shows that the call proceeded and that DTMF was disabled:

```
*Nov 30 00:46:23.663: vtsp_process_event: [0:D:12, 0.4, 15] act_dcollect_proc
*Nov 30 00:46:23.663: dsp_cp_tone_off: [0:D:12] packet_len=8 channel_id=8737
packet_id=71
*Nov 30 00:46:23.663: dsp_idle_mode: [0:D:12] packet_len=8 channel_id=8737
packet_id=68
```

The following output shows that the telephony call leg was conferenced with the packet network call leg and performed capabilities exchange with the network-side call leg:

```
*Nov 30 00:46:23.699: vtsp_process_event: [0:D:12, 0.5, 17] act_bridge
*Nov 30 00:46:23.699: vtsp_process_event: [0:D:12, 0.5, 22] act_caps_ind
*Nov 30 00:46:23.699: vtsp_process_event: [0:D:12, 0.5, 23] act_caps_ack
Go into voice mode with codec indicated in caps exchange.
*Nov 30 00:46:23.699: dsp_cp_tone_off: [0:D:12] packet_len=8 channel_id=8737
packet_id=71
*Nov 30 00:46:23.699: dsp_idle_mode: [0:D:12] packet_len=8 channel_id=8737
packet_id=68
*Nov 30 00:46:23.699: dsp_voice_mode: [0:D:12] packet_len=20 channel_id=8737
packet_id=73 coding_type=6 voice_field_size=20 VAD_flag=1 echo_length=64
comfort_noise=1 fax_detect=1
```

The following output shows the call connected at remote side:

```
*Nov 30 00:46:23.779: vtsp_process_event: [0:D:12, 0.5, 10] act_connect
```

The following output shows that disconnect was indicated, and passed to upper layers:

```
*Nov 30 00:46:30.267: vtsp_process_event: [0:D:12, 0.11, 5] act_generate_disc
```

The following output shows that the conference was torn down and disconnect handshake completed:

```
*Nov 30 00:46:30.267: vtsp_process_event: [0:D:12, 0.11, 18] act_bdrop
*Nov 30 00:46:30.267: dsp_cp_tone_off: [0:D:12] packet_len=8 channel_id=8737
packet_id=71
*Nov 30 00:46:30.267: vtsp_process_event: [0:D:12, 0.11, 20] act_disconnect
*Nov 30 00:46:30.267: dsp_get_error_stat: [0:D:12] packet_len=10 channel_id=0
packet_id=6 reset_flag=1
*Nov 30 00:46:30.267: vtsp_timer: 279862
```

The following output shows that the final DSP statistics were retrieved:

```
*Nov 30 00:46:30.275: vtsp_process_event: [0:D:12, 0.17, 30] act_get_error
*Nov 30 00:46:30.275: 0:D:12: rx_dropped=0 tx_dropped=0 rx_control=353
tx_control=338 tx_control_dropped=0 dsp_mode_channel_1=2 dsp_mode_channel_2=0
c[0]=71 c[1]=71 c[2]=71 c[3]=71 c[4]=68 c[5]=71 c[6]=68 c[7]=73 c[8]=83 c[9]=84
c[10]=87 c[11]=83 c[12]=84 c[13]=87 c[14]=71 c[15]=6
*Nov 30 00:46:30.275: dsp_get_levels: [0:D:12] packet_len=8 channel_id=8737
packet_id=89
*Nov 30 00:46:30.279: vtsp_process_event: [0:D:12, 0.17, 34] act_get_levels
*Nov 30 00:46:30.279: dsp_get_tx_stats: [0:D:12] packet_len=10 channel_id=8737
packet_id=86 reset_flag=1
*Nov 30 00:46:30.287: vtsp_process_event: [0:D:12, 0.17, 31] act_stats_complete
*Nov 30 00:46:30.287: dsp_cp_tone_off: [0:D:12] packet_len=8 channel_id=8737
packet_id=71
*Nov 30 00:46:30.287: dsp_idle_mode: [0:D:12] packet_len=8 channel_id=8737
packet_id=68
*Nov 30 00:46:30.287: vtsp_timer: 279864
```

The following output shows that the DSP channel was closed and released:

```
*Nov 30 00:46:30.287: vtsp_process_event: [0:D:12, 0.18, 6] act_wrelease_release
*Nov 30 00:46:30.287: dsp_cp_tone_off: [0:D:12] packet_len=8 channel_id=8737
packet_id=71
*Nov 30 00:46:30.287: dsp_idle_mode: [0:D:12] packet_len=8 channel_id=8737
packet_id=68
*Nov 30 00:46:30.287: dsp_close_voice_channel: [0:D:12] packet_len=8
channel_id=8737 packet_id=75
*Nov 30 00:46:30.287: vtsp_process_event: [0:D:12, 0.16, 42] act_terminate
```

debug vtsp stats

Use the **debug vtsp stats** EXEC command to debug periodic messages sent and received from the **digital signal processor** (DSP) requesting statistical information during the call. Use the **no** form of this command to disable debugging output.

> **[no] debug vtsp stats**

Usage Guidelines

The **debug vtsp stats** command generates a collection of DSP statistics for generating RTP Control Protocol (RTCP) packets and a collection of other statistical information.

Sample Display

The following output shows sample **debug vtsp stats** output:

```
router# debug vtsp stats
*Nov 30 00:53:26.499: vtsp_process_event: [0:D:14, 0.11, 19] act_packet_stats
*Nov 30 00:53:26.499: dsp_get_voice_playout_delay_stats: [0:D:14] packet_len=10
channel_id=8753 packet_id=83 reset_flag=0
*Nov 30 00:53:26.499: dsp_get_voice_playout_error_stats: [0:D:14] packet_len=10
channel_id=8753 packet_id=84 reset_flag=0
*Nov 30 00:53:26.499: dsp_get_rx_stats: [0:D:14] packet_len=10 channel_id=8753
packet_id=87 reset_flag=0
*Nov 30 00:53:26.503: vtsp_process_dsp_message: MSG_TX_GET_VOICE_PLAYOUT_DELAY:
clock_offset=-1664482334 curr_rx_delay_estimate=69 low_water_mark_rx_delay=69
high_water_mark_rx_delay=70
*Nov 30 00:53:26.503: vtsp_process_event: [0:D:14, 0.11, 28]
act_packet_stats_res
*Nov 30 00:53:26.503: vtsp_process_dsp_message: MSG_TX_GET_VOICE_PLAYOUT_ERROR:
predective_concelement_duration=0 interpolative_concelement_duration=0
silence_concelement_duration=0 retroactive_mem_update=0
buf_overflow_discard_duration=10 num_talkspurt_detection_errors=0
*Nov 30 00:53:26.503: vtsp_process_event: [0:D:14, 0.11, 29]
act_packet_stats_res
*Nov 30 00:53:26.503: vtsp_process_dsp_message: MSG_TX_GET_RX_STAT:
num_rx_pkts=152 num_early_pkts=-2074277660 num_late_pkts=327892
num_signaling_pkts=0 num_comfort_noise_pkts=0 receive_durtation=3130
voice_receive_duration=2970 fax_receive_duration=0 num_pack_ooseq=0
num_bad_header=0
*Nov 30 00:53:26.503: vtsp_process_event: [0:D:14, 0.11, 32]
act_packet_stats_res
```

Cisco 1750 Router Voice-over-IP Configuration Guide

(DRAFT LABEL) ALPHA DRAFT - CISCO CONFIDENTIAL