

debug clns esis events

Use the **debug clns esis events** privileged EXEC command to display uncommon End System-to-Intermediate System (ES-IS) events, including previously unknown neighbors, neighbors that have aged out, and neighbors that have changed roles (ES-IS, for example). The **no** form of this command disables debugging output.

debug clns esis events

no debug clns esis events

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the **debug clns esis events** command:

```
Router# debug clns esis events
```

```
ES-IS: ISH from aa00.0400.2c05 (Ethernet1), HT 30  
ES-IS: ESH from aa00.0400.9105 (Ethernet1), HT 150  
ES-IS: ISH sent to All ESs (Ethernet1): NET 49.0001.AA00.0400.6904.00, HT 299, HLEN 20
```

The following line indicates that the router received a hello packet (ISH) from the IS at MAC address aa00.0400.2c05 on the Ethernet1 interface. The hold time (or number of seconds to consider this packet valid before deleting it) for this packet is 30 seconds.

```
ES-IS: ISH from aa00.0400.2c05 (Ethernet1), HT 30
```

The following line indicates that the router received a hello packet (ESH) from the ES at MAC address aa00.0400.9105 on the Ethernet1 interface. The hold time is 150 seconds.

```
ES-IS: ESH from aa00.0400.9105 (Ethernet1), HT 150
```

The following line indicates that the router sent an IS hello packet on the Ethernet0 interface to all ESs on the network. The network entity title (NET) address of the router is 49.0001.0400.AA00.6904.00; the hold time for this packet is 299 seconds; and the header length of this packet is 20 bytes.

```
ES-IS: ISH sent to All ESs (Ethernet1): NET 49.0001.AA00.0400.6904.00, HT 299, HLEN 20
```

debug clns isis packets

Use the **debug clns isis packets** privileged EXEC command to enable display information on End System-to-Intermediate System (ES-IS) packets that the router has received and sent. The **no** form of this command disables debugging output.

debug clns isis packets

no debug clns isis packets

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the **debug clns isis packets** command:

```
Router# debug clns isis packets

ES-IS: ISH sent to All ESs (Ethernet0): NET
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00, HT 299, HLEN 33
ES-IS: ISH sent to All ESs (Ethernet1): NET
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00, HT 299, HLEN 34
ES-IS: ISH from aa00.0400.6408 (Ethernet0), HT 299
ES-IS: ISH sent to All ESs (Tunnel0): NET
47.0005.80ff.ef00.0000.0001.5940.1600.0906.4023.00, HT 299, HLEN 34
IS-IS: ESH from 0000.0c00.bda8 (Ethernet0), HT 300
```

The following line indicates that the router has sent an IS hello packet on Ethernet0 to all ESs on the network. This hello packet indicates that the NET of the router is 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00. The hold time for this packet is 299 seconds. The packet header is 33 bytes in length.

```
ES-IS: ISH sent to All ESs (Ethernet0): NET
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00, HT 299, HLEN 33
```

The following line indicates that the router has sent an IIS hello packet on Ethernet1 to all ESs on the network. This hello packet indicates that the network entity title (NET) of the router is 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00. The hold time for this packet is 299 seconds. The packet header is 33 bytes in length.

```
ES-IS: ISH sent to All ESs (Ethernet1): NET
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00, HT 299, HLEN 34
```

The following line indicates that the router received a hello packet on Ethernet0 from an intermediate system, aa00.0400.6408. The hold time for this packet is 299 seconds.

```
ES-IS: ISH from aa00.0400.6408 (Ethernet0), HT 299
```

The following line indicates that the router has sent an IS hello packet on Tunnel0 to all ESs on the network. This hello packet indicates that the NET of the router is 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00. The hold time for this packet is 299 seconds. The packet header is 33 bytes in length.

```
ES-IS: ISH sent to All ESs (Tunnel0): NET
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00, HT 299, HLEN 34
```

The following line indicates that on Ethernet0, the router received a hello packet from an end system with an SNPA of 0000.0c00.bda8. The hold time for this packet is 300 seconds.

```
IS-IS: ESH from 0000.0c00.bda8 (Ethernet0), HT 300
```

debug clns events

Use the **debug clns events** privileged EXEC command to display CLNS events that are occurring at the router. The **no** form of this command disables debugging output.

debug clns events

no debug clns events

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the **debug clns events** command:

```
Router# debug clns events

CLNS: Echo PDU received on Ethernet3 from 39.0001.2222.2222.2222.00!
CLNS: Sending from 39.0001.3333.3333.3333.00 to 39.0001.2222.2222.2222.00
      via 2222.2222.2222 (Ethernet3 0000.0c00.3a18)
CLNS: Forwarding packet size 117
      from 39.0001.2222.2222.2222.00
      to 49.0002.0001.AAAA.AAAA.AAAA.00
      via 49.0002 (Ethernet3 0000.0c00.b5a3)
CLNS: RD Sent on Ethernet3 to 39.0001.2222.2222.2222.00 @ 0000.0c00.3a18,
      redirecting 49.0002.0001.AAAA.AAAA.AAAA.00 to 0000.0c00.b5a3
```

The following line indicates that the router received an echo PDU on Ethernet3 from source network service access point (NSAP) 39.0001.2222.2222.2222.00. The exclamation point at the end of the line has no significance.

```
CLNS: Echo PDU received on Ethernet3 from 39.0001.2222.2222.2222.00!
```

The following lines indicate that the router at source NSAP 39.0001.3333.3333.3333.00 is sending a CLNS echo packet to destination NSAP 39.0001.2222.2222.2222.00 via an IS with system ID 2222.2222.2222. The packet is being sent on the Ethernet3 interface, with a MAC address of 0000.0c00.3a18.

```
CLNS: Sending from 39.0001.3333.3333.3333.00 to 39.0001.2222.2222.2222.00
      via 2222.2222.2222 (Ethernet3 0000.0c00.3a18)
```

The following lines indicate that a CLNS echo packet 117 bytes in size is being sent from source NSAP 39.0001.2222.2222.2222.00 to destination NSAP 49.0002.0001.AAAA.AAAA.AAAA.00 via the router at NSAP 49.0002. The packet is being forwarded on the Ethernet3 interface, with a MAC address of 0000.0c00.b5a3.

```
CLNS: Forwarding packet size 117
      from 39.0001.2222.2222.2222.00
      to 49.0002.0001.AAAA.AAAA.AAAA.00
      via 49.0002 (Ethernet3 0000.0c00.b5a3)
```

The following lines indicate that the router sent a redirect packet on the Ethernet3 interface to the NSAP 39.0001.2222.2222.2222.00 at MAC address 0000.0c00.3a18 to indicate that NSAP 49.0002.0001.AAAA.AAAA.AAAA.00 can be reached at MAC address 0000.0c00.b5a3.

```
CLNS: RD Sent on Ethernet3 to 39.0001.2222.2222.2222.00 @ 0000.0c00.3a18,
      redirecting 49.0002.0001.AAAA.AAAA.AAAA.00 to 0000.0c00.b5a3
```

debug clns igrp packets

Use the **debug clns igrp packets** privileged EXEC command to display debugging information on all ISO-IGRP routing activity. The **no** form of this command disables debugging output.

debug clns igrp packets

no debug clns igrp packets

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the **debug clns igrp packets** command:

```
Router# debug clns igrp packets

ISO-IGRP: Hello sent on Ethernet3 for DOMAIN_green1
ISO-IGRP: Received hello from 39.0001.3333.3333.00, (Ethernet3), ht 51
ISO-IGRP: Originating level 1 periodic update
ISO-IGRP: Advertise dest: 2222.2222.2222
ISO-IGRP: Sending update on interface: Ethernet3
ISO-IGRP: Originating level 2 periodic update
ISO-IGRP: Advertise dest: 0001
ISO-IGRP: Sending update on interface: Ethernet3
ISO-IGRP: Received update from 3333.3333.3333 (Ethernet3)
ISO-IGRP: Opcode: area
ISO-IGRP: Received level 2 adv for 0001 metric 1100
ISO-IGRP: Opcode: station
ISO-IGRP: Received level 1 adv for 3333.3333.3333 metric 1100
```

The following line indicates that the router is sending a hello packet to advertise its existence in the DOMAIN_green1 domain:

```
ISO-IGRP: Hello sent on Ethernet3 for DOMAIN_green1
```

The following line indicates that the router received a hello packet from a certain network service access point (NSAP) on the Ethernet3 interface. The hold time for this information is 51 seconds.

```
ISO-IGRP: Received hello from 39.0001.3333.3333.00, (Ethernet3), ht 51
```

The following lines indicate that the router is generating a Level 1 update to advertise reachability to destination NSAP 2222.2222.2222 and that it is sending that update to all systems that can be reached through the Ethernet3 interface:

```
ISO-IGRP: Originating level 1 periodic update
ISO-IGRP: Advertise dest: 2222.2222.2222
ISO-IGRP: Sending update on interface: Ethernet3
```

The following lines indicate that the router is generating a Level 2 update to advertise reachability to destination area 1 and that it is sending that update to all systems that can be reached through the Ethernet3 interface:

```
ISO-IGRP: Originating level 2 periodic update
ISO-IGRP: Advertise dest: 0001
ISO-IGRP: Sending update on interface: Ethernet3
```

The following lines indicate that the router received an update from NSAP 3333.3333.3333 on Ethernet3. This update indicated the area the router at this NSAP could reach.

```
ISO-IGRP: Received update from 3333.3333.3333 (Ethernet3)
ISO-IGRP: Opcode: area
```

The following lines indicate that the router received an update advertising that the source of that update can reach area 1 with a metric of 1100. A station opcode indicates that the update included system addresses.

```
ISO-IGRP: Received level 2 adv for 0001 metric 1100
ISO-IGRP: Opcode: station
```

debug clns packet

Use the **debug clns packet** privileged EXEC command to display information about packet receipt and forwarding to the next interface. The **no** form of this command disables debugging output.

debug clns packet

no debug clns packet

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the **debug clns packet** command:

```
Router# debug clns packet

CLNS: Forwarding packet size 157
      from 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.00 STUPI-RBS
      to 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00
      via 1600.8906.4017 (Ethernet0 0000.0c00.bda8)
CLNS: Echo PDU received on Ethernet0 from
      47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00!
CLNS: Sending from 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00 to
      47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00
      via 1600.8906.4017 (Ethernet0 0000.0c00.bda8)
```

In the following lines, the first line indicates that a Connectionless Network Service (CLNS) packet of size 157 bytes is being forwarded. The second line indicates the network service access point (NSAP) and system name of the source of the packet. The third line indicates the destination NSAP for this packet. The fourth line indicates the next-hop system ID, interface, and SNPA of the router interface used to forward this packet.

```
CLNS: Forwarding packet size 157
      from 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.00 STUPI-RBS
      to 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00
      via 1600.8906.4017 (Ethernet0 0000.0c00.bda8)
```

In the following lines, the first line indicates that the router received an Echo PDU on the specified interface from the source NSAP. The second line indicates which source NSAP is used to send a CLNS packet to the destination NSAP, as shown on the third line. The fourth line indicates the next-hop system ID, interface, and SNPA of the router interface used to forward this packet.

```
CLNS: Echo PDU received on Ethernet0 from
      47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00!
CLNS: Sending from 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00 to
      47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00
      via 1600.8906.4017 (Ethernet0 0000.0c00.bda8)
```

debug clns routing

Use the **debug clns routing** privileged EXEC command to display debugging information for all Connectionless Network Service (CLNS) routing cache updates and activities involving the CLNS routing table. The **no** form of this command disables debugging output.

debug clns routing

no debug clns routing

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the **debug clns routing** command:

```
Router# debug clns routing
```

```
CLNS-RT: cache increment:17  
CLNS-RT: Add 47.0023.0001.0000.0000.0003.0001 to prefix table, next hop 1920.3614.3002  
CLNS-RT: Aging cache entry for: 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.06  
CLNS-RT: Deleting cache entry for: 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.06
```

The following line indicates that a change to the routing table has resulted in an addition to the fast-switching cache:

```
CLNS-RT: cache increment:17
```

The following line indicates that a specific prefix route was added to the routing table, and indicates the next-hop system ID to that prefix route. In other words, when the router receives a packet with the prefix 47.0023.0001.0000.0000.0003.0001 in that packet's destination address, it forwards that packet to the router with the MAC address 1920.3614.3002.

```
CLNS-RT: Add 47.0023.0001.0000.0000.0003.0001 to prefix table, next hop 1920.3614.3002
```

The following lines indicate that the fast-switching cache entry for a certain network service access point (NSAP) has been invalidated and then deleted:

```
CLNS-RT: Aging cache entry for: 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.06  
CLNS-RT: Deleting cache entry for: 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.06
```

debug cls message

Use the **debug cls message** privileged EXEC command to display information about Cisco Link Services (CLS) messages. The **no** form of this command disables debugging output.

debug cls message

no debug cls message

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The **debug cls message** command displays the primitives (state), selector, header length, and data size.

Examples

The following is sample output from the **debug cls message** command. For example, CLS-->DLU indicates the direction of the flow that is described by the status. From CLS to DLU, a request was established to the connection end point. The header length is 48 bytes, and the data size is 104 bytes.

```
Router# debug cls message
```

```
(FRAS Daemon:CLS-->DLU):
  ID_STN.Ind to uSAP: 0x607044C4 sel: LLC hlen: 40, dlen: 54
(FRAS Daemon:CLS-->DLU):
  ID_STN.Ind to uSAP: 0x6071B054 sel: LLC hlen: 40, dlen: 46
(FRAS Daemon:DLU-->SAP):
  REQ_OPNSTN.Reg to pSAP: 0x608021F4 sel: LLC hlen: 48, dlen: 104
(FRAS Daemon:CLS-->DLU):
  REQ_OPNSTN.Cfm(NO_REMOTE_STN) to uCEP: 0x607FFE84 sel: LLC hlen: 48, dlen: 104
```

The status possibilities include the following: enabled, disabled, request open station, open station, close station, activate SA, deactivate SAP, XID, XID station, connect station, signal station, connect, disconnect, connected, data, flow, unnumbered data, modify SAP, test, activate ring, deactivate ring, test station, and unnumbered data station.

Related Commands

Command	Description
debug fras error	Displays information about FRAS protocol errors.
debug fras message	Displays general information about FRAS messages.
debug fras state	Displays information about FRAS data-link control state changes.

debug cls vdlc

Use the **debug cls vdlc** privileged EXEC command to display information about Cisco Link Services (CLS) Virtual Data Link Control (VDLC). The **no** form of this command disables debugging output.

debug cls vdlc

no debug cls vdlc

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The **debug cls message** command displays primitive state transitions, selector, and source and destination media access control (MAC) and service access points (SAPs).

Also use the **show cls** command to display additional information on CLS VDLC.



Caution

Use the **debug cls vdlc** command with caution because it can generate a significant amount of output.

Examples

The following messages are sample output from the **debug cls vdlc** command. In the following scenario, the SNA service point—also called *native service point (NSP)*—is setting up two connections through VDLC and data link switching (DLSw): one from NSP to VDLC and one from DLSw to VDLC. VDLC's task is to join the two.

The NSP initiates a connection from 4000.05d2.0001 as follows:

```
VDLC: Req Open Stn Req PSap 0x7ACE00, port 0x79DF98
      4000.05d2.0001(0C)->4000.1060.1000(04)
```

In the next message, VDLC sends a test station request to DLSw for destination address 4000.1060.1000.

```
VDLC: Send UFrame E3: 4000.05d2.0001(0C)->4000.1060.1000(00)
```

In the next two messages, DLSw replies with test station response, and NSP goes to a half-open state. NSP is waiting for the DLSw connection to VDLC.

```
VDLC: Sap to Sap TEST_STN_RSP VSap 0x7B68C0 4000.1060.1000(00)->4000.05d2.0001(0C)
VDLC: 4000.05d2.0001(0C)->4000.1060.1000(04): VDLC_OPENING->VDLC_HALF_OPEN
```

The NSP sends an exchange identification (XID) and changes state as follows:

```
VDLC: 4000.05d2.0001(0C)->4000.1060.1000(04): VDLC_HALF_OPEN->VDLC_XID_RSP_PENDING
VDLC: CEP to SAP ID_REQ 4000.05d2.0001(0C)->4000.1060.1000(04) via bridging SAP (DLSw)
```

In the next several messages, DLSw initiates its connection, which matches the half-open connection with NSP.

```
VDLC: Req Open Stn Req PSap 0x7B68C0, port 0x7992A0
      4000.1060.1000(04)->4000.05d2.0001(0C)
VDLC: two-way connection established
VDLC: 4000.1060.1000(04)->4000.05d2.0001(0C): VDLC_IDLE->VDLC_OPEN
```

In the following messages, DLSw sends an XID response, and NSP's connection goes from the state XID Response Pending to Open. The XID exchange follows:

```

V DLC: CEP to CEP ID_RSP 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_XID_RSP_PENDING->V DLC_OPEN
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_OPEN->V DLC_XID_RSP_PENDING
V DLC: CEP to CEP ID_REQ 4000.05d2.0001(0C)->4000.1060.1000(04)
V DLC: CEP to CEP ID_RSP 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_XID_RSP_PENDING->V DLC_OPEN
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_OPEN->V DLC_XID_RSP_PENDING
V DLC: CEP to CEP ID_REQ 4000.05d2.0001(0C)->4000.1060.1000(04)
V DLC: CEP to CEP ID_RSP 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_XID_RSP_PENDING->V DLC_OPEN
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_OPEN->V DLC_XID_RSP_PENDING
V DLC: CEP to CEP ID_REQ 4000.05d2.0001(0C)->4000.1060.1000(04)
V DLC: CEP to CEP ID_RSP 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_XID_RSP_PENDING->V DLC_OPEN
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_OPEN->V DLC_XID_RSP_PENDING
V DLC: CEP to CEP ID_REQ 4000.05d2.0001(0C)->4000.1060.1000(04)

```

When DLSw is ready to connect, the front-end processor (FEP) sends a set asynchronous balanced mode extended (SABME) command as follows:

```

V DLC: CEP to CEP CONNECT_REQ 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_XID_RSP_PENDING->V DLC_OPEN

```

In the following messages, NSP accepts the connection and sends an unnumbered acknowledgment (UA) to the FEP:

```

V DLC: CEP to CEP CONNECT_RSP 4000.05d2.0001(0C)->4000.1060.1000(04)
V DLC: FlowReq QUENCH OFF 4000.1060.1000(04)->4000.05d2.0001(0C)

```

The following messages show the data flow:

```

V DLC: DATA 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: DATA 4000.05d2.0001(0C)->4000.1060.1000(04)
...
V DLC: DATA 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: DATA 4000.05d2.0001(0C)->4000.1060.1000(04)

```

Related Commands

Command	Description
<code>debug cls message</code>	Displays information about CLS messages

debug compress

To debug compression, enter the **debug compress** privileged EXEC configuration command. To disable debugging output, use the **no** form of this command.

debug compress

no debug compress

Syntax Description This command has no arguments or keywords.

Defaults Disabled

Command History	Release	Modification
	10.0	This command was introduced.

Usage Guidelines Use this command to display output from the compression and decompression configuration you made. Live traffic must be configured through the Cisco 2600 with a data compression Advanced Interface Module (AIM) installed for this command to work.

Examples The following example is output from the debug compress command, which shows that compression is taking place on a Cisco 2600 Access Router using data compression Advanced Interface Module (AIM) hardware compression is configured correctly.

```
Router# debug compress
COMPRESS debugging is on
Router#compr-in:pak:0x810C6B10 npart:0 size:103
pak:0x810C6B10 start:0x02406BD4 size:103 npart:0
compr-out:pak:0x8118C8B8 stat:0x00000000 npart:1 size:71 lcb:0xED
pak:0x8118C8B8 start:0x0259CD3E size:71 npart:1
  mp:0x8118A980 start:0x0259CD3E size:71

decmp-in:pak:0x81128B78 start:0x0255AF44 size:42 npart:1 hdr:0xC035
pak:0x81128B78 start:0x0255AF44 size:42 npart:1
  mp:0x81174480 start:0x0255AF44 size:42
decmp-out:pak:0x8118C8B8 start:0x025B2C42 size:55 npart:1 stat:0
pak:0x8118C8B8 start:0x025B2C42 size:55 npart:1
  mp:0x8118B700 start:0x025B2C42 size:55
```

For a description of the debug output, see Table 30.

Table 30 *debug compress Command Field Descriptions*

Field	Description
compr-in	Indicates that a packet needs to be compressed.
compr-out	Indicates completion of compression of packet.

Table 30 *debug compress Command Field Descriptions (continued)*

Field	Description
decmp-in	Indicates receipt of a compressed packet which needs to be decompressed.
decmp-out	Indicates completion of decompression of packet.
pak:0x810c6b10	Provides the address in memory of a software structure which describes the compressed packet.
start:0x02406bd4 size:103 npart:0	The 'npart:0' indicates that the packet is contained in a single, contiguous area of memory. The start address of the packet is 0x02406bd4 and the size of the packet is 103.
start:0x0259cd3E size:71 npart:1	The npart:1 indicates that the packet is contained in 1 or more regions of memory. The start address of the packet is 0x0259cd3e and the size of the packet is 71.
mp:0x8118a980 start:0x0259cd3e size:71	Describes one of these regions of memory.
mp:0x8118a980	Provides the address of a structure describing this region.
start 0x0259cd3e	Provides the address of the start of this region.

Related Commands

Command	Description
debug frame-relay	Displays debugging information about the packets that are received on a Frame Relay interface.
debug ppp	Displays information on traffic and exchanges in an internetwork implementing the PPP.
show compress	Displays compression statistics.
show diag	Displays hardware information including DRAM, SRAM, and the revision-level information on the line card.

debug condition

To limit output for some debugging commands based on specified conditions, use the **debug condition** privileged EXEC command. The **no** form of this command removes the specified condition.

debug condition { **username** *username* | **called** *dial-string* | **caller** *dial-string* }

no debug condition { *condition-id* | **all** }

Syntax Description

username <i>username</i>	Generates debugging messages for interfaces with the specified username.
called <i>dial-string</i>	Generates debugging messages for interfaces with the called party number.
caller <i>dial-string</i>	Generates debugging messages for interfaces with the calling party number.
<i>condition-id</i>	Removes the condition indicated.
all	Removes all debugging conditions, conditions specified by the debug condition interface command. Use this keyword to disable conditional debugging and reenable debugging for all interfaces.

Defaults

All debugging messages for enabled protocol-specific **debug** commands are generated.

Usage Guidelines

Use the **debug condition** command to restrict the debug output for some commands. If any **debug condition** commands are enabled, output is only generated for interfaces associated with the specified username, called party number, or calling party number. In addition, this command enables debugging output for conditional debugging events. Messages are displayed as different interfaces meet specific conditions.

The **no** form of this command removes the debug condition specified by the condition identifier. The condition identifier is displayed after you enter a **debug condition** command or in the output of the **show debug condition** command. If the last condition is removed, debugging output resumes for all interfaces. You will be asked for confirmation before removing the last condition or all conditions.

Not all debugging output is affected by the **debug condition** command. Some commands generate output whenever they are enabled, regardless of whether or not they meet any conditions. The commands that are affected by the **debug condition** commands are generally related to dial access functions, where a large amount of output is expected. Output from the following commands is controlled by the **debug condition** command:

- **debug aaa** { **accounting** | **authorization** | **authentication** }
- **debug dialer** { **events** | **packets** }
- **debug isdn** { **q921** | **q931** }
- **debug modem** { **oob** | **trace** }
- **debug ppp** { **all** | **authentication** | **chap** | **error** | **negotiation** | **multilink events** | **packet** }

Examples

In the following example, the router only displays debugging messages for interfaces that use a username of fred. The condition identifier displayed after the command is entered identifies this particular condition.

```
Router# debug condition username fred  
Condition 1 set
```

Related Commands

Command	Description
debug condition interface	Limits output for some debugging commands based on the interfaces.

debug condition interface

To limit output for some debugging commands based on the interface, use the **debug condition interface** privileged EXEC command. The **no** form of this command removes the interface condition and resets the interface so that it must be triggered by a condition.

debug condition interface { *interface* | **all** }

no debug condition interface { *interface* | **all** }

Syntax Description

<i>interface</i>	Interface type and number.
all	All interfaces.

Defaults

All debug messages for enabled debugging commands are displayed.

Usage Guidelines

Use this command to restrict the debug output for some commands to output based on its related interface. When you enter this command, debugging output is turned off for all interfaces except the specified interface. In addition, this command enables debugging output for conditional debugging events. Messages are displayed as different interfaces meet specific conditions.

The **no** form of the command has two functions:

- It disables the **debug condition interface** command for the specified interface. Output is no longer generated for the interface, assuming that the interface meets no other conditions. If the interface meets other active conditions, as set by another **debug condition** command, debugging output will still be generated for the interface.
- The command also resets the debugging trigger on the interface. If some other **debug condition** command has been enabled, this command resets the trigger on the interface. Output is stopped for that interface until the condition is met on the interface.

You will be asked for confirmation before removing the last condition or all conditions.

Not all debugging output is affected by the **debug condition** command. Some commands generate output whenever they are enabled, regardless of whether or not they meet any conditions. The commands that are affected by the **debug condition** commands are generally related to dial access functions, where a large amount of output is expected. Output from the following commands is controlled by the **debug condition** command:

- **debug aaa** { **accounting** | **authorization** | **authentication** }
- **debug dialer** { **events** | **packets** }
- **debug isdn** { **q921** | **q931** }
- **debug modem** { **oob** | **trace** }
- **debug ppp** { **all** | **authentication** | **chap** | **error** | **negotiation** | **multilink events** | **packet** }

Examples

In this example, only **debug** command output related to serial interface 1 is displayed. The condition identifier for this command is 1.

```
Router# debug condition interface serial1  
Condition 1 set
```

Related Commands

Command	Description
debug condition	Limits output for some debugging commands based on specific conditions.

debug confmodem

Use the **debug confmodem** privileged EXEC command to display information associated with the discovery and configuration of the modem attached to the router. The **no** form of this command disables debugging output.

debug confmodem

no debug confmodem

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The **debug confmodem** command is used in debugging configurations that use the **modem autoconfig** command.

Examples

The following is sample output from the **debug confmodem** command. In the first three lines, the router is searching for a speed at which it can communicate with the modem. The remaining lines show the actual sending of the modem command.

```
Router# debug confmodem

TTY4:detection speed(115200) response -----
TTY4:detection speed(57600) response -----
TTY4:detection speed(38400) response ---OK---
TTY4:Modem command: --AT&F&C1&D2S180=3S190=1S0=1--
TTY4: Modem configuration succeeded
TTY4: Done with modem configuration
```

debug cot

Use the **debug cot** privileged EXEC command to display information about the COT functionality. The **no** form of this command disables debugging output.

debug cot {**api** | **dsp** | **queue** | **detail**}

no debug cot {**api** | **dsp** | **queue** | **detail**}

Syntax Description

api	Display information about the COT Application Program Interface (API).
dsp	Display information related to the COT/DSP interface. Typical DSP functions include: data modems, voice CODECS, fax modems and CODECs, and low-level signaling such as CAS/R2.
queue	Display information related to the COT internal queue.
detail	Display information about COT internal detail; summary of debug cot api , debug cot dsp , and debug cot queue .

Command History

Release	Modification
11.3(7)	This command was introduced.

Examples

Figure 1 shows sample **debug cot api** output.

Figure 1 Sample debug cot api Command Output

```
5300# debug cot api
COT API debugging is on
08:29:55: cot_request_handler(): CDB@0x60DEDE14, req(COT_CHECK_TONE_ON):
08:29:55:     shelf 0 slot 0 appl_no 1 ds0 1
08:29:55:     freqTX 2010 freqRX 1780 key 0xFFF1 duration 60000
```

Table 31 describes the fields in these displays.

Table 31 debug cot api Command Field Descriptions

Field	Description
CDB	Internal controller information.
req	Type of COT operation requested.
shelf	Shelf ID of COT operation request.
slot	Designate the slot number, 1 to 4.
application	Hardware unit that provides the external interface connections from a router to the network.
ds0	Number of COT operation request.
key	COT operation identifier.
duration	Timeout duration of COT operation.

Table 31 *debug cot api Command Field Descriptions (continued)*

Field	Description
freqTX	Requested transmit tone frequency.
freqRX	Requested receive tone frequency.

Figure 2 shows sample **debug cot dsp** output.

Figure 2 *Sample debug cot dsp Command Output*

```
5300# debug cot dsp
5300#
00:10:42:COT:DSP (1/1) Allocated
00:10:43:In cot_callback
00:10:43: returned key 0xFFF1, status = 0
00:10:43:COT:Received DSP Q Event
00:10:43:COT:DSP (1/1) Done
00:10:43:COT:DSP (1/1) De-allocated
```

Table 32 describes the fields in these displays.

Table 32 *debug cot dsp Command Field Descriptions*

Field	Description
DSP (1/1) Allocated	Slot and port of the DSP allocated for the COT operation.
Received DSP Q Event	Indicates the COT subsystem received an event from the DSP.
DSP (1/1) Done	Slot and port of the DSP transitioning to IDLE state.
DSP (1/1) De-allocated	Slot and port of the DSP de-allocated after the completion of the COT operation.

Figure 3 shows sample **debug cot queue** output.

Figure 3 *Sample debug cot queue Command Output*

```
5300# debug cot queue
5300#
00:11:26:COT(0x60EBB48C):Adding new request (0x61123DBC) to In
Progress Q
00:11:26:COT(0x60EBB48C):Adding COT(0x61123DBC) to the Q head
00:11:27:In cot_callback
00:11:27: returned key 0xFFF1, status = 0
```

Table 33 describes the fields in these displays.

Table 33 *debug cot api Command Field Descriptions*

Field	Description
COT	Internal COT operation request.
Adding new request	Internal COT operation request queue.

Figure 4 shows sample **debug cot detail** output.

Figure 4 *Sample debug cot detail Command Output*

```
5300# debug cot detail
5300#
00:04:57:cot_request_handler():CDB@0x60EBB48C, req(COT_CHECK_TONE_ON):

00:04:57:    shelf 0 slot 0 appl_no 1 ds0 1
00:04:57:    freqTX 1780 freqRX 2010 key 0xFFFF1 duration 1000

00:04:57:COT:DSP (1/0) Allocated
00:04:57:COT:Request Transition to COT_WAIT_TD_ON
00:04:57:COT(0x60EBB48C):Adding new request (0x61123DBC) to In
Progress Q
00:04:57:COT(0x60EBB48C):Adding COT(0x61123DBC) to the Q head
00:04:57:COT:Start Duration Timer for Check Tone Request
00:04:58:COT:Received Timer Event
00:04:58:COT:T24 Timer Expired
00:04:58:COT Request@ 0x61123DBC, CDB@ 0x60EBB48C, Params@0x61123E08
00:04:58: request type = COT_CHECK_TONE_ON
00:04:58: shelf 0 slot 0 appl_no 1 ds0 1
00:04:58: duration 1000 key FFF1 freqTx 1780 freqRx 2010
00:04:58: state COT_WAIT_TD_ON_CT
00:04:58: event_proc(0x6093B55C)

00:04:58:Invoke NI2 callback to inform COT request status
00:04:58:In cot_callback
00:04:58: returned key 0xFFFF1, status = 0
00:04:58:Return from NI2 callback
00:04:58:COT:Request Transition to IDLE
00:04:58:COT:Received DSP Q Event
00:04:58:COT:DSP (1/0) Done
00:04:58:COT:DSP (1/0) De-allocated
```

Because **debug cot detail** is a summary of **debug cot api**, **debug cot dsp**, and **debug cot queue**, the field descriptions are the same.

debug cpp event

Use the **debug cpp event** privileged EXEC command to display general Combinet Proprietary Protocol (CPP) events. The **no** form of this command disables debugging output.

debug cpp event

no debug cpp event

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The CPP protocol allows a router to engage in negotiation over an ISDN B channel to establish connections with a Combinet bridge.

The **debug cpp event** command displays events such as CPP sequencing, group creation, and keepalives.

Examples

One or more of the messages in Table 34 appear when you use the **debug cpp event** command. Each message begins with the short name of the interface the event occurred on (for example, SERIAL0:1 or BRI0:1) and might contain one or more packet sequence numbers or remote site names.

Table 34 *debug cpp event Command Messages*

Message	Description
BRI0:1: negotiation complete	Call was set up on the interface (in this example, BRI0:1).
BRI0:1: negotiation timed out	Call timed out.
BRI0:1: sending negotiation packet	Negotiation packet was sent to set up the call.
BRI0:1: out of sequence packet - got 10, range 1 8	Packet was received that was out of sequence. The first number displayed in the message is the sequence number received and the following numbers are the range of valid sequence numbers.
BRI0:1: Sequence timer expired - Lost 11 Trying sequence 12	Timer expired before the packet was received. The first number displayed in the message is the sequence number of the packet that was lost, and the second number is the next sequence number.
BRI0:1: Line Integrity Violation	Router fails to maintain keepalives.
BRI0:1: create cpp group ber19 destroyed cpp group ber19	Dialer group is created on the remote site (in this example, <i>ber19</i>).

Related Commands

Command	Description
debug cpp negotiation	Displays CPP negotiation events.
debug cpp packet	Displays CPP packets.

debug cpp negotiation

Use the **debug cpp negotiation** privileged EXEC command to display Combinet Proprietary Protocol (CPP) negotiation events. The **no** form of this command disables debugging output.

debug cpp negotiation

no debug cpp negotiation

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The CPP protocol allows a router to engage in negotiation over an ISDN B channel to establish connections with a Combinet bridge.

The **debug cpp negotiation** command displays events such as the type of packet and packet size being sent.

Examples

The following is sample output from the **debug cpp negotiation** command. In this example, a sample connection is shown.

```
Router# debug cpp negotiation

%LINK-3-UPDOWN: Interface BRI0: B-Channel 2, changed state to down
%LINK-3-UPDOWN: Interface BRI0, changed state to up
%SYS-5-CONFIG_I: Configured from console by console
%LINK-3-UPDOWN: Interface BRI0: B-Channel 1, changed state to up
BR0:1:(I) NEG packet - len 77
  attempting proto:2
  ether id:0040.f902.c7b4
  port 1 number:5559876
  port 2 number:5559876
  origination port:1
  remote name:ber19
  password is correct
```

Table 35 describes the fields and messages in the output.

Table 35 Debug CPP Negotiation Field Descriptions

Field	Description
BR0:1 (I) NEG packet - len 77	Interface name, packet type, and packet size.
attempting proto:	CPP protocol type.
ether id:	Ethernet address of the destination router.
port 1 number:	ISDN phone number of remote B channel #1.
port 2 number:	ISDN phone number of remote B channel #2.
origination port:	B channel 1 or 2 called.
remote name:	Remote site name to which this call is connecting.
password is correct	Password is accepted so the connection is established.

■ **debug cpp negotiation**

Related Commands	Command	Description
	debug cot	Displays information about the COT functionality.
	debug cpp packet	Displays CPP packets.

debug cpp packet

Use the **debug cpp packet** privileged EXEC command to display Combinet Proprietary Protocol (CPP) packets. The **no** form of this command disables debugging output.

debug cpp packet

no debug cpp packet

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The CPP protocol allows a router to engage in negotiation over an ISDN B channel to establish connections with a Combinet bridge.

The **debug cpp packet** command displays the hexadecimal values of the packets.

Examples

The following is sample output from the **debug cpp packet** command. This example shows the interface name, packet type, packet size, and the hexadecimal values of the packet.

```
Router# debug cpp packet

BR0:1:input packet - len 60
00 00 00 00 00 00 00 40 F9 02 C7 B4 08 0. !6 00 01
08 00 06 04 00 02 00 40 F9 02 C7 B4 83 6C A1 02!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 64/66/68 ms
BR0:1 output packet - len 116
06 00 00 40 F9 02 C7 B4 00 00 0C 3E 12 3A 08 00
45 00 00 64 00 01 00 00 FF 01 72 BB 83 6C A1 01
```

Related Commands

Command	Description
debug cot	Displays information about the COT functionality.
debug cpp negotiation	Displays CPP negotiation events.

debug crypto engine

To display debug messages about crypto engines, which perform encryption and decryption, use the **debug crypto engine** privileged EXEC command. To disable debugging output, use the **no** form of this command.

debug crypto engine

no debug crypto engine

Syntax Description This command has no arguments or keywords.

Defaults Disabled.

Command History	Release	Modification
	12.0	This command was introduced.

Usage Guidelines Use the **debug crypto engine** command to display information pertaining to the crypto engine, such as when Cisco IOS software is performing encryption or decryption operations.

The crypto engine is the actual mechanism that performs encryption and decryption. A crypto engine can be software or a hardware accelerator. Some platforms can have multiple crypto engines; therefore, the router will have multiple hardware accelerators.

Examples The following is sample output from the **debug crypto engine** command. The first sample output shows messages from a router that successfully generates RSA keys. The second sample output shows messages from a router that decrypts the RSA key during Internet Key Exchange (IKE) negotiation.

```
pki-36a# debug crypto engine

00:25:13:CryptoEngine0:generate key pair
00:25:13:CryptoEngine0:CRYPTO_GEN_KEY_PAIR
00:25:13:CRYPTO_ENGINE:key process suspended and continued
00:25:14:CRYPTO_ENGINE:key process suspended and continuedcr

pki-36a# debug crypto engine

00:27:45:%SYS-5-CONFIG_I:Configured from console by console
00:27:51:CryptoEngine0:generate alg parameter
00:27:51:CRYPTO_ENGINE:Dh phase 1 status:0
00:27:51:CRYPTO_ENGINE:Dh phase 1 status:0
00:27:51:CryptoEngine0:generate alg parameter
00:27:52:CryptoEngine0:calculate pkey hmac for conn id 0
00:27:52:CryptoEngine0:create ISAKMP SKEYID for conn id 1
00:27:52:Crypto engine 0:RSA decrypt with public key
00:27:52:CryptoEngine0:CRYPTO_RSA_PUB_DECRYPT
00:27:52:CryptoEngine0:generate hmac context for conn id 1
00:27:52:CryptoEngine0:generate hmac context for conn id 1
00:27:52:Crypto engine 0:RSA encrypt with private key
```

```
00:27:52:CryptoEngine0:CRYPTO_RSA_PRIV_ENCRYPT
00:27:53:CryptoEngine0:clear dh number for conn id 1
00:27:53:CryptoEngine0:generate hmac context for conn id 1
00:27:53:validate proposal 0
00:27:53:validate proposal request 0
00:27:54:CryptoEngine0:generate hmac context for conn id 1
00:27:54:CryptoEngine0:generate hmac context for conn id 1
00:27:54:ipsec allocate flow 0
00:27:54:ipsec allocate flow 0
```

Related Commands

Command	Description
crypto key generate rsa	Generates RSA key pairs.

debug crypto ipsec

Use the **debug crypto ipsec** privileged EXEC command to display IPsec events. The **no** form of this command disables debugging output.

debug crypto ipsec

no debug crypto ipsec

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the **debug crypto ipsec** command. In this example, security associations (SAs) have been successfully established.

```
Router# debug crypto ipsec
```

IPsec requests SAs between 172.21.114.123 and 172.21.114.67, on behalf of the **permit ip host 172.21.114.123 host 172.21.114.67** command. It prefers to use the transform set esp-des w/esp-md5-hmac, but it will also consider ah-sha-hmac.

```
00:24:30: IPSEC(sa_request): ,
(key eng. msg.) src= 172.21.114.123, dest= 172.21.114.67,
src_proxy= 172.21.114.123/255.255.255.255/0/0 (type=1),
dest_proxy= 172.21.114.67/255.255.255.255/0/0 (type=1),
protocol= ESP, transform= esp-des esp-md5-hmac ,
lifedur= 120s and 4608000kb,
spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4
00:24:30: IPSEC(sa_request): ,
(key eng. msg.) src= 172.21.114.123, dest= 172.21.114.67,
src_proxy= 172.21.114.123/255.255.255.255/0/0 (type=1),
dest_proxy= 172.21.114.67/255.255.255.255/0/0 (type=1).,
protocol= AH, transform= ah-sha-hmac ,
lifedur= 120s and 4608000kb,
spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x0.
```

IKE asks for SPIs from IPsec. For inbound security associations, IPsec controls its own SPI space.

```
00:24:34: IPSEC(key_engine): got a queue event...
00:24:34: IPSEC spi_response): getting spi 3029740121d for SA
from 172.21.114.67 to 172.21.114.123 for prot 3
00:24:34: IPSEC spi_response): getting spi 5250759401d for SA
from 172.21.114.67 to 172.21.114.123 for prot 2
```

IKE will ask IPsec if it accepts the SA proposal. In this case, it will be the one sent by the local IPsec in the first place:

```
00:24:34: IPSEC(validate_proposal_request): proposal part #1,
(key eng. msg.) dest= 172.21.114.67, src= 172.21.114.123,
dest_proxy= 172.21.114.67/255.255.255.255/0/0 (type=1),
src_proxy= 172.21.114.123/255.255.255.255/0/0 (type=1),
protocol= ESP, transform= esp-des esp-md5-hmac ,
lifedur= 0s and 0kb,
spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4
```

After the proposal is accepted, IKE finishes the negotiations, generates the keying material, and then notifies IPSec of the new security associations (one security association for each direction).

```
00:24:35: IPSEC(key_engine): got a queue event...
```

The following output pertains to the inbound SA. The `conn_id` value references an entry in the crypto engine connection table.

```
00:24:35: IPSEC(initialize_sas): ,
(key eng. msg.) dest= 172.21.114.123, src= 172.21.114.67,
dest_proxy= 172.21.114.123/255.255.255.255/0/0 (type=1),
src_proxy= 172.21.114.67/255.255.255.255/0/0 (type=1),
protocol= ESP, transform= esp-des esp-md5-hmac ,
lifedur= 120s and 4608000 kb,
spi= 0x120F043C(302974012), conn_id= 29, keysize= 0, flags= 0x4
```

The following output pertains to the outbound SA:

```
00:24:35: IPSEC(initialize_sas): ,
(key eng. msg.) src= 172.21.114.123, dest= 172.21.114.67,
src_proxy= 172.21.114.123/255.255.255.255/0/0 (type=1),
dest_proxy= 172.21.114.67/255.255.255.255/0/0 (type=1),
protocol= ESP, transform= esp-des esp-md5-hmac ,
lifedur= 120s and 4608000kb,
spi= 0x38914A4(59315364), conn_id= 30, keysize= 0, flags= 0x4
```

IPSec now installs the security association information into its security association database.

```
00:24:35: IPSEC(create_sa): sa created,
(sa) sa_dest= 172.21.114.123, sa_prot= 50,
sa_spi= 0x120F043C(302974012),
sa_trans= esp-des esp-md5-hmac , sa_conn_id= 29
00:24:35: IPSEC(create_sa): sa created,
(sa) sa_dest= 172.21.114.67, sa_prot= 50,
sa_spi= 0x38914A4(59315364),
sa_trans= esp-des esp-md5-hmac , sa_conn_id= 30
```

The following is sample output for the **debug crypto ipsec** command as seen on the peer router. In this example, IKE asks IPSec if it will accept an SA proposal. Although the peer sent two proposals, IPSec accepted the first proposal.

```
00:26:15: IPSEC(validate_proposal_request): proposal part #1,
(key eng. msg.) dest= 172.21.114.67, src= 172.21.114.123,
dest_proxy= 172.21.114.67/255.255.255.255/0/0 (type=1),
src_proxy= 172.21.114.123/255.255.255.255/0/0 (type=1),
protocol= ESP, transform= esp-des esp-md5-hmac ,
lifedur= 0s and 0kb,
spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4
```

IKE asks for SPIs.

```
00:26:15: IPSEC(key_engine): got a queue event...
00:26:15: IPSEC(spi_response): getting spi 593153641d for SA
from 172.21.114.123 to 172.21.114.67 for prot 3
```

IKE does the remaining processing, completing the negotiation and generating keys. It then tells IPSec about the new SAs.

```
00:26:15: IPSEC(key_engine): got a queue event...
```

The following output pertains to the inbound SA:

```
00:26:15: IPSEC(initialize_sas): ,
(key eng. msg.) dest= 172.21.114.67, src= 172.21.114.123,
  dest_proxy= 172.21.114.67/0.0.0.0/0/0 (type=1),
  src_proxy= 172.21.114.123/0.0.0.0/0/0 (type=1),
  protocol= ESP, transform= esp-des esp-md5-hmac ,
  lifedur= 120s and 4608000kb,
  spi= 0x38914A4(59315364), conn_id= 25, keysize= 0, flags= 0x4
```

The following output pertains to the outbound SA:

```
00:26:15: IPSEC(initialize_sas): ,
(key eng. msg.) src= 172.21.114.67, dest= 172.21.114.123,
  src_proxy= 172.21.114.67/0.0.0.0/0/0 (type=1),
  dest_proxy= 172.21.114.123/0.0.0.0/0/0 (type=1),
  protocol= ESP, transform= esp-des esp-md5-hmac ,
  lifedur= 120s and 4608000kb,
  spi= 0x120F043C(302974012), conn_id= 26, keysize= 0, flags= 0x4
```

IPSec now installs the security association information into its security association database:

```
00:26:15: IPSEC(create_sa): sa created,
(sa) sa_dest= 172.21.114.67, sa_prot= 50,
  sa_spi= 0x38914A4(59315364),
  sa_trans= esp-des esp-md5-hmac , sa_conn_id= 25
00:26:15: IPSEC(create_sa): sa created,
(sa) sa_dest= 172.21.114.123, sa_prot= 50,
  sa_spi= 0x120F043C(302974012),
  sa_trans= esp-des esp-md5-hmac , sa_conn_id= 26
```

debug crypto isakmp

Use the **debug crypto isakmp** privileged EXEC command to display messages about IKE events. The **no** form of this command disables debugging output.

debug crypto isakmp

no debug crypto isakmp

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the **debug crypto isakmp** command for an IKE peer that initiates an IKE negotiation:

First, IKE negotiates its own security association (SA), checking for a matching IKE policy:

```
MyRouter# debug crypto isakmp
20:26:58: ISAKMP (8): beginning Main Mode exchange
20:26:58: ISAKMP (8): processing SA payload. message ID = 0
20:26:58: ISAKMP (8): Checking ISAKMP transform 1 against priority 10 policy
20:26:58: ISAKMP:      encryption DES-CBC
20:26:58: ISAKMP:      hash SHA
20:26:58: ISAKMP:      default group 1
20:26:58: ISAKMP:      auth pre-share
20:26:58: ISAKMP (8): atts are acceptable. Next payload is 0
```

IKE has found a matching policy. Next, the IKE SA is used by each peer to authenticate the other peer:

```
20:26:58: ISAKMP (8): SA is doing pre-shared key authentication
20:26:59: ISAKMP (8): processing KE payload. message ID = 0
20:26:59: ISAKMP (8): processing NONCE payload. message ID = 0
20:26:59: ISAKMP (8): SKEYID state generated
20:26:59: ISAKMP (8): processing ID payload. message ID = 0
20:26:59: ISAKMP (8): processing HASH payload. message ID = 0
20:26:59: ISAKMP (8): SA has been authenticated
```

Next, IKE negotiates to set up the IPsec SA by searching for a matching transform set:

```
20:26:59: ISAKMP (8): beginning Quick Mode exchange, M-ID of 767162845
20:26:59: ISAKMP (8): processing SA payload. message ID = 767162845
20:26:59: ISAKMP (8): Checking IPsec proposal 1
20:26:59: ISAKMP: transform 1, ESP_DES
20:26:59: ISAKMP:   attributes in transform:
20:26:59: ISAKMP:     encaps is 1
20:26:59: ISAKMP:     SA life type in seconds
20:26:59: ISAKMP:     SA life duration (basic) of 600
20:26:59: ISAKMP:     SA life type in kilobytes
20:26:59: ISAKMP:     SA life duration (VPI) of
20:26:59: ISAKMP:       0x0 0x46 0x50 0x0
20:26:59: ISAKMP:     authenticator is HMAC-MD5
20:26:59: ISAKMP (8): atts are acceptable.
```

A matching IPsec transform set has been found at the two peers. Now the IPsec SA can be created (one SA is created for each direction):

```
20:26:59: ISAKMP (8): processing NONCE payload. message ID = 767162845
20:26:59: ISAKMP (8): processing ID payload. message ID = 767162845
20:26:59: ISAKMP (8): processing ID payload. message ID = 767162845
20:26:59: ISAKMP (8): Creating IPsec SAs
```

```
20:26:59:      inbound SA from 155.0.0.2 to 155.0.0.1 (proxy 155.0.0.2 to 155.0.0.1
)
20:26:59:      has spi 454886490 and conn_id 9 and flags 4
20:26:59:      lifetime of 600 seconds
20:26:59:      lifetime of 4608000 kilobytes
20:26:59:      outbound SA from 155.0.0.1      to 155.0.0.2      (proxy 155.0.0.1
to 155.0.0.2
)
20:26:59:      has spi 75506225 and conn_id 10 and flags 4
20:26:59:      lifetime of 600 seconds
20:26:59:      lifetime of 4608000 kilobytes
```

debug crypto key-exchange

Use the **debug crypto key-exchange** privileged EXEC command to show Digital Signature Standard (DSS) public key exchange messages. The **no** form of this command disables debugging output.

debug crypto key-exchange

no debug crypto key-exchange

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

Encryption and authentication are provided by a software service on the router called a *crypto engine*. The crypto engine performs authentication through DSS public and private keys when a connection is set up. DSS is a means of sending a “signature” at the end of a message that positively identifies the author of the message. The signature cannot be forged or duplicated by others, so whoever received a message with a DSS signature knows exactly who sent the message.

If the process of exchanging DSS public keys with a peer router by means of the **config crypto key-exchange** command is not successful, try to exchange DSS public keys again after enabling the **debug crypto key-exchange** command to help you diagnose the problem.

Examples

The following is sample output from the **debug crypto key-exchange** command. The first shows output from the initiating router in a key exchange. The second shows output from the passive router in a key exchange. The number of bytes received should match the number of bytes sent from the initiating side, although the number of messages can be different.

```
Router# debug crypto key-exchange
```

```
CRYPTO-KE: Sent 4 bytes.
CRYPTO-KE: Sent 2 bytes.
CRYPTO-KE: Sent 2 bytes.
CRYPTO-KE: Sent 2 bytes.
CRYPTO-KE: Sent 64 bytes.
```

```
Router# debug crypto key-exchange
```

```
CRYPTO-KE: Received 4 bytes.
CRYPTO-KE: Received 2 bytes.
CRYPTO-KE: Received 2 bytes.
CRYPTO-KE: Received 2 bytes.
CRYPTO-KE: Received 49 bytes.
CRYPTO-KE: Received 15 bytes.
```

Related Commands

Command	Description
debug crypto sesgmt	Shows connection setup messages and their flow through the router.

debug crypto pki messages

To display debug messages for the details of the interaction (message dump) between the certification authority (CA) and the router, use the **debug crypto pki messages** privileged EXEC command. To disable debugging output, use the **no** form of this command.

```
debug crypto pki messages
```

```
debug crypto pki messages
```

Syntax Description This command has no arguments or keywords.

Defaults Disabled.

Command History	Release	Modification
	12.0	This command was introduced.

Usage Guidelines Use the **debug crypto pki messages** command to display messages about the actual data being sent and received during Public Key Infrastructure (PKI) transactions.

You can also use the **show crypto ca certificates** command to display information about your certificate.

Examples The following example is sample output for the **debug crypto pki messages** command:

```
pki-36a# debug crypto pki messages

Fingerprint: 2CFC6265 77BA6496 3AEFCB50 29BC2BF2
00:48:23:Write out pkcs#10 content:274
00:48:23:30 82 01 0E 30 81 B9 02 01 00 30 22 31 20 30 1E 06 09 2A 86
00:48:23:48 86 F7 0D 01 09 02 16 11 70 6B 69 2D 33 36 61 2E 63 69 73
00:48:23:63 6F 2E 63 6F 6D 30 5C 30 0D 06 09 2A 86 48 86 F7 0D 01 01
00:48:23:01 05 00 03 4B 00 30 48 02 41 00 DD 2C C6 35 A5 3F 0F 97 6C
00:48:23:11 E2 81 95 01 6A 80 34 25 10 C4 5F 3D 8B 33 1C 19 50 FD 91
00:48:23:6C 2D 65 4C B6 A6 B0 02 1C B2 84 C1 C8 AC A4 28 6E EF 9D 3B
00:48:23:30 98 CB 36 A2 47 4E 7E 6F C9 3E B8 26 BE 15 02 03 01 00 01
00:48:23:A0 32 30 10 06 09 2A 86 48 86 F7 0D 01 09 07 31 03 13 01 63
00:48:23:30 1E 06 09 2A 86 48 86 F7 0D 01 09 0E 31 11 14 0F 30 0D 30
00:48:23:0B 06 03 55 1D 0F 04 04 03 02 05 A0 30 0D 06 09 2A 86 48 86
00:48:23:F7 0D 01 01 04 05 00 03 41 00 2C FD 88 2C 8A 13 B6 81 88 EA
00:48:23:5C FD AE 52 8F 2C 13 95 9E 9D 8B A4 C9 48 32 84 BF 05 03 49
00:48:23:63 27 A3 AC 6D 74 EB 69 E3 06 E9 E4 9F 0A A8 FB 20 F0 02 03
00:48:23:BE 90 57 02 F2 75 8E 0F 16 60 10 6F BE 2B
00:48:23:Enveloped Data ...

00:48:23:30 80 06 09 2A 86 48 86 F7 0D 01 07 03 A0 80 30 80 02 01 00
00:48:23:31 80 30 82 01 0F 02 01 00 30 78 30 6A 31 0B 30 09 06 03 55
00:48:23:04 06 13 02 55 53 31 0B 30 09 06 03 55 04 08 13 02 43 41 31
00:48:23:13 30 11 06 03 55 04 07 13 0A 53 61 6E 74 61 20 43 72 75 7A
00:48:23:31 15 30 13 06 03 55 04 0A 13 0C 43 69 73 63 6F 20 53 79 73
```

```

00:48:23:74 65 6D 31 0E 30 0C 06 03 55 04 0B 13 05 49 50 49 53 55 31
00:48:23:Signed Data 1382 bytes
00:48:23:30 80 06 09 2A 86 48 86 F7 0D 01 07 02 A0 80 30 80 02 01 01
00:48:23:31 0E 30 0C 06 08 2A 86 48 86 F7 0D 02 05 05 00 30 80 06 09
00:48:23:2A 86 48 86 F7 0D 01 07 01 A0 80 24 80 04 82 02 75 30 80 06
00:48:23:02 55 53 31 0B 30 09 06 03 55 04 08 13 02 43 41 31 13 30 11
00:48:23:33 34 5A 17 0D 31 30 31 31 31 35 31 38 35 34 33 34 5A 30 22
00:48:23:31 20 30 1E 06 09 2A 86 48 86 F7 0D 01 09 02 16 11 70 6B 69
00:48:23:2D 33 36 61 2E 63 69 73 63 6F 2E 63 6F 6D 30 5C 30 0D 06 09
00:48:23:2A 86 48 86 F7 0D 01 01 01 05 00 03 4B 00 30 48 02 41 00 DD
00:48:23:2C C6 35 A5 3F 0F 97 6C 11 E2 81 95 01 6A 80 34 25 10 C4 5F
00:48:23:3D 8B 33 1C 19 50 FD 91 6C 2D 65 4C B6 A6 B0 02 1C B2 84 C1
00:48:23:86 F7 0D 01 01 01 05 00 04 40 C6 24 36 D6 D5 A6 92 80 5D E5
00:48:23:15 F7 3E 15 6D 71 E1 D0 13 2B 14 64 1B 0C 0F 96 BF F9 2E 05
00:48:23:EF C2 D6 CB 91 39 19 F8 44 68 0E C5 B5 84 18 8B 2D A4 B1 CD
00:48:23:3F EC C6 04 A5 D9 7C B1 56 47 3F 5B D4 93 00 00 00 00 00
00:48:23:00 00
00:48:24:Received pki message:1778 types
00:48:24:30 82 06 EE 06 09 2A 86 48 86 F7 0D 01 07 02 A0 82 06 DF 30
00:48:24:82 06 DB 02 01 01 31 0E 30 0C 06 08 2A 86 48 86 F7 0D 02 05
00:48:24:05 00 30 82 04 C5 06 09 2A 86 48 86 F7 0D 01 07 01 A0 82 04
00:48:24:B6 04 82 04 B2 30 82 04 AE 06 09 2A 86 48 86 F7 0D 01 07 03
00:48:24:0E 61 85 48 B1 DA 3D 73 F1 4B D8 5E 03 6E F3 E5 72 5D D7 17
00:48:24:17 3D 03 19 B3 8F 06 8B FE FB B1 CE D4 4C 4D 1B 81 CF 59 B7
00:48:24:78 DD 27 BA 28 2F 85 09 F0 61 74 0F 0F 92 F0 C8 C7 5B 96 E7
00:48:24:71 AF 87 D2 72 75 B7 F7 89 6F E4 E7 57 84 76 53 0B 50 8A B9
00:48:24:05 54 6F 06 75 72 8A AF 54 A6 EF 70 2D 15 6C B7 30 91 1C 00
00:48:24:CB 26 80 8D DC 89 77 57 1E D5 7A 37 86 BE 44 F8 66 60
00:48:24:Verified signed data 1202 bytes:
00:48:24:30 82 04 AE 06 09 2A 86 48 86 F7 0D 01 07 03 A0 82 04 9F 30
00:48:24:82 04 9B 02 01 00 31 81 9F 30 81 9C 02 01 00 30 46 30 22 31
00:48:24:20 30 1E 06 09 2A 86 48 86 F7 0D 01 09 02 16 11 70 6B 69 2D
00:48:24:33 36 61 2E 63 69 73 63 6F 2E 63 6F 6D 02 20 34 45 45 41 44
00:48:24:E2 55 65 DE DB 23 91 D7 60 53 96 64 BE F2 30 A7 8B 1B D9 EB
00:48:24:2E EB 9B 0D 75 EC 8E AF C0 9C 62 78 29 E0 97 00 EA 84 80 DD
00:48:24:AB 83 32 89 3E 5B A9 9F A9 9A 6D 3A 87 E2 71 16 C9 C1 E4 DB
00:48:24:FA 5A FC F3 31 98 2B 8E 55 71 C4 F6 BF CE 45 CA A5 47 40 9B
00:48:24:19 E3 1A C3 F5 ED 4D 81 1F 6F 34 35 E2 00 B3 93 DD A0 6A 74
00:48:24:EA 2B A8 D4 32 53 A7 86 50 71 5E 2A 64 BE 4B B1 72 AB 8C DA
00:48:24:AB 7A 2A 07 C0 7E C1 A7 12 31 33 AB 94 E0 3B A2 68 17 DE CE
00:48:24:57 70 2D 0B F5 C8 A7 FC FE 40 74 E8 EB 9C 82 77 DE A4 FA 75
00:48:24:FF 6F 7B E6 74 E2 F5 A1 9A C8 3C 23 DB 4A 90 BE 4A 94 EB 8B
00:48:24:ED F3
00:48:24:Decrypted enveloped content:
00:48:24:30 82 03 C8 06 09 2A 86 48 86 F7 0D 01 07 02 A0 82 03 B9 30
00:48:24:82 03 B5 02 01 01 31 00 30 0B 06 09 2A 86 48 86 F7 0D 01 07
00:48:24:01 A0 82 03 9D 30 82 03 99 30 82 03 43 A0 03 02 01 02 02 0A
00:48:24:70 45 B3 F6 00 00 00 00 01 23 30 0D 06 09 2A 86 48 86 F7 0D
00:48:24:35 35 32 32 5A 30 22 31 20 30 1E 06 09 2A 86 48 86 F7 0D 01
00:48:24:09 02 13 11 70 6B 69 2D 33 36 61 2E 63 69 73 63 6F 2E 63 6F
00:48:24:6D 30 5C 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00 03 4B
00:48:24:00 30 48 02 41 00 DD 2C C6 35 A5 3F 0F 97 6C 11 E2 81 95 01
00:48:24:6A 80 34 25 10 C4 5F 3D 8B 33 1C 19 50 FD 91 6C 2D 65 4C B6
00:48:24:63 6F 2E 63 6F 6D 2F 43 65 72 74 45 6E 72 6F 6C 6C 2F 6D 73
00:48:24:63 61 2D 72 6F 6F 74 5F 6D 73 63 61 2D 72 6F 6F 74 2E 63 72
00:48:24:74 30 41 06 08 2B 06 01 05 05 07 30 02 86 35 66 69 6C 65 3A
00:48:24:2F 2F 5C 5C 6D 73 63 61 2D 72 6F 6F 74 5C 43 65 72 74 45 6E
00:48:24:72 6F 6C 6C 5C 6D 73 63 61 2D 72 6F 6F 74 5F 6D 73 63 61 2D
00:48:24:72 6F 6F 74 2E 63 72 74 30 0D 06 09 2A 86 48 86 F7 0D 01 01
00:48:24:05 05 00 03 41 00 56 30 AD 99 1F FA 0D 1A C3 3D 71 2A DB A0
00:48:24:48 C5 EB C8 D4 FE 62 49 9C 69 5D E4 80 77 19 3E 07 B8 2B 4F
00:48:24:9A D7 72 A7 26 25 61 AE 5B 1C B5 7B 4C 18 CA 17 C3 D0 76 84
00:48:24:75 41 92 74 5E A4 E8 9E 09 60 31 00
00:48:24:%CRYPTO-6-CERTRET:Certificate received from Certificate Authority

```

Related Commands

Command	Description
crypto ca enroll	Obtains the certificate of your router from the CA.
debug crypto pki transactions	Displays debug messages for the trace of interaction (message type) between the CA and the router.
show crypto ca certificates	Displays information about your certificate, the certificate of the CA, and any RA certificates.

debug crypto sesmgmt

Use the **debug crypto sesmgmt** privileged EXEC command to show connection setup messages and their flow through the router. The **no** form of this command disables debugging output.

debug crypto sesmgmt

no debug crypto sesmgmt

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

Encryption and authentication are provided by a software service on the router called a *crypto engine*. The crypto engine performs authentication through DSS public and private keys when a connection is set up. DSS is a means of sending a “signature” at the end of a message that positively identifies the author of the message. The signature cannot be forged or duplicated by others, so whoever receives a message with a DSS signature knows exactly who sent the message.

When connections are not completing, use the **debug crypto sesmgmt** command to follow the progress of connection messages as a first step in diagnosing the problem. You see a record of each connection message as the router discovers it, and can track its progress through the necessary signing, verifying, and encryption session setup operations. Other significant connection setup events, such as the pregeneration of Diffie-Hellman public numbers, are also shown. For information on Diffie-Hellman public numbers, refer to the *Security Configuration Guide*.

Also use the **show crypto connections** command to display additional information on connections.

Examples

The following is sample output from the **debug crypto sesmgmt** command. The first shows messages from a router that initiates a successful connection. The second shows messages from a router that receives a connection.

```
Router# debug crypto sesmgmt

CRYPTO: Dequeued a message: Inititate_Connection
CRYPTO: DH gen phase 1 status for conn_id 2 slot 0:OK
CRYPTO: Signing done. Status:OK
CRYPTO: ICMP message sent: s=172.21.114.163, d=172.21.114.162
CRYPTO-SDU: send_nnc_req: NNC Echo Request sent
CRYPTO: Dequeued a message: CRM
CRYPTO: DH gen phase 2 status for conn_id 2 slot 0:OK
CRYPTO: Verify done. Status=OK
CRYPTO: Signing done. Status:OK
CRYPTO: ICMP message sent: s=172.21.114.163, d=172.21.114.162
CRYPTO-SDU: recv_nnc_rpy: NNC Echo Confirm sent
CRYPTO: Create encryption key for conn_id 2 slot 0:OK
CRYPTO: Replacing -2 in crypto maps with 2 (slot 0)

Router# debug crypto sesmgmt

CRYPTO: Dequeued a message: CIM
CRYPTO: Verify done. Status=OK
CRYPTO: DH gen phase 1 status for conn_id 1 slot 0:OK
```

■ **debug crypto sesgmt**

```
CRYPTO: DH gen phase 2 status for conn_id 1 slot 0:OK  
CRYPTO: Signing done. Status:OK  
CRYPTO: ICMP message sent: s=172.21.114.162, d=172.21.114.163  
CRYPTO-SDU: act_on_nnc_req: NNC Echo Reply sent  
CRYPTO: Create encryption key for conn_id 1 slot 0:OK  
CRYPTO: Replacing -2 in crypto maps with 1 (slot 0)  
CRYPTO: Dequeued a message: CCM  
CRYPTO: Verify done. Status=OK
```

Related Commands

Command	Description
debug crypto key-exchange	Shows DSS public key exchange messages.

debug crypto pki transactions

To display debug messages for the trace of interaction (message type) between the certification authority (CA) and the router, use the **debug crypto pki transactions** privileged EXEC command. To disable debugging output, use the **no** form of this command.

debug crypto pki transactions

no debug crypto pki transaction

Syntax Description This command has no arguments or keywords.

Defaults Disabled.

Command History	Release	Modification
	12.0	This command was introduced.

Usage Guidelines Use the **debug crypto pki transactions** command to display debug messages pertaining to Public Key Infrastructure (PKI) certificates. The messages will show status information during certificate enrollment and verification.

You can also use the **show crypto ca certificates** command to display information about your certificate.

Examples The following example, which authenticates and enrolls a CA, contains sample output for the **debug crypto pki transactions** command:

```
pki-36a(config)# crypto ca authenticate msca
Certificate has the following attributes:
Fingerprint:A5DE3C51 AD8B0207 B60BED6D 9356FB00
% Do you accept this certificate? [yes/no]:y

pki-36a# debug crypto pki transactions

00:44:00:CRYPTO_PKI:Sending CA Certificate Request:
GET /certsrv/mscep/mscep.dll/pkiclient.exe?operation=GetCACert&message=msca HTTP/1.0

00:44:00:CRYPTO_PKI:http connection opened
00:44:01:CRYPTO_PKI:HTTP response header:
HTTP/1.1 200 OK
Server:Microsoft-IIS/5.0
Date:Fri, 17 Nov 2000 18:50:59 GMT
Content-Length:2693
Content-Type:application/x-x509-ca-ra-cert

Content-Type indicates we have received CA and RA certificates.

00:44:01:CRYPTO_PKI:WARNING:A certificate chain could not be constructed while selecting
certificate status
```

```

00:44:01:CRYPTO_PKI:WARNING:A certificate chain could not be constructed while selecting
certificate status

00:44:01:CRYPTO_PKI:Name:CN = msca-rootRA, O = Cisco System, C = US
00:44:01:CRYPTO_PKI:Name:CN = msca-rootRA, O = Cisco System, C = US
00:44:01:CRYPTO_PKI:transaction GetCACert completed
00:44:01:CRYPTO_PKI:CA certificate received.
00:44:01:CRYPTO_PKI:CA certificate received.
pki-36a(config)# crypto ca enroll msca
%
% Start certificate enrollment ..
% Create a challenge password. You will need to verbally provide this
password to the CA Administrator in order to revoke your certificate.
For security reasons your password will not be saved in the configuration.
Please make a note of it.

Password:
Re-enter password:

% The subject name in the certificate will be:pki-36a.cisco.com
% Include the router serial number in the subject name? [yes/no]:n
% Include an IP address in the subject name? [yes/no]:n
Request certificate from CA? [yes/no]:y
% Certificate request sent to Certificate Authority
% The certificate request fingerprint will be displayed.
% The 'show crypto ca certificate' command will also show the fingerprint.

pki-36a(config)#      Fingerprint: 2CFC6265 77BA6496 3AEFCB50 29BC2BF2

00:44:29:CRYPTO_PKI:transaction PKCSReq completed
00:44:29:CRYPTO_PKI:status:
00:44:29:CRYPTO_PKI:http connection opened
00:44:29:CRYPTO_PKI: received msg of 1924 bytes
00:44:29:CRYPTO_PKI:HTTP response header:
  HTTP/1.1 200 OK
Server:Microsoft-IIS/5.0
Date:Fri, 17 Nov 2000 18:51:28 GMT
Content-Length:1778
Content-Type:application/x-pki-message

00:44:29:CRYPTO_PKI:signed attr:pki-message-type:
00:44:29:13 01 33
00:44:29:CRYPTO_PKI:signed attr:pki-status:
00:44:29:13 01 30
00:44:29:CRYPTO_PKI:signed attr:pki-recipient-nonce:
00:44:29:04 10 B4 C8 2A 12 9C 8A 2A 4A E1 E5 15 DE 22 C2 B4 FD
00:44:29:CRYPTO_PKI:signed attr:pki-transaction-id:
00:44:29:13 20 34 45 45 41 44 42 36 33 38 43 33 42 42 45 44 45 39 46
00:44:29:34 38 44 33 45 36 39 33 45 33 43 37 45 39
00:44:29:CRYPTO_PKI:status = 100:certificate is granted
00:44:29:CRYPTO__PKI:All enrollment requests completed.
00:44:29:%CRYPTO-6-CERTRET:Certificate received from Certificate Authority

```

Related Commands

Command	Description
crypto ca authenticate	Authenticates the CA (by getting the certificate of the CA).
crypto ca enroll	Obtains the certificate of your router from the CA.

Command	Description
debug crypto pki messages	Displays debug messages for details of the interaction (message dump) between the CA and the router.
show crypto ca certificates	Displays information about your certificate, the certificate of the CA, and any RA certificates.

debug csm voice

Use the **debug csm voice** privileged EXEC command to turn on debugging for all CSM Voice-over-IP calls. Use the **no** form of this command to disable debugging output.

```
debug csm voice [slot | dspm | dsp | dsp-channel]
```

```
no debug csm voice [slot | dspm | dsp | dsp-channel]
```

Syntax Description	
<code>slot dspm dsp dsp-channel</code>	Identifies the location of a particular DSP channel.

Usage Guidelines

The **debug csm voice** command turns on debugging for all CMS Voice-over-IP calls. If this command has no keyword specified, then debugging is enabled for all voice calls. The **no debug csm voice** command turns off debugging information for all voice calls.

If the keyword `slot | dspm | dsp | dsp-channel` is specified, then (if the specified DSP channel is engaged in a CSM call) CMS call-related debugging information will be turned on for this channel. The **no** form of this command turns off debugging for that particular channel.

Examples

The following examples show sample output from the **debug csm voice** command. Figure 5 shows that CSM has received an event from ISDN.

Figure 5 Sample debug csm voice Command

```
Oct 18 04:05:07.052: EVENT_FROM_ISDN::dchan_idb=0x60D7B6B8, call_id=0xCF, ces=0x1
bchan=0x0, event=0x1, cause=0x0
```

In this example:

- `dchan_idb`—Indicates the address of the hardware IDB for the D channel
- `call_id`—Indicates the call id assigned by ISDN
- `bchan`—Indicates the number of the B channel assigned for this call
- `cause`—Indicates the ISDN event cause

Figure 6 shows that CSM has allocated the CSM voice control block for the DSP device on slot 1 port 10 for this call.

Figure 6 Sample debug csm voice Command

```
Oct 18 04:05:07.052: VDEV_ALLOCATE: slot 1 and port 10 is allocated.
```

This AS5300 might not be actually used to handle this call. CSM must first allocate the CSM voice control block to initiate the state machine. After the voice control block has been allocated, CSM obtains from DSP Resource Manager the actual DSP channel that will be used for the call. At that point, CSM will switch over to the actual logical port number. The slot number refers to the physical slot on the AS5300. The port number is the logical DSP number interpreted as listed in Table 36.

Table 36 Logical DSP Numbers

Logical Port Number	Physical DSP Channel
Port 0	DSPRM 1, DSP 1, DSP channel 1
Port 1	DSPRM 1, DSP 1, DSP channel 2
Port 2	DSPRM 1, DSP 2, DSP channel 1
Port 3	DSPRM 1, DSP 2, DSP channel 2
Port 4	DSPRM 1, DSP 3, DSP channel 1
Port 5	DSPRM 1, DSP 3, DSP channel 2
Port 6	DSPRM 1, DSP 4, DSP channel 1
Port 7	DSPRM 1, DSP 4, DSP channel 2
Port 8	DSPRM 1, DSP 5, DSP channel 1
Port 9	DSPRM 1, DSP 5, DSP channel 2
Port 10	DSPRM 1, DSP 6, DSP channel 1
Port 11	DSPRM 1, DSP 6, DSP channel 2
Port 12	DSPRM 2, DSP 1, DSP channel 1
Port 13	DSPRM 2, DSP 1, DSP channel 2
Port 14	DSPRM 2, DSP 2, DSP channel 1
Port 15	DSPRM 2, DSP 2, DSP channel 2
Port 16	DSPRM 2, DSP 3, DSP channel 1
Port 17	DSPRM 2, DSP 3, DSP channel 2
Port 18	DSPRM 2, DSP 4, DSP channel 1
Port 19	DSPRM 2, DSP 4, DSP channel 2
Port 20	DSPRM 2, DSP 5, DSP channel 1
Port 21	DSPRM 2, DSP 5, DSP channel 2
Port 22	DSPRM 2, DSP 6, DSP channel 1
Port 23	DSPRM 2, DSP 6, DSP channel 2
Port 48	DSPRM 5, DSP 1, DSP channel 1
Port 49	DSPRM 5, DSP 1, DSP channel 2
Port 50	DSPRM 5, DSP 2, DSP channel 1
Port 51	DSPRM 5, DSP 2, DSP channel 2
Port 52	DSPRM 5, DSP 3, DSP channel 1
Port 53	DSPRM 5, DSP 3, DSP channel 2
Port 54	DSPRM 5, DSP 4, DSP channel 1
Port 55	DSPRM 5, DSP 4, DSP channel 2
Port 56	DSPRM 5, DSP 5, DSP channel 1
Port 57	DSPRM 5, DSP 5, DSP channel 2
Port 58	DSPRM 5, DSP 6, DSP channel 1
Port 59	DSPRM 5, DSP 6, DSP channel 2

Figure 7 shows that the function `csm_vtsp_init_tdm()` has been called with a voice control block of address `0x60B8562C`. This function will only be called when the call is treated as a voice call.

Figure 7 Sample debug csm voice Command

```
Oct 18 04:05:07.052: csm_vtsp_init_tdm (voice_vdev=0x60B8562C)
```

Figure 8 shows that CSM has obtained a DSP channel from the DSP Resource Manager.

Figure 8 Sample debug csm voice Command

```
Oct 18 04:05:07.052: csm_vtsp_init_tdm: dsprm_tdm_allocate: tdm slot 1, dspm 2, dsp 5,
dsp_channel 1csm_vtsp_init_tdm: dsprm_tdm_allocate: tdm stream 5, channel 9, bank 0,
bp_channel 10
```

The DSP channel has the following initialized TDM channel information:

- `tdm slot 1, dspm 2, dsp 5, dsp_channel 1`—Indicates the physical DSP channel that will be used for this call.
- `tdm stream 5, channel 9, bank 0, bp_channel 10`—Indicates the on-chip and backplane tdm channel assigned to this DSP channel. Stream 5, channel 9 gives the on-chip tdm channel mapped to the DSP; bank 0, bp_channel 10 means that the backplane stream 0 and backplane channel #1 are assigned to this DSP.

Figure 9 shows that CSM has received an incoming call event from ISDN.

Figure 9 Sample debug csm voice Command

```
Oct 18 04:05:07.052: EVENT_FROM_ISDN:(00CF): DEV_INCALL at slot 1 and port 20
```

Slot 1, port 20 means the logical DSP channel 20 (mapped to DSPRM 2, DSP 5, DSP channel 1).

Figure 10 shows that the `DEV_INCALL` message has been translated into `CSM_EVENT_ISDN_CALL` message

Figure 10 Sample debug csm voice Command

```
Oct 18 04:05:07.052: CSM_PROC_IDLE: CSM_EVENT_ISDN_CALL at slot 1, port 20
```

This message is passed to the CSM central state machine while it is in the `CSM_IDLE` state and is in the `CSM_PROC_IDLE` procedure. The logical DSP channel port 20 on slot 1 is used to handle this call.

Figure 11 shows that CSM has invoked the `vtsp_ic_notify()` function with a CSM voice call control block `0x60B8562C`.

Figure 11 Sample debug csm voice Command

```
Oct 18 04:05:07.052: vtsp_ic_notify : (voice_vdev= 0x60B8562C)
```

Inside this function, CSM will send a `SETUP INDICATION` message to VTSP. This function will only be invoked if the call is a voice call.

Figure 12 shows that CSM has received a `SETUP INDICATION RESPONSE` message from VTSP as an acknowledgement.

Figure 12 Sample debug csm voice Command

```
Oct 18 04:05:07.056: csm_vtsp_call_setup_resp (vdev_info=0x60B8562C, vtsp_cdb=0x60FCA114)
```

This means that VTSP has received the CALL SETUP INDICATION message previously sent and has proceeded to process the call.

- vdev_info—Contains the address of the CSM voice data block.
- vtsp_cdb—Contains the address of the VTSP call control block.

Figure 13 shows that CSM has received a CALL CONNECT message from VTSP.

Figure 13 Sample debug csm voice Command

```
Oct 18 04:05:07.596: csm_vtsp_call_connect (vtsp_cdb=0x60FCA114, voice_vdev=0x60B8562C)
```

This indicates that VTSP has received a CONNECT for the call leg initiated to the Internet side.

- voice_vdev—Contains the address of the CSM voice data block.
- vtsp_cdb—Contains the address of the VTSP call control block.

Figure 14 shows that while CSM is in the CSM_IC2_RING state, it receives a SETUP INDICATION RESPONSE from VTSP. This message is translated into CSM_EVENT_MODEM_OFFHOOK and passed to the CSM central state machine.

Figure 14 Sample debug csm voice Command

```
Oct 18 04:05:07.596: CSM_PROC_IC2_RING: CSM_EVENT_MODEM_OFFHOOK at slot 1, port 20
```

Figure 15 shows that CSM has received a CONNECT message from ISDN for the call using the logical DSP channel on slot 1 and port 20.

Figure 15 Sample debug csm voice Command

```
Oct 18 04:05:07.616: EVENT_FROM_ISDN:(00CF): DEV_CONNECTED at slot 1 and port 20
```

Figure 16 shows that CSM has translated the CONNECT event from ISDN into the CSM_EVENT_ISDN_CONNECTED message, which is then passed to the CSM central state machine.

Figure 16 Sample debug csm voice Command

```
Oct 18 04:05:07.616: CSM_PROC_IC4_WAIT_FOR_CARRIER: CSM_EVENT_ISDN_CONNECTED at slot 1, port 20
```

Figure 17 shows that CSM has received a CALL SETUP REQUEST from VTSP.

Figure 17 Sample debug csm voice Command

```
May 16 12:22:27.580: csm_vtsp_call_setup_request (vtsp_cdb=0x60FCFA20, vtsp_sdb=0x60DFB608)
```

This represents a request to make an outgoing call to the PSTN:

- vtsp_cdb—Contains the address of the VTSP call control block
- vtsp_sdb—Contains the address of the signalling data block for the signalling interface to be used to send the outgoing call

Figure 18 shows that the physical DSP channel has been allocated for this outgoing call.

Figure 18 Sample debug csm voice Command

```
May 16 12:22:27.580: csm_vtsp_call_setup_request: tdm slot 1, dspm 5, dsp 4, dsp_channel 1
```

Figure 19 shows the on-chip and backplane tdm channel assigned to this DSP channel.

Figure 19 Sample debug csm voice Command

```
May 16 12:22:27.580: csm_vtsp_call_setup_request: tdm stream 5, channel 25, bank 0, bp_channel 27
```

In this sample output, tdm stream 5, channel 25, bank 0, bp_channel 27 indicates the on-chip and backplane tdm channel assigned to this dsp channel. Stream 5, channel 25 gives the on-chip tdm channel mapped to the DSP; bank 0, bp_channel 27 means backplane stream 0, backplane channel #1 is assigned to this DSP.

Figure 20 shows the calling number and the called number for this call.

Figure 20 Sample debug csm voice Command

```
May 16 12:22:27.580: csm_vtsp_call_setup_request: calling number: 10001, called number: 30001
```

Figure 21 shows that the CALL SETUP REQUEST from VTSP has been translated into the ' CSM_EVENT_MODEM_OFFHOOK message and is passed to the CSM central state machine.

Figure 21 Sample debug csm voice Command

```
May 16 12:22:27.580: CSM_PROC_IDLE: CSM_EVENT_MODEM_OFFHOOK at slot 1, port 54
```

The logical DSP channel number for the DSP (slot 1, port 54) is now displayed, which maps to the physical DSP channel slot 1, dspm 5, dsp 4, dsp_channel 1.

Figure 22 shows that CSM has collected all the digits for dialing out.

Figure 22 Sample debug csm voice Command

```
May 16 12:22:27.580: CSM_PROC_OC3_COLLECT_ALL_DIGIT: CSM_EVENT_GET_ALL_DIGITS at slot 1, port 54
```

For PRI and for applications that do not require digit collection of outdialing digits (for example, voice calls), the intermediate digit collection states are skipped and the CSM state machine jumps to this state directly, pretending that the digit collection has been done.

Figure 23 shows an information message.

Figure 23 Sample debug csm voice Command

```
May 16 12:22:27.580: CSM_PROC_OC3_COLLECT_ALL_DIGIT: called party num: (30001) at slot 1, port 54
```

Figure 24 shows that CSM attempts to find a free signalling D channel to direct the outgoing call.

Figure 24 Sample debug csm voice Command

```
May 16 12:22:27.580: csm_vtsp_check_dchan (voice_vdev=0x60B8562C)
May 16 12:22:27.580: csm_vtsp_check_dchan (vtsp requested dchan=0x60D7ACB0,
dchan_idb=0x60E8ACF0)
May 16 12:22:27.580: csm_vtsp_check_dchan (voice_vdev=0x60B8562C)
May 16 12:22:27.580: csm_vtsp_check_dchan (vtsp requested dchan=0x60D7ACB0,
dchan_idb=0x60D7ACB0)
```

In the case of voice calls, the free signalling D channel must match the voice interface specified inside the signalling data block (vtsp_sdb) passed from VTSP.

Figure 25 shows that CSM has received an event from ISDN.

Figure 25 Sample debug csm voice Command

```
May 16 12:22:27.624: EVENT_FROM_ISDN::dchan_idb=0x60D7ACB0, call_id=0xA121, ces=0x1
bchan=0x1E, event=0x3, cause=0x0
```

In this sample output:

- dchan_idb—indicates the address of the hardware IDB for the D channel
- call_id—Indicates the call id assigned by ISDN
- bchan—Indicates the number of the B channel assigned for this call
- cause—Indicates the ISDN event cause

Figure 26 shows that CSM has received a CALL PROCEEDING message from ISDN.

Figure 26 Sample debug csm voice Command

```
May 16 12:22:27.624: EVENT_FROM_ISDN:(A121): DEV_CALL_PROC at slot 1 and port 54
```

Figure 27 shows that the CALL PROCEEDING event received from ISDN has been interpreted as a CSM_EVENT_ISDN_BCHAN_ASSIGNED message.

Figure 27 Sample debug csm voice Command

```
*May 16 12:22:27.624: CSM_PROC_OC4_DIALING: CSM_EVENT_ISDN_BCHAN_ASSIGNED at slot 1, port
54
```

ISDN has assigned a B channel for this outgoing call. This B channel must be on the same PRI span as the signalling D channel we have allocated previously.

Figure 28 shows that the csm_vtsp_setup_for_oc function is called.

Figure 28 Sample debug csm voice Command

```
May 16 12:22:27.624: csm_vtsp_setup_for_oc (voice_vdev=0x60B8562C)
```

This is invoked when an outgoing call initiated by the VTSP receives a response from the ISDN stack.

Figure 29 shows that ISDN has sent a CONNECT message to CSM indicating that the call leg to the PSTN side has been established.

Figure 29 Sample debug csm voice Command

```
May 16 12:22:28.084: EVENT_FROM_ISDN::dchan_idb=0x60D7ACB0, call_id=0xA121, ces=0x1
    bchan=0x1E, event=0x4, cause=0x0
May 16 12:22:28.084: EVENT_FROM_ISDN:(A121): DEV_CONNECTED at slot 1 and port 54
```

Figure 30 shows that while CSM is in the OC5_WAIT_FOR_CARRIER state, it has received the 'CONNECT' message from ISDN and has translated it into the CSM_EVENT_ISDN_CONNECTED message, which is passed to the CSM central state machine.

Figure 30 Sample debug csm voice Command

```
May 16 12:22:28.084: CSM_PROC_OC5_WAIT_FOR_CARRIER: CSM_EVENT_ISDN_CONNECTED at slot 1,
port 54
```

Figure 31 shows that the function vtsp_confirm_oc() has been called.

Figure 31 Sample debug csm voice Command

```
May 16 12:22:28.084: vtsp_confirm_oc : (voice_vdev= 0x60B8562C)
```

This is invoked after CSM received the CONNECT from ISDN. CSM sends a confirmation of the CONNECT to VTSP.

debug custom-queue

To enable custom queueing output, use the **debug custom-queue EXEC** command. Use the **no** form of this command to disable custom queueing output.

debug custom-queue

no debug custom-queue

Syntax Description

This command has no arguments or keywords.

Examples

The following is an example of enabling custom queueing output:

```
Router# debug custom-queue
Custom output queueing debugging is on
```

The following is sample output from the **debug custom-queue** command:

```
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 2
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 2 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 2
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 2 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 2
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 2 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
00:27:38: CQ: Serial0 output (Pk size/Q: 232/1) Q # was 1 now 1
```

Related Commands

Command	Description
debug priority	Enables priority queueing output.

debug dbconn all

Use the **debug dbconn all** privileged EXEC command to turn on all debug flags for Database Connection. The Database Connection debug flags include **appc**, **config**, **drda**, **event**, and **tcp**. Use the **no** form of this command to disable all debugging output.

debug dbconn all

no debug dbconn all

Syntax Description

This command has no arguments or keywords.

Defaults

Debugging is not enabled for Database Connection.

Usage Guidelines

The **debug dbconn all** command displays debug output for APPC, Database Connection configuration, DRDA, error messages, event traces, and TCP.

Examples

See the sample outputs provided for the **debug dbconn appc**, **debug dbconn config**, **debug dbconn drda**, **debug dbconn event**, and **debug dbconn tcp** commands.

Related Commands

Command	Description
debug dbconn appc	Displays APPC-related trace or error messages.
debug dbconn config	Displays trace or error messages for Database Connection configuration and control blocks.
debug dbconn drda	Displays error messages and stream traces for DRDA.
debug dbconn event	Displays trace or error messages for Database Connection events.
debug dbconn tcp	Displays error messages and traces for TCP.

debug dbconn appc

Use the **debug dbconn appc** privileged EXEC command to display APPC-related trace or error messages. Use the **no** form of this command to disable debugging output.

debug dbconn appc

no debug dbconn appc

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

In a router with stable Database Connection, the `alias_cp_name` field in the trace message should not be blank. There should be no other APPC error message. You can use APPN debug commands with this debug command to track APPN-related errors.

Examples

The following is sample output from the **debug dbconn appc** command. In a normal situation, only the following message is displayed:

```
DBCONN-APPC: alias_cp_name is "ASH"
```

The following error messages are displayed if there is a network configuration error or other APPN related problem:

```
DBCONN-APPC-612C2B28: APPC error: opcode 0x1, primary_rc 0x0003,
secondary_rc 0x00000004
DBCONN-APPC-612C2B28: Verb block =
DBCONN-APPC-612C2B28: 0001 0200 0003 0000 0000 0004 0020 100C
DBCONN-APPC-612C2B28: 610A 828B 0000 0000 0000 0000 0000 0000
DBCONN-APPC-612C2B28: 0000 0000 8014 0003 0000 0000 0000 0000
DBCONN-APPC-612C2B28: D3E4 F6F2 E2E3 C1D9 C4C2 F240 4040 4040
DBCONN-APPC-612C2B28: 4040 4040 4040 4040 4040 4040 4040 4040
DBCONN-APPC-612C2B28: 4040 4040 4040 4040 4040 4040 4040 4040
DBCONN-APPC-612C2B28: 4040 4040 4040 4040 4040 4040 4040 4040
DBCONN-APPC-612C2B28: 4040 4040 4040 4040 0200 0000 0000 0000
DBCONN-APPC-612C2B28: 0000 0000 D4C5 D9D9 C9C5 4040 4040 D7C5
DBCONN-APPC-612C2B28: E3C5 D940 4040 4040 0000 0000 0000 0000
DBCONN-APPC-612C2B28: 00E2 E3C1 D9E6 4BE3 D6D9 C3C8 4040 4040
DBCONN-APPC-612C2B28: 4040 0000 0000 0000 0000 0000
DBCONN-APPC-612C2B28: ALLOCATE verb block =
DBCONN-APPC-612C2B28: 0001 0200 0003 0000 0000 0004 0020 100C
DBCONN-APPC-612C2B28: 610A 828B 0000 0000 0000 0000 0000 0000
DBCONN-APPC-612C2B28: 0000 0000 8014 0003 0000 0000 0000 0000
DBCONN-APPC-612C2B28: D3E4 F6F2 E2E3 C1D9 C4C2 F240 4040 4040
DBCONN-APPC-612C2B28: 4040 4040 4040 4040 4040 4040 4040 4040
DBCONN-APPC-612C2B28: 4040 4040 4040 4040 4040 4040 4040 4040
DBCONN-APPC-612C2B28: 4040 4040 4040 4040 4040 4040 4040 4040
DBCONN-APPC-612C2B28: 4040 4040 4040 4040 0200 0000 0000 0000
```

You can use the **debug appn** command to obtain more information.

The following message is displayed if a database connection is manually cleared and there is an outstanding APPC verb pending:

```
DBCONN-APPC-%612C2B28: Canceling pending APPC verb 0x1
```

Related Commands

Command	Description
debug dbconn all	Turns on all debug flags for Database Connection.
debug dbconn config	Displays trace or error messages for Database Connection configuration and control blocks.
debug dbconn drda	Displays error messages and stream traces for DRDA.
debug dbconn event	Displays trace or error messages for Database Connection events.
debug dbconn tcp	Displays error messages and traces for TCP.

debug dbconn config

Use the **debug dbconn config** privileged EXEC command to display trace or error messages for Database Connection configuration and control blocks. Use the **no** form of this command to disable debugging output.

debug dbconn config

no debug dbconn config

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

Most of the messages for Database Connection and control blocks do not report any errors. If a connection is inactive and cannot be cleared, use this command with **debug dbconn appc**, **debug dbconn tcp**, and **debug appn** commands to locate the problem. The `alias_cp_name` field must match the configured APPN cpname.

Examples

The following is sample output from the **debug dbconn config** command:

```

DBCONN-CONFIG: alias_cp_name is "ASH      "
DBCONN-CONFIG: connection 612BDAAC matching server on 198.147.235.5:0 with
rdbname=STELLA
DBCONN-CONFIG: APPN shutdown; clearing connection 1234abcd
DBCONN-CONFIG: created server 612C2720
DBCONN-CONFIG: server 612C2720 (listen 60F72E94) is active
DBCONN-CONFIG: server 612C2720 (listen 60F72E94) is active
DBCONN-CONFIG: new connection 612BDAAC
DBCONN-CONFIG: listen 60F72E94 accepts connection 612BDAAC
DBCONN-CONFIG: server 60F74614 takes connection 612BDAAC
DBCONN-CONFIG: listen 60F72E94 releases connection 612BDAAC
DBCONN-CONFIG: server 60F74614 releases connection 612BDAAC
DBCONN-CONFIG: deleting connection 612BDAAC
DBCONN-CONFIG: listen 60F72E94 abandons connection 612BDAAC
DBCONN-CONFIG: server 612C2720 abandons connection 612BDAAC
DBCONN-CONFIG: deleting server 612C2720
DBCONN-CONFIG: daemon 60381738 takes zombie connection 612BDAAC
DBCONN-CONFIG: daemon 60381738 releases zombie connection 612BDAAC

```

Related Commands

Command	Description
debug dbconn all	Turns on all debug flags for Database Connection.
debug dbconn appc	Displays APPC-related trace or error messages.
debug dbconn drda	Displays error messages and stream traces for DRDA.
debug dbconn event	Displays trace or error messages for Database Connection events.
debug dbconn tcp	Displays error messages and traces for TCP.

debug dbconn drda

Use the **debug dbconn drda** privileged EXEC command to display error messages and stream traces for DRDA. Use the **no** form of this command to disable debugging output.

debug dbconn drda

no debug dbconn drda

Syntax Description This command has no arguments or keywords.

Defaults By default, debugging is not enabled for the dbconn subsystem.

Command History	Release	Modification
	11.3(2)T	This command was introduced.
	12.0(5)XN	Command moved from CDBC feature to CTRC feature.

Examples The following example displays output from the **debug dbconn drda** command:

```
Router# debug dbconn drda
*Jun 30 16:09:32.363: DBCONN-DRDA-62008300: DSS X'006CD0410001', length 108, in chain,
REQDSS, correlator 1
*Jun 30 16:09:32.363: DBCONN-DRDA-62008300: OBJECT X'00661041', length 98, code point
X'1041'
*Jun 30 16:09:32.363: DBCONN-DRDA-62008300: OBJECT X'0020115E' in COLLECTION X'1041',
length 28, code point X'115E'
*Jun 30 16:09:32.363: DBCONN-DRDA-62008300: OBJECT X'000C116D' in COLLECTION X'1041',
length 8, code point X'116D'
*Jun 30 16:09:32.363: DBCONN-DRDA-62008300: OBJECT X'0013115A' in COLLECTION X'1041',
length 15, code point X'115A' (skipping...)
```

Related Commands	Command	Description
	debug dbconn all	Displays all CTRC debugging information related to communications with DB2.
	debug dbconn appc	Displays APPC-related trace or error messages for communications with DB2.
	debug dbconn config	Displays trace or error messages for CTRC configuration and control blocks for DB2 communications.
	debug dbconn event	Displays trace or error messages for CTRC events related to DB2 communications.
	debug dbconn tcp	Displays error messages or traces for TCP/IP communications with DB2.
	debug snasw	Displays debugging information related to SNA Switching Services.

debug dbconn event

Use the **debug dbconn event** privileged EXEC command to display trace or error messages for CTRC events related to DB2 communications. Use the **no** form of this command to disable debugging output.

debug dbconn event

no debug dbconn event

Syntax Description This command has no arguments or keywords.

Defaults By default, debugging is not enabled for the dbconn subsystem.

Command History	Release	Modification
	11.3(2)T	This command was introduced.
	12.0(5)XN	Command moved from CDBC feature to CTRC feature.

Examples The following examples display output from the **debug dbconn event** command in a variety of situations. A normal trace for the **debug dbconn event** displays as follows:

```
Router# debug dbconn event
DBCONN-EVENT: Dispatch to 60FD6C00, from 0, msg 60F754CC, msgid 6468 'dh',
buffer 0.
DBCONN-EVENT: [*] Post to 61134240(cn), from 60EC5470(tc), msg 611419E4,
msgid 0x6372 'cr', buffer 612BF68C.
DBCONN-EVENT: Flush events called for pto 61182742, pfrom 61239837.
DBCONN-EVENT: Event discarded: to 61182742 (cn), from 61239837(ap), msg
61339273, msgid 0x6372 'cr' buffer 0.
DBCONN-EVENT: == Send to 1234abcd, from 22938acd, msg 72618394, msgid
0x6372 'cr', buffer 0.
```

If the following messages are displayed, contact Cisco technical support personnel:

```
DBCONN-TCPFSM-1234abcd: Cannot occur in state 2 on input 6363 ('cc')
DBCONN-APPCFSM-1234abcd: Cannot occur in state 3 on input 6363 ('cc')
```

Related Commands

Command	Description
debug dbconn all	Displays all CTRC debugging information related to communications with DB2.
debug dbconn appc	Displays APPC-related trace or error messages for communications with DB2.
debug dbconn config	Displays trace or error messages for CTRC configuration and control blocks for DB2 communications.
debug dbconn drda	Displays error messages or stream traces for DRDA communications with DB2.
debug dbconn tcp	Displays error messages or traces for TCP/IP communications with DB2.
debug snasw	Displays debugging information related to SNA Switching Services.
show debugging	Displays the state of each debugging option.

debug dbconn tcp

Use the **debug dbconn tcp** privileged EXEC command to display error messages and traces for TCP. Use the **no** form of this command to disable debugging output.

debug dbconn tcp

no debug dbconn tcp

Syntax Description This command has no arguments or keywords.

Defaults Debugging is not enabled for the dbconn subsystem.

Command History	Release	Modification
	11.3(2)T	This command was introduced.
	12.0(5)XN	Command moved from CDBC feature to CTRC feature.

Examples The following example displays output from the **debug dbconn tcp** command:

```
Router# debug dbconn tcp
DBCONN-TCP-63528473: tcpdriver_passive_open returned NULL
DBCONN-TCP-63528473: (no memory) tcp_reset(63829482) returns 4
DBCONN-TCP: tcp_accept(74625348,&error) returns tcb 63829482, error 4
DBCONN-TCP: (no memory) tcp_reset(63829482) returns 4
DBCONN-TCP-63528473: (open) tcp_create returns 63829482, error = 4
DBCONN-TCP-63528473: tcb_connect(63829482,1.2.3.4,2010) returns 4
DBCONN-TCP-63528473: (open error) tcp_reset(63829482) returns 4
DBCONN-TCP-63528473: tcp_create returns 63829482, error = 4
DBCONN-TCP-63528473: tcb_bind(63829482,0.0.0.0,2001) returns 4
DBCONN-TCP-63528473: tcp_listen(63829482,,) returns 4
DBCONN-TCP-63528473: (errors) Calling tcp_close (63829482)
```

Related Commands	Command	Description
	debug dbconn all	Displays all CTRC debugging information related to communications with DB2.
	debug dbconn appc	Displays APPC-related trace or error messages for communications with DB2.
	debug dbconn config	Displays trace or error messages for CTRC configuration and control blocks for DB2 communications.
	debug dbconn drda	Displays error messages or stream traces for DRDA communications with DB2.

Command	Description
debug dbconn event	Displays trace or error messages for CTTC events related to DB2 communications.
debug ip tcp	Displays debugging information related to TCP/IP.
debug snasw	Displays debugging information related to SNA Switching Services.
show debugging	Displays the state of each debugging option.

debug decnet adj

Use the **debug decnet adj** privileged EXEC command to display debugging information on DECnet adjacencies. The **no** form of this command disables debugging output.

debug decnet adj

no debug decnet adj

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the sample **debug decnet adj** command:

```
Router# debug decnet adj

DNET-ADJ: Level 1 hello from 1.3
DNET-ADJ: sending hellos
DNET-ADJ: Sending hellos to all routers on interface Ethernet0, blksize 1498
DNET-ADJ: Level 1 hello from 1.3
DNET-ADJ: 1.5 adjacency initializing
DNET-ADJ: sending triggered hellos
DNET-ADJ: Sending hellos to all routers on interface Ethernet0, blksize 1498
DNET-ADJ: Level 1 hello from 1.3
DNET-ADJ: 1.5 adjacency up
DNET-ADJ: Level 1 hello from 1.5
DNET-ADJ: 1.5 adjacency down, listener timeout
```

The following line indicates that the router is sending hellos to all routers on this segment, which in this case is Ethernet 0:

```
DNET-ADJ: Sending hellos to all routers on interface Ethernet0, blksize 1498
```

The following line indicates that the router has heard a hello from address 1.5 and is creating an adjacency entry in its table. The initial state of this adjacency will be *initializing*.

```
DNET-ADJ: 1.5 adjacency initializing
```

The following line indicates that the router is sending an unscheduled (triggered) hello as a result of some event, such as new adjacency being heard:

```
DNET-ADJ: sending triggered hellos
```

The following line indicates that the adjacency with 1.5 is now up, or active:

```
DNET-ADJ: 1.5 adjacency up
```

The following line indicates that the adjacency with 1.5 has timed out, because no hello has been heard from adjacency 1.5 in the time interval originally specified in the hello from 1.5:

```
DNET-ADJ: 1.5 adjacency down, listener timeout
```

The following line indicates that the router is sending an unscheduled hello, as a result of some event, such as the adjacency state changing:

```
DNET-ADJ: hello update triggered by state changed in dn_add_adjacency
```

debug decnet connects

Use the **debug decnet connects** privileged EXEC command to display debugging information of all connect packets that are filtered (permitted or denied) by DECnet access lists. The **no** form of this command disables debugging output.

debug decnet connects

no debug decnet connects

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

When you use connect packet filtering, it may be helpful to use the **decnet access-group** configuration command to apply the following basic access list:

```
access-list 300 permit 0.0 63.1023 eq any
```

You can then log all connect packets transmitted on interfaces to which you applied this list, in order to determine those elements on which your connect packets must be filtered.



Note

Packet password and account information is not logged in the **debug decnet connects** message, nor is it displayed by the **show access** EXEC command. If you specify **password** or **account** information in your access list, they can be viewed by anyone with access to the configuration of the router.

Examples

The following is sample output from the **debug decnet connects** command:

```
Router# debug decnet connects
```

```
DNET-CON: list 300 item #2 matched src=19.403 dst=19.309 on Ethernet0: permitted
  srcname="RICK" srcuic=[0,017]
  dstobj=42 id="USER"
```

Table 37 describes significant fields in the output.

Table 37 *debug decnet connects Command Field Descriptions*

Field	Description
DNET-CON:	Indicates that this is a debug decnet connects packet.
list 300 item #2 matched	Indicates that a packet matched the second item in access list 300.
src = 19.403	Indicates the source DECnet address for the packet.
dst = 19.309	Indicates the destination DECnet address for the packet.
on Ethernet0:	Indicates the router interface on which the access list filtering the packet was applied.
permitted	Indicates that the access list permitted the packet.
srcname = "RICK"	Indicates the originator user of the packet.

Table 37 *debug decnet connects Command Field Descriptions (continued)*

Field	Description
srcuic = [0,017]	Indicates the source UIC of the packet.
dstobj = 42	Indicates that DECnet object 42 is the destination.
id="USER"	Indicates the access user.

debug decnet events

Use the **debug decnet events** privileged EXEC command to display debugging information on DECnet events. The **no** form of this command disables debugging output.

debug decnet events

no debug decnet events

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the **debug decnet events** command:

```
Router# debug decnet events
```

```
DNET: Hello from area 50 rejected - exceeded 'max area' parameter (45)
```

```
DNET: Hello from area 50 rejected - exceeded 'max area' parameter (45)
```

The following line indicates that the router received a hello from a router whose area was greater than the max-area parameter with which this router was configured:

```
DNET: Hello from area 50 rejected - exceeded 'max area' parameter (45)
```

The following line indicates that the router received a hello from a router whose node ID was greater than the max-node parameter with which this router was configured:

```
DNET: Hello from node 1002 rejected - exceeded 'max node' parameter (1000)
```

debug decnet packet

Use the **debug decnet packet** privileged EXEC command to display debugging information on DECnet packet events. The **no** form of this command disables debugging output.

debug decnet packet

no debug decnet packet

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the **debug decnet packet** command:

```
Router# debug decnet packet
```

```
DNET-PKT: src 1.4 dst 1.5 sending to PHASEV
```

```
DNET-PKT: Packet fwded from 1.4 to 1.5, via 1.5, snpa 0000.3080.cf90, TokenRing0
```

The following line indicates that the router is sending a converted packet addressed to node 1.5 to Phase V:

```
DNET-PKT: src 1.4 dst 1.5 sending to PHASEV
```

The following line indicates that the router forwarded a packet from node 1.4 to node 1.5. The packet is being sent to the next hop of 1.5 whose subnetwork point of attachment (MAC address) on that interface is 0000.3080.cf90.

```
DNET-PKT: Packet fwded from 1.4 to 1.5, via 1.5, snpa 0000.3080.cf90, TokenRing0
```

debug decnet routing

Use the **debug decnet routing** privileged EXEC command to display all DECnet routing-related events occurring at the router. The **no** form of this command disables debugging output.

debug decnet routing

no debug decnet routing

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the **debug decnet routing** command:

```
Router# debug decnet routing

DNET-RT: Received level 1 routing from 1.3 on Ethernet0 at 1:16:34
DNET-RT: Sending routes
DNET-RT: Sending normal routing updates on Ethernet0
DNET-RT: Sending level 1 routing updates on interface Ethernet0
DNET-RT: Level1 routes from 1.5 on Ethernet0: entry for node 5 created
DNET-RT: route update triggered by after split route pointers in dn_rt_input
DNET-RT: Received level 1 routing from 1.5 on Ethernet 0 at 1:18:35
DNET-RT: Sending L1 triggered routes
DNET-RT: Sending L1 triggered routing updates on Ethernet0
DNET-RT: removing route to node 5
```

The following line indicates that the router has received a level 1 update on interface Ethernet 0:

```
DNET-RT: Received level 1 routing from 1.3 on Ethernet0 at 1:16:34
```

The following line indicates that the router is sending its scheduled updates on interface Ethernet 0:

```
DNET-RT: Sending normal routing updates on Ethernet0
```

The following line indicates that the route will send an unscheduled update on this interface as a result of some event. In this case, the unscheduled update is a result of a new entry created in the interface's routing table.

```
DNET-RT: route update triggered by after split route pointers in dn_rt_input
```

The following line indicates that the router sent the unscheduled update on Ethernet 0:

```
DNET-RT: Sending L1 triggered routes
DNET-RT: Sending L1 triggered routing updates on Ethernet0
```

The following line indicates that the router removed the entry for node 5 because the adjacency with node 5 timed out, or the route to node 5 through a next-hop router went away:

```
DNET-RT: removing route to node 5
```

debug dhcp

To display debugging information about the Dynamic Host Configuration Protocol (DHCP) client activities and to monitor the status of DHCP packets, use the **debug dhcp** command in privileged EXEC mode. The **no** form of this command disables debugging output.

debug dhcp [detail]

no debug dhcp [detail]

Syntax Description

detail (Optional) Displays additional debug information.

Usage Guidelines

You can also use the **debug dhcp** command to monitor the subnet allocation and releasing for on-demand address pools.

For debugging purposes, the **debug dhcp detail** command provides the most useful information such as the lease entry structure of the client and the state transitions of the lease entry. The debug output shows the scanned option values from received DHCP messages that are replies to a router request. The values of the op, htype, hlen, hops, server identifier option, xid, secs, flags, ciaddr, yiaddr, siaddr, and giaddr fields of the DHCP packet are shown in addition to the length of the options field.

Examples

The following examples show and explain some of the typical debug messages you might see when using the **debug dhcp detail** command.

The following example shows the debug output when a DHCP client sends out a DHCPDISCOVER broadcast message to find its local DHCP server:

```
Router# debug dhcp detail
00:07:16:DHCP:DHCP client process started:10
00:07:16:RAC:Starting DHCP discover on Ethernet2
00:07:16:DHCP:Try 1 to acquire address for Ethernet2
00:07:16:%SYS-5-CONFIG_I:Configured from console by console
00:07:19:DHCP:Shutting down from get_netinfo()
00:07:19:DHCP:Attempting to shutdown DHCP Client
00:07:21:DHCP:allocate request
00:07:21:DHCP:new entry. add to queue
00:07:21:DHCP:SDiscover attempt # 1 for entry:
```

The first seven lines of the following output show the current values stored in the lease entry structure for the client:

```
00:07:21:Temp IP addr:0.0.0.0 for peer on Interface:Ethernet2
00:07:21:Temp sub net mask:0.0.0.0
00:07:21: DHCP Lease server:0.0.0.0, state:1 Selecting
00:07:21: DHCP transaction id:582
00:07:21: Lease:0 secs, Renewal:0 secs, Rebind:0 secs
00:07:21: Next timer fires after:00:00:03
00:07:21: Retry count:1 Client-ID:cisco-0010.7b6e.afd8-Et2
00:07:21:DHCP:SDiscover:sending 308 byte length DHCP packet
00:07:21:DHCP:SDiscover 308 bytes
00:07:21: B'cast on Ethernet2 interface from 0.0.0.0
```

The following example shows the offered addresses and parameters sent to the DHCP client by the DHCP server via a DHCPOFFER message. The messages containing the field “Scan” indicate the options that were scanned from the received BOOTP packet and the corresponding values.

```
00:07:23:DHCP:Received a BOOTREP pkt
00:07:23:DHCP:Scan:Message type:DHCP Offer
00:07:23:DHCP:Scan:Server ID Option:10.1.1.1 = A010101
00:07:23:DHCP:Scan:Lease Time:180
00:07:23:DHCP:Scan:Renewal time:90
00:07:23:DHCP:Scan:Rebind time:157
00:07:23:DHCP:Scan:Subnet Address Option:255.255.255.0
```

The following debug output shows selected fields in the received BOOTP packet:

```
00:07:23:DHCP:rcvd pkt source:10.1.1.1, destination: 255.255.255.255
00:07:23: UDP sport:43, dport:44, length:308
00:07:23: DHCP op:2, htype:1, hlen:6, hops:0
00:07:23: DHCP server identifier:10.1.1.1
00:07:23: xid:582, secs:0, flags:8000
00:07:23: client:0.0.0.0, your:10.1.1.2
00:07:23: srvr: 0.0.0.0, gw:0.0.0.0
00:07:23: options block length:60

00:07:23:DHCP Offer Message Offered Address:10.1.1.2
00:07:23:DHCP:Lease Seconds:180 Renewal secs: 90 Rebind secs:157
00:07:23:DHCP:Server ID Option:10.1.1.1
00:07:23:DHCP:offer received from 10.1.1.1
```

The following example shows the debug output when the DHCP client sends out a DHCPREQUEST broadcast message to the DHCP server to accept the offered parameters:

```
00:07:23:DHCP:SRequest attempt # 1 for entry:
00:07:23:Temp IP addr:10.1.1.2 for peer on Interface:Ethernet2
00:07:23:Temp sub net mask:255.255.255.0
00:07:23: DHCP Lease server:10.1.1.1, state:2 Requesting
00:07:23: DHCP transaction id:582
00:07:23: Lease:180 secs, Renewal:0 secs, Rebind:0 secs
00:07:23: Next timer fires after:00:00:02
00:07:23: Retry count:1 Client-ID:cisco-0010.7b6e.afd8-Et2
00:07:23:DHCP:SRequest- Server ID option:10.1.1.1
00:07:23:DHCP:SRequest- Requested IP addr option:10.1.1.2
00:07:23:DHCP:SRequest placed lease len option:180
00:07:23:DHCP:SRequest:326 bytes
00:07:23:DHCP:SRequest:326 bytes
00:07:23: B'cast on Ethernet2 interface from 0.0.0.0
```

The following example shows the debug output when the DHCP server sends a DHCPACK message to the client with the full set of configuration parameters:

```
00:07:23:DHCP:Received a BOOTREP pkt
00:07:23:DHCP:Scan:Message type:DHCP Ack
00:07:23:DHCP:Scan:Server ID Option:10.1.1.1 = A010101
00:07:23:DHCP:Scan:Lease Time:180
00:07:23:DHCP:Scan:Renewal time:90
00:07:23:DHCP:Scan:Rebind time:157
00:07:23:DHCP:Scan:Subnet Address Option:255.255.255.0
00:07:23:DHCP:rcvd pkt source:10.1.1.1, destination: 255.255.255.255
00:07:23: UDP sport:43, dport:44, length:308
00:07:23: DHCP op:2, htype:1, hlen:6, hops:0
00:07:23: DHCP server identifier:10.1.1.1
00:07:23: xid:582, secs:0, flags:8000
00:07:23: client:0.0.0.0, your:10.1.1.2
00:07:23: srvr: 0.0.0.0, gw:0.0.0.0
00:07:23: options block length:60
```

```
00:07:23:DHCP Ack Messag
00:07:23:DHCP:Lease Seconds:180 Renewal secs: 90 Rebind secs:157
00:07:23:DHCP:Server ID Option:10.1.1.1Interface Ethernet2 assigned DHCP address
10.1.1.2, mask 255.255.255.0
```

```
00:07:26:DHCP Client Pooling:***Allocated IP address:10.1.1.2
00:07:26:Allocated IP address = 10.1.1.2 255.255.255.0
```

Most fields are self-explanatory; however, fields that may need further explanation are described in Table 38.

Table 38 *debug dhcp Command Field Descriptions*

Fields	Description
DHCP:Scan:Subnet Address Option:255.255.255.0	Subnet mask option (option 1).
DHCP server identifier:1.1.1.1	Value of the DHCP server id option (option 54). Note that this is not the same as the siaddr field, which is the server IP address.
svr:0.0.0.0, gw:0.0.0.0	svr is the value of the siaddr field. gw is the value of the giaddr field.

Related Commands

Command	Description
debug ip dhcp server	Enables DHCP server debugging.
show dhcp lease	Displays DHCP addresses leased from a server.

debug dialer events

Use the **debug dialer events** privileged EXEC command to display debugging information about the packets received on a dialer interface. The **no** form of this command disables debugging output.

debug dialer events

no debug dialer events

Syntax Description

This command has no arguments or keywords.

Examples

When DDR is enabled on the interface, information concerning the cause of any call (called the *Dialing cause*) is displayed. The following line of output for an IP packet lists the name of the DDR interface and the source and destination addresses of the packet:

```
Dialing cause: Serial0: ip (s=172.16.1.111 d=172.16.2.22)
```

The following line of output for a bridged packet lists the DDR interface and the type of packet (in hexadecimal). For information on these packet types, see the “Ethernet Type Codes” appendix of the *Cisco IOS Bridging and IBM Networking Command Reference* publication.

```
Dialing cause: Serial1: Bridge (0x6005)
```

Most messages are self-explanatory; however, messages that may need some explanation are described in Table 39.

Table 39 General Debug Dialer Events Message Descriptions

Message	Description
Dialer0: Already <i>xxx</i> call(s) in progress on Dialer0, dialing not allowed	Number of calls in progress (<i>xxx</i>) exceeds the maximum number of calls set on the interface.
Dialer0: No free dialer - starting fast idle timer	All the lines in the interface or rotary group are busy and a packet is waiting to be sent to the destination.
BRI0: rotary group to <i>xxx</i> overloaded (<i>yyy</i>)	Number dialer (<i>xxx</i>) exceeds the load set on the interface (<i>yyy</i>).
BRI0: authenticated host <i>xxx</i> with no matching dialer profile	No dialer profile matches <i>xxx</i> , the remote host's CHAP name or remote name.
BRI0: authenticated host <i>xxx</i> with no matching dialer map	No dialer map matches <i>xxx</i> , the remote host's CHAP name or remote name.
BRI0: Can't place call, verify configuration	Dialer string or dialer pool on an interface not set.

Table 40 describes the messages that the **debug dialer events** command can generate for a serial interface used as a V.25bis dialer for dial-on-demand routing (DDR).

Table 40 Debug Dialer Events Message Descriptions for DDR

Message	Description
Serial 0: Dialer result = xxxxxxxxxx	Result returned from the V.25bis dialer. It is useful in debugging if calls are failing. On some hardware platforms, this message cannot be displayed due to hardware limitations. Possible values for the xxxxxxxxxx variable depend on the V.25bis device with which the router is communicating.
Serial 0: No dialer string defined. Dialing cannot occur.	Packet is received that should cause a call to be placed. However, there is no dialer string configured, so dialing cannot occur. This message usually indicates a configuration problem.
Serial 0: Attempting to dial xxxxxxxxxx	Packet has been received that passes the dial-on-demand access lists. That packet causes phone number xxxxxxxxxx to be dialed.
Serial 0: Unable to dial xxxxxxxxxx	Phone call to xxxxxxxxxx cannot be placed. This failure might be due to a lack of memory, full output queues, or other problems.
Serial 0: disconnecting call	Router hangs up a call.
Serial 0: idle timeout Serial 0: re-enable timeout Serial 0: wait for carrier timeout	One of these three messages is displayed when a dialer timer expires. These messages are mostly informational, but are useful for debugging a disconnected call or call failure.

Related Commands

Command	Description
debug dialer packets	Displays debugging information about the packets received on a dialer interface.