

## debug modem

Use the **debug modem** EXEC command to observe modem line activity on an access server. The **no** form of this command disables debugging output.

**[no] debug modem**

### Usage Guidelines

This output of this command is self-explanatory.

### Sample Display

Figure 2-181 shows sample **debug modem** output.

**Figure 2-181 Sample Debug Modem Output**

```
Router# debug modem

15:25:51: TTY4: DSR came up
15:25:51: tty4: Modem: IDLE->READY
15:25:51: TTY4: Autoselect started
15:27:51: TTY4: Autoselect failed
15:27:51: TTY4: Line reset
15:27:51: TTY4: Modem: READY->HANGUP
15:27:52: TTY4: dropping DTR, hanging up
15:27:52: tty4: Modem: HANGUP->IDLE
15:27:57: TTY4: restoring DTR
15:27:58: TTY4: DSR came up
```

The output in Figure 2-181 shows when the modem line changes state.

## debug modem csm

Use the **debug modem csm** EXEC command to debug the call state machine used to connect calls on the modem. Use the **no** form of this command to disable debug output.

```
[no] debug modem csm [slot/modem-port | group group-number]
```

### Syntax Description

*slot/modem-port* (Optional) Slot and modem port number.

**group** *group-number* (Optional) Modem group.

### Usage Guidelines

Use the **debug modem csm** command to troubleshoot call switching problems. With this command, you can trace the complete sequence of switching incoming and outgoing calls.

### Sample Displays

Figure 2-182 shows sample **debug modem csm** output. In this example, a call enters the modem (incoming) on slot 1 port 0.

**Figure 2-182 Sample Debug Modem CSM Output—Incoming Call**

```
router(config)# service timestamps debug uptime
router(config)# end
Router# debug modem csm

00:04:09: ccpri_ratetoteup bear rate is 10
00:04:09: CSM_MODEM_ALLOCATE: slot 1 and port 0 is allocated.
00:04:09: MODEM_REPORT(0001): DEV_INCALL at slot 1 and port 0
00:04:09: CSM_PROC_IDLE: CSM_EVENT_ISDN_CALL at slot 1, port 0
00:04:11: CSM_RING_INDICATION_PROC: RI is on
00:04:13: CSM_RING_INDICATION_PROC: RI is off
00:04:15: CSM_PROC_IC1_RING: CSM_EVENT_MODEM_OFFHOOK at slot 1, port 0
00:04:15: MODEM_REPORT(0001): DEV_CONNECTED at slot 1 and port 0
00:04:15: CSM_PROC_IC2_WAIT_FOR_CARRIER: CSM_EVENT_ISDN_CONNECTED at slot 1, port 0
```

Figure 2-183 shows sample **debug modem csm** output when call is dialed from the modem into the network (outgoing) from slot 1 port 2.

**Figure 2-183 Sample Debug Modem CSM Output—Outgoing Call**

```
Router# debug modem csm

atdt16665202
00:11:21: CSM_PROC_IDLE: CSM_EVENT_MODEM_OFFHOOK at slot 1, port 2
00:11:21: T1_MAIL_FROM_NEAT: DC_READY_RSP: mid = 1, slot = 0, unit = 0
00:11:21: CSM_PROC_OC1_REQUEST_DIGIT: CSM_EVENT_DIGIT_COLLECT_READY at slot 1, port 2
00:11:24: T1_MAIL_FROM_NEAT: DC_FIRST_DIGIT_RSP: mid = 1, slot = 0, unit = 0
00:11:24: CSM_PROC_OC2_COLLECT_1ST_DIGIT: CSM_EVENT_GET_1ST_DIGIT at slot 1, port 2
00:11:27: T1_MAIL_FROM_NEAT: DC_ALL_DIGIT_RSP: mid = 1, slot = 0, unit = 0
00:11:27: CSM_PROC_OC3_COLLECT_ALL_DIGIT: CSM_EVENT_GET_ALL_DIGITS (16665202) at slot 1, port 2
00:11:27: ccpri_ratetoteup bear rate is 10
```

```
00:11:27: MODEM_REPORT(A000): DEV_CALL_PROC at slot 1 and port 2
00:11:27: CSM_PROC_OC4_DIALING: CSM_EVENT_ISDN_BCHAN_ASSIGNED at slot 1, port 2
00:11:31: MODEM_REPORT(A000): DEV_CONNECTED at slot 1 and port 2
00:11:31: CSM_PROC_OC5_WAIT_FOR_CARRIER: CSM_EVENT_ISDN_CONNECTED at slot 1, port 2
CONNECT 19200/REL - MNP
```

### Related Commands

**debug modem oob**

**debug modem trace**

## debug modem oob

Use the **debug modem oob** EXEC command to debug the out-of-band port used to poll modem events on the modem. Use the **no** form of this command to disable debug output.

**[no] debug modem oob** [*slot/modem-port* | **group** *group-number*]

### Syntax Description

*slot/modem-port* (Optional) Slot and modem port number.

**group** *group-number* (Optional) Modem group.

### Usage Guidelines



**Caution** Entering the **debug modem oob** command without specifying a slot and modem number debugs *all* out-of-band ports, which generates a significant amount of information.

The message types and sequence numbers that appear in the debug output are initiated by the Modem Out-of-Band (OOB) Protocol and used by service personnel for debugging purposes.

### Sample Display

Figure 2-184 shows sample **debug modem oob** output. This example debugs the out-of-band port on modem 2/0, which creates modem startup messages between the network management software and the modem.

**Figure 2-184 Sample Debug Modem OOB Output**

```
Router# debug modem oob 2/0

MODEM(2/0): One message sent --Message type:3, Sequence number:0
MODEM(2/0): Modem DC session data reply
MODEM(2/0): One message sent --Message type:83, Sequence number:1
MODEM(2/0): DC session event =
MODEM(2/0): One message sent --Message type:82, Sequence number:2
MODEM(2/0): No status changes since last polled
MODEM(2/0): One message sent --Message type:3, Sequence number:3
MODEM(2/0): Modem DC session data reply
MODEM(2/0): One message sent --Message type:83, Sequence number:4
```

### Related Commands

**debug modem csm**

**debug modem trace**

## debug modem trace

Use the **debug modem trace** EXEC command to debug a call trace on the modem to determine why calls are terminated. Use the **no** form of this command to disable debug output.

**[no] debug modem trace [normal | abnormal | all] [slot/modem-port | group group-number]**

### Syntax Description

<b>normal</b>	(Optional) Uploads the call trace to the syslog server on normal call termination (for example, a local user hangup or a remote user hangup).
<b>abnormal</b>	(Optional) Uploads the call trace to the syslog server on abnormal call termination (for example, any call termination other than normal termination, such as a lost carrier or a watchdog timeout).
<b>all</b>	(Optional) Uploads the call trace on all call terminations including normal and abnormal call termination.
<i>slot/modem-port</i>	(Optional) Slot and modem port number.
<b>group</b> <i>group-number</i>	(Optional) Modem group.

### Usage Guidelines

The **debug modem trace** command applies only to manageable modems.

For additional information, use the **show modem** command.

### Sample Display

Figure 2-185 shows sample **debug modem trace abnormal** output.

**Figure 2-185 Sample Debug Modem Trace Command**

```
Router# debug modem trace abnormal 1/14

Modem 1/14 Abnormal End of Connection Trace. Caller 123-4567
  Start-up Response: AS5200 Modem, Firmware 1.0
  Control Reply: 0x7C01
  DC session response: brasil firmware 1.0
  RS232 event:
  DSR=On, DCD=On, RI=Off, TST=Off
  changes: RTS=No change, DTR=No change, CTS=No change
  changes: DSR=No change, DCD=No change, RI=No change, TST=No change
  Modem State event: Connected
  Connection event: Speed = 19200, Modulation = VFC
  Direction = Originate, Protocol = reliable/LAPM, Compression = V42bis
  DTR event: DTR On
  Modem Activity event: Data Active
  Modem Analog signal event: TX = -10, RX = -24, Signal to noise = -32
  End connection event: Duration = 10:34-11:43,
  Number of xmit char = 67, Number of rcvd char = 88, Reason: Watchdog Time-out.
```

Related Commands

**debug modem csm**

**debug modem oob**

## debug ncia circuit

Use the **debug ncia circuit** EXEC command to display circuit-related information between the native client interface architecture (NCIA) server and client. The **no** form of this command disables debugging output.

**[no] debug ncia circuit [error | event | flow-control | state]**

### Syntax Description

<b>error</b>	(Optional) Displays the error situation for each circuit.
<b>event</b>	(Optional) Displays the packets received and transmitted for each circuit.
<b>flow-control</b>	(Optional) Displays the flow control information for each circuit.
<b>state</b>	(Optional) Displays the state changes for each circuit.

### Usage Guidelines

NCIA is an architecture developed by Cisco for accessing SNA applications. This architecture allows native SNA interfaces on hosts and clients to access TCP/IP backbones.

You cannot enable debugging output for a particular client or particular circuit.



**Caution** Do not enable the **debug ncia circuit** command during normal operation because this command generates a large amount of output messages and could slow down the router.

### Sample Displays

Figure 2-186 shows sample **debug ncia circuit error** output. In this example, the possible errors are displayed. The first error message indicates that the router is out of memory. The second message indicates that the router has an invalid circuit control block. The third message indicates that the router is out of memory. The remaining messages identify errors related to the finite state machine.

**Figure 2-186 Sample Debug NCIA Circuit Error Output**

```
Router# debug ncia circuit error

NCIA: ncia_circuit_create memory allocation fail
NCIA: ncia_send_ndlc: invalid circuit control block
NCIA: send_ndlc: fail to get buffer for ndlc primitive xxx
NCIA: ncia circuit fsm: Invalid input
NCIA: ncia circuit fsm: Illegal state
NCIA: ncia circuit fsm: Illegal input
NCIA: ncia circuit fsm: Unexpected input
NCIA: ncia circuit fsm: Unknown error rtn code
```

Figure 2-187 shows sample **debug ncia circuit event** output. In this example, a session start-up sequence is displayed.

**Figure 2-187 Sample Debug NCIA Circuit Event Output**

```
Router# debug ncia circuit event

NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_START_DL, Len: 24, tmac: 4000.1060.1000,
         tsap: 4, csap 8, oid: 8A91E8, tid 0, lfs 16, ws 1
NCIA: create circuit: saddr 4000.1060.1000, ssap 4, daddr 4000.3000.0003, dsap 8 sid:
         8B09A8
NCIA: send NDLC_DL_STARTED to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_DL_STARTED, Len: 2,4 tmac: 4000.1060.1000,
          tsap: 4, csap 8, oid: 8A91E8, tid 8B09A8, lfs 16, ws 1
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_XID_FRAME, Len: 12, sid: 8B09A8, FC 0x81
NCIA: send NDLC_XID_FRAME to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_XID_FRAME, Len: 12, sid: 8A91E8, FC 0xC1
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_XID_FRAME, Len: 18, sid: 8B09A8, FC 0xC1
NCIA: send NDLC_CONTACT_STN to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_CONTACT_STN, Len: 12, sid: 8A91E8, FC 0xC1
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_STN_CONTACTED, Len: 12, sid: 8B09A8, FC 0xC1
NCIA: send NDLC_INFO_FRAME to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_INFO_FRAME, Len: 30, sid: 8A91E8, FC 0xC1
```

Table 2-89 describes the fields and messages shown in Figure 2-187.

**Table 2-89 Debug NCIA Circuit Event Field Descriptions**

Field	Description
IN	Incoming message from client.
OUT	Outgoing message to client.
Ver_Id	NDLC version ID.
MsgType	NDLC message type.
Len	NDLC message length.
tmac	Target MAC.
tsap	Target SAP.
csap	Client SAP.
oid	Origin ID.
tid	Target ID
lfs	Largest frame size flag.
ws	Window size.
saddr	Source MAC address.
ssap	Source SAP.
daddr	Destination MAC address.
dsap	Destination SAP.
sid	Session ID.
FC	Flow control flag.

In the following messages, an NDLC\_START\_DL messages is received from a client. to start a data link session:

```
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_START_DL, Len: 24, tmac: 4000.1060.1000,
         tsap: 4, csap 8, oid: 8A91E8, tid 0, lfs 16, ws 1
NCIA: create circuit: saddr 4000.1060.1000, ssap 4, daddr 4000.3000.0003, dsap 8 sid:
      8B09A8
```

The next two messages indicate that an NDLC\_DL\_STARTED message is sent to a client. The server informs the client that a data link session is started.

```
NCIA: send NDLC_DL_STARTED to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_DL_STARTED, Len: 2,4 tmac: 4000.1060.1000,
          tsap: 4, csap 8, oid: 8A91E8, tid 8B09A8, lfs 16, ws 1
```

In the following two messages, an NDLC\_XID\_FRAME message is received from a client, and the client starts an XID exchange:

```
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_XID_FRAME, Len: 12, sid: 8B09A8, FC 0x81
NCIA: send NDLC_XID_FRAME to client 10.2.20.3 for ckt: 8B09A8
```

In the following two messages, an NDLC\_XID\_FRAME message is sent from a client, and an DLC\_XID\_FRAME message is received from a client:

```
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_XID_FRAME, Len: 12, sid: 8A91E8, FC 0xC1
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_XID_FRAME, Len: 18, sid: 8B09A8, FC 0xC1
```

The next two messages show that an NDLC\_CONTACT\_STN message is sent to a client:

```
NCIA: send NDLC_CONTACT_STN to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_CONTACT_STN, Len: 12, sid: 8A91E8, FC 0xC1
```

In the following message, an NDLC\_STN\_CONTACTED message is received from a client. The client informs server that the station has been contacted.

```
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_STN_CONTACTED, Len: 12, sid: 8B09A8, FC 0xC1
```

In the last two messages, an NDLC\_INFO\_FRAME is sent to a client, and the server sends data to the client:

```
NCIA: send NDLC_INFO_FRAME to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_INFO_FRAME, Len: 30, sid: 8A91E8, FC 0xC1
```

Figure 2-188 shows sample **debug ncia circuit flow-control** output. In this example, the flow control in a session start-up sequence is displayed.

### Figure 2-188 Sample Debug NCIA Circuit Flow-Control Output

```
Router# debug ncia circuit flow-control

NCIA: no flow control in NDLC_DL_STARTED frame
NCIA: receive Increment Window Op for circuit 8ADE00
NCIA: ncia_flow_control_in FC 0x81, IW 1 GP 2 CW 2, Client IW 1 GP 0 CW 1
NCIA: grant client more packet by sending Repeat Window Op
NCIA: ncia_flow_control_out FC: 0xC1, IW 1 GP 2 CW 2, Client IW 1 GP 2 CW 2
NCIA: receive FCA for circuit 8ADE00
NCIA: receive Increment Window Op for circuit 8ADE00
NCIA: ncia_flow_control_in FC 0xC1, IW 1 GP 5 CW 3, Client IW 1 GP 2 CW 2
NCIA: grant client more packet by sending Repeat Window Op
NCIA: ncia_flow_control_out FC: 0xC1, IW 1 GP 5 CW 3, Client IW 1 GP 5 CW 3
NCIA: receive FCA for circuit 8ADE00
NCIA: receive Increment Window Op for circuit 8ADE00
NCIA: ncia_flow_control_in FC 0xC1, IW 1 GP 9 CW 4, Client IW 1 GP 5 CW 3
```

```

NCIA: grant client more packet by sending Repeat Window Op
NCIA: ncia_flow_control_out FC: 0xC1, IW 1 GP 8 CW 4, Client IW 1 GP 9 CW 4
NCIA: reduce ClientGrantPacket by 1 (Granted: 8)
NCIA: receive FCA for circuit 8ADE00
NCIA: receive Increment Window Op for circuit 8ADE00
    
```

Table 2-90 describes the important fields shown in Figure 2-188.

**Table 2-90 Debug NCIA Circuit Flow-Control Field Descriptions**

Field	Description
IW	Initial window size.
GP	Granted packet number.
CW	Current window size.

Figure 2-189 shows sample **debug ncia circuit state** output. In this example, a session start-up sequence is displayed.

**Figure 2-189 Sample Debug NCIA Circuit State Output**

```

Router# debug ncia circuit state

NCIA: pre-server fsm: event CONN_OPENED
NCIA: pre-server fsm: event NDLC_PRIMITIVES
NCIA: server event: WAN - STDL state: CLSOED
NCIA: ncia server fsm action 32
NCIA: circuit state: CLOSED -> START_DL_RCVD
NCIA: server event: DLU - TestStn.Rsp state: START_DL_RCVD
NCIA: ncia server fsm action 17
NCIA: circuit state: START_DL_RCVD -> DL_STARTED_SND
NCIA: pre-server fsm: event NDLC_PRIMITIVES
NCIA: server event: WAN - XID state: DL_STARTED_SND
NCIA: ncia server fsm action 33
NCIA: circuit state: DL_STARTED_SND -> DL_STARTED_SND
NCIA: server event: DLU - ReqOpnStn Req state: DL_STARTED_SND
NCIA: ncia server fsm action 33
NCIA: circuit state: DL_STARTED_SND -> OPENED
NCIA: server event: DLU - Id.Rsp state: OPENED
NCIA: ncia server fsm action 11
NCIA: circuit state: OPENED -> OPENED
NCIA: pre-server fsm: event NDLC_PRIMITIVES
NCIA: server event: WAN - XID state: OPENED
NCIA: ncia server fsm action 33
NCIA: circuit state: OPENED -> OPENED
NCIA: server event: DLU - Connect.Req state: OPENED
NCIA: ncia server fsm action 6
NCIA: circuit state: OPENED -> CONNECT_PENDING
NCIA: pre-server fsm: event NDLC_PRIMITIVES
NCIA: server event: WAN - CONR state: CONNECT_PENDING
NCIA: ncia server fsm action 33 --> CLS_CONNECT_CNF sets NciaClsBusy
NCIA: circuit state: CONNECT_PENDING -> CONNECTED
NCIA: server event: DLU - Flow.Req (START) state: CONNECTED
NCIA: ncia server fsm action 25 --> unset NciaClsBusy
NCIA: circuit state: CONNECTED -> CONNECTED
NCIA: server event: DLU - Data.Rsp state: CONNECTED
NCIA: ncia server fsm action 8
NCIA: circuit state: CONNECTED -> CONNECTED
    
```

Table 2-91 describes the important fields shown in Figure 2-189.

**Table 2-91      Debug NCIA Circuit State Field Descriptions**

<b>Field</b>	<b>Description</b>
WAN	Event from WAN (client).
DLU	Event from upstream module—dependent logical unit (DLU).
ADMIN	Administrative event.
TIMER	Timer event.

#### Related Commands

**debug dlsw**

**debug ncia client**

**debug ncia server**

## debug ncia client

Use the **debug ncia client** EXEC command to display debug information for all native client interface architecture (NCIA) client processing that occurs in the router. The **no** form of this command disables debugging output.

```
[no] debug ncia client [ip-address | error [ip-address] | event [ip-address] | message [ip-address]]
```

### Syntax Description

<i>ip-address</i>	(Optional) Remote client IP address.
<b>error</b>	(Optional) Triggers the recording of messages only when errors occur. The current state and event of a NCIA client are normally included in the message. If you do not specify an IP address, the error messages are logged for all active clients.
<b>event</b>	(Optional) Triggers the recording of messages that describe the current state and event—and sometimes the action that just completed—for the NCIA client. If you do not specify an IP address, the messages are logged for all active clients.
<b>message</b>	(Optional) Triggers the recording of messages that contain up to the first 32 bytes of data in a TCP packet sent to or received from an NCIA client. If you do not specify an IP address, the messages are logged for all active clients.

### Usage Guidelines

NCIA is an architecture developed by Cisco for accessing SNA applications. This architecture allows native SNA interfaces on hosts and clients to access TCP/IP backbones.

Use the **debug ncia client event** command to determine the sequences of activities that occur while a NCIA client is in different processing states.

Use the **debug ncia client error** command to see only certain error conditions that occur.

Use the **debug ncia client message** command to see only the first 32 bytes of data in a TCP packet sent to or received from an NCIA client.

The **debug ncia client** command can be used in conjunction with the **debug ncia server** and **debug ncia circuit** commands to get a complete picture of NCIA activity.

### Sample Display

Figure 2-190 shows sample **debug ncia circuit** output. Following the example is a description of each sample output message.

**Figure 2-190 Sample Debug NCIA Circuit Output**

```
Router# debug ncia client
NCIA: Passive open 10.2.20.123(1088) -> 1973
NCIA: index for client hash queue is 27
NCIA: number of element in client hash queue 27 is 1
```

```
NCIA: event PASSIVE_OPEN, state NCIA_CLOSED for client 10.2.20.123
NCIA: Rcvd msg type NDLC_CAP_XCHG in tcp packet for client 10.2.20.123
NCIA: First 17 byte of data rcvd: 8112001100000000000000400050104080C
NCIA: Sent msg type NDLC_CAP_XCHG in tcp packet to client 10.2.20.123
NCIA: First 17 byte of data sent: 811200111000000010000400050104080C
NCIA: event CAP_CMD_RCVD, state NCIA_CAP_WAIT, for client 10.2.20.123, cap xchg cmd sent
NCIA: Rcvd msg type NDLC_CAP_XCHG in tcp packet for client 10.2.20.123
NCIA: First 17 byte of data rcvd: 811200111000000010000000050104080C
NCIA: event CAP_RSP_RCVD, state NCIA_CAP_NEG for client 10.2.20.123

NCIA: Rcvd msg type NDLC_PEER_TEST_REQ in tcp packet for client 10.2.20.123
NCIA: First 4 byte of data rcvd: 811D0004
NCIA: event KEEPALIVE_RCVD, state NCIA_OPENED for client 10.2.20.123
NCIA: Sent msg type NDLC_PEER_TEST_RSP in tcp packet to client 10.2.20.123
NCIA: First 4 byte of data sent: 811E0004IA

NCIA: event TIME_OUT, state NCIA_OPENED, for client 10.2.20.123, keepalive_count = 0
NCIA: Sent msg type NDLC_PEER_TEST_REQ, in tcp packet to client 10.2.20.123
NCIA: First 4 byte of data sent: 811D0004
NCIA: Rcvd msg type NDLC_PEER_TEST_RSP in tcp packet for client 10.2.20.123
NCIA: First 4 byte of data rcvd: 811E0004
NCIA: event KEEPALIVE_RSP_RCVD, state NCIA_OPENED for client 10.2.20.123

NCIA: Error, event PASIVE_OPEN, state NCIA_OPENED, for client 10.2.20.123, should not
have occurred.
NCIA: Error, active_open for pre_client_fsm while client 10.2.20.123 is active or not
configured, registered.
```

Messages in lines 1 through 12 show the events that occur when a client connects to the router (the NCIA server). These messages show a passive\_open process.

Messages in lines 13 to 17 show the events that occur when a TIME\_OUT event is detected by a client PC workstation. The workstation sends an NDLC\_PEER\_TEST\_REQ message to the NCIA server, and the router responds with an NDLC\_PEER\_TEST\_RSP message.

Messages in lines 18 to 23 show the events that occur when a TIME\_OUT event is detected by the router (the NCIA server). The router sends an NDLC\_PEER\_TEST\_REQ message to the client PC workstation, and the PC responds with an NDLC\_PEER\_TEST\_RSP message.

When you use the **debug ncia client message** command, the messages shown on lines 6, 8, 11, 14, 17, 20, and 22 are output in addition to other messages not shown in this example.

When you use the **debug ncia client error** command, the messages shown on lines 24 and 25 are output in addition to other messages not shown in this example.

## Related Commands

**debug ncia circuit**

**debug ncia server**

## debug ncia server

Use the **debug ncia server EXEC** command to display debug information for the native client interface architecture (NCIA) server and its upstream software modules. The **no** form of this command disables debugging output.

**[no] debug ncia server**

### Usage Guidelines

NCIA is an architecture developed by Cisco for accessing SNA applications. This architecture allows native SNA interfaces on hosts and clients to access TCP/IP backbones.

The **debug ncia server** command displays all Cisco Link Services (CLS) messages between the NCIA server and its upstream modules, such as data-link switching (DLSw) and downstream physical units (DSPUs). Use this command when a problem exists between the NCIA server and other software modules within the router.

You cannot enable debugging output for a particular client or particular circuit.

### Sample Display

Figure 2-191 shows sample **debug ncia server** output. In this example, a session start-up sequence is displayed. Following the example is a description of each group of sample output messages.

**Figure 2-191 Sample Debug NCIA Server Output**

```
Router# debug ncia server

NCIA: send CLS_TEST_STN_IND to DLU
NCIA: Receive TestStn.Rsp
NCIA: send CLS_ID_STN_IND to DLU
NCIA: Receive ReqOpnStn Req
NCIA: send CLS_REQ_OPNSTN_CNF to DLU
NCIA: Receive Id.Rsp
NCIA: send CLS_ID_IND to DLU
NCIA: Receive Connect Req
NCIA: send CLS_CONNECT_CNF to DLU
NCIA: Receive Flow Req
NCIA: Receive Data Req
NCIA: send CLS_DATA_IND to DLU
NCIA: send CLS_DISC_IND to DLU
NCIA: Receive Disconnect.Rsp
```

In the following messages, the client is sending a test message to the host and the test message is received by the host:

```
NCIA: send CLS_TEST_STN_IND to DLU
NCIA: Receive TestStn.Rsp
```

In the next message, the server is sending an XID message to the host.

```
NCIA: send CLS_ID_STN_IND to DLU
```

In the next two messages, the host opens the station and the server responds.

```
NCIA: Receive ReqOpnStn Req
NCIA: send CLS_REQ_OPNSTN_CNF to DLU
```

In the following two messages, the client is performing an XID exchange with the host:

```
NCIA: Receive Id.Rsp
NCIA: send CLS_ID_IND to DLU
```

In the next group of messages, the host attempts to establish a session with the client.

```
NCIA: Receive Connect.Req
NCIA: send CLS_CONNECT_CNF to DLU
NCIA: Receive Flow.Req
```

In the next two messages, the host sends data to the client.

```
NCIA: Receive Data.Req
NCIA: send CLS_DATA_IND to DLU
```

In the last two messages, the client closes the session.

```
NCIA: send CLS_DISC_IND to DLU
NCIA: Receive Disconnect.Rsp
```

### Related Commands

```
debug dlsw
debug ncia circuit
debug ncia client
```

## debug netbios error

Use the **debug netbios error** EXEC command to display information about Network Basic Input/Output System (NetBIOS) protocol errors. The **no** form of this command disables debugging output.

**[no] debug netbios error**

### Usage Guidelines

For complete information on the NetBIOS process, use the **debug netbios packet** command along with the **debug netbios error** command.

### Sample Display

Figure 2-192 shows sample **debug netbios error** output. This example shows that an illegal packet has been received on the async interface.

**Figure 2-192 Sample Debug NetBIOS Error Output**

```
Router# debug netbios error
Async1 nbf Bad packet
```

### Related Commands

**debug netbios-name-cache**  
**debug netbios packet**

## debug netbios-name-cache

Use the **debug netbios-name-cache** EXEC command to display name caching activities on a router. The **no** form of this command disables debugging output.

**[no] debug netbios-name-cache**

### Usage Guidelines

Examine the display to diagnose problems in NetBIOS name caching.

### Sample Display

Figure 2-193 illustrates a collection of sample **debug netbios-name-cache** output listings.

**Figure 2-193 Sample Debug NetBIOS-Name-Cache Output**

```
Router# debug netbios-name-cache

NETBIOS: L checking name ORINDA, vrn=0
NETBIOS name cache table corrupted at offset 13
NETBIOS name cache table corrupted at later offset, at location 13
NETBIOS: U chk name=ORINDA, addr=1000.4444.5555, idb=TR1, vrn=0, type=1
NETBIOS: U upd name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0,type=1
NETBIOS: U add name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0,type=1
NETBIOS: U no memory to add cache entry. name=ORINDA,addr=1000.4444.5555
NETBIOS: Invalid structure detected in netbios_name_cache_ager
NETBIOS: flushed name=ORINDA, addr=1000.4444.5555
NETBIOS: expired name=ORINDA, addr=1000.4444.5555
NETBIOS: removing entry. name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0
NETBIOS: Tossing ADD_NAME/STATUS/NAME/ADD_GROUP frame
NETBIOS: Lookup Failed -- not in cache
NETBIOS: Lookup Worked, but split horizon failed
NETBIOS: Could not find RIF entry
NETBIOS: Cannot duplicate packet in netbios_name_cache_proxy
```

**Note** The sample display in Figure 2-193 is a composite output. Debugging output that you actually see would not necessarily occur in this sequence.

Table 2-92 describes selected output fields shown in Figure 2-193.

**Table 2-92 Debug NetBIOS-Name-Cache Field Descriptions**

Field	Description
NETBIOS	This is a NetBIOS name caching debugging output.
L, U	L means lookup; U means update.
addr=1000.4444.5555	MAC address 1000.4444.5555 of machine being looked up in NetBIOS name cache.
idb=TR1	Indication that name of machine was learned from Token Ring interface number 1; idb translates into interface data block.

**Table 2-92 Debug NetBIOS-Name-Cache Field Descriptions (Continued)**

Field	Description
vrn=0	Router determined that the packet comes from virtual ring number 0; this packet actually comes from a real Token Ring interface, because virtual ring number 0 is not valid.
type=1	The type field indicates the way that the router learned about the specified machine. The possible values for type are as follows: <ul style="list-style-type: none"> <li>• 1 = Learned from traffic</li> <li>• 2 = Learned from a remote peer</li> <li>• 4, 8 = Statically entered via the router's configuration</li> </ul>

The following discussion briefly outlines each line shown in the example provided in Figure 2-193.

With the first line of output, the router declares that it has examined the NetBIOS name cache table for the machine name ORINDA and that the packet that prompted the lookup came from virtual ring 0. In this case, this packet comes from a real interface—virtual ring number 0 is not valid.

```
NETBIOS: L checking name ORINDA, vrn=0
```

The following two lines indicate that an invalid NetBIOS entry exists and that the corrupted memory was detected. The invalid memory will be removed from the table; no action is needed.

```
NETBIOS name cache table corrupted at offset 13
NETBIOS name cache table corrupted at later offset, at location 13
```

The following line indicates that the router attempted to check the NetBIOS cache table for the name ORINDA with MAC address 1000.4444.5555. This name was obtained from Token Ring interface 1. The type field indicates that the name was learned from traffic.

```
NETBIOS: U chk name=ORINDA, addr=1000.4444.5555, idb=TR1, vrn=0, type=1
```

The following line indicates that the NetBIOS name ORINDA is in the name cache table and was updated to the current value:

```
NETBIOS: U upd name=ORINDA, addr=1000.4444.5555, idb=TR1, vrn=0, type=1
```

The following line indicates that the NetBIOS name ORINDA is not in the table and must be added to the table:

```
NETBIOS: U add name=ORINDA, addr=1000.4444.5555, idb=TR1, vrn=0, type=1
```

The following line indicates that there was insufficient cache buffer space when the router tried to add this name:

```
NETBIOS: U no memory to add cache entry. name=ORINDA, addr=1000.4444.5555
```

The following line indicates that the NetBIOS ager detects an invalid memory in the cache. The router clears the entry; no action is needed.

```
NETBIOS: Invalid structure detected in netbios_name_cache_ager
```

The following line indicates that the entry for ORINDA was flushed from the cache table:

```
NETBIOS: flushed name=ORINDA, addr=1000.4444.5555
```

The following line indicates that the entry for ORINDA timed out and was flushed from the cache table:

```
NETBIOS: expired name=ORINDA, addr=1000.4444.5555
```

The following line indicates that the router removed the ORINDA entry from its cache table:

```
NETBIOS: removing entry. name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0
```

The following line indicates that the router discarded a NetBIOS packet of type ADD\_NAME, STATUS, NAME\_QUERY, or ADD\_GROUP. These packets are discarded when multiple copies of one of these packet types are detected during a certain period of time.

```
NETBIOS: Tossing ADD_NAME/STATUS/NAME/ADD_GROUP frame
```

The following line indicates that the system could not find a NetBIOS name in the cache:

```
NETBIOS: Lookup Failed -- not in cache
```

The following line indicates that the system found the destination NetBIOS name in the cache, but located on the same ring from which the packet came. The router will drop this packet because the packet should not leave this ring.

```
NETBIOS: Lookup Worked, but split horizon failed
```

The following line indicates that the system found the NetBIOS name in the cache, but the router could not find the corresponding RIF. The packet will be sent as a broadcast frame.

```
NETBIOS: Could not find RIF entry
```

The following line indicates that no buffer was available to create a NetBIOS name-cache proxy. A proxy will not be created for the packet, which will be forwarded as a broadcast frame.

```
NETBIOS: Cannot duplicate packet in netbios_name_cache_proxy
```

## Related Commands

**debug netbios error**

**debug netbios packet**

## debug netbios packet

Use the **debug netbios packet** EXEC command to display general information about NetBIOS packets. The **no** form of this command disables debugging output.

**[no] debug netbios packet**

### Usage Guidelines

For complete information on the NetBIOS process, use the **debug netbios error** command along with the **debug netbios packet** command.

### Sample Display

Figure 2-194 shows sample **debug netbios packet** and **debug netbios error** output. This example shows the LLC header for an asynchronous interface followed by the NetBIOS information. For additional information on the NetBIOS fields, refer to *IBM LAN Technical Reference IEEE 802.2*.

**Figure 2-194 Sample Debug NetBIOS Packet Output**

```
Router# debug netbios packet

Async1 (i) U-format UI C_R=0x0
(i) NETBIOS_ADD_NAME_QUERY
  Resp_correlator= 0x6F 0x0
  Src name=CS-NT-1

Async1 (i) U-format UI C_R=0x0
(i) NETBIOS_ADD_GROUP_QUERY
  Resp_correlator= 0x6F 0x0
  Src name=COMMSERVER-WG

Async1 (i) U-format UI C_R=0x0
(i) NETBIOS_ADD_NAME_QUERY
  Resp_correlator= 0x6F 0x0
  Src name=CS-NT-1

Ethernet0 (i) U-format UI C_R=0x0
(i) NETBIOS_DATAGRAM
  Length= 0x2C 0x0
  Dest name=COMMSERVER-WG
  Src name=CS-NT-3
```

### Related Commands

**debug netbios error**

**debug netbios-name-cache**

## debug nhrp

Use the **debug nhrp** EXEC command to display information about Next Hop Resolution Protocol (NHRP) activity. The **no** form of this command disables debugging output.

**[no] debug nhrp**

### Usage Guidelines

Use this command when some nodes on a TCP/IP or IPX network are not responding. It shows whether the router is sending or receiving NHRP packets.

### Sample Display

Figure 2-195 shows sample **debug nhrp** output.

**Figure 2-195 Sample Debug NHRP Output**

```
Router# debug nhrp

NHRP: Cache update 172.19.145.57 None
NHRP: Sent request src 172.19.145.56 dst 255.255.255.255
NHRP M: id 0 src 172.19.145.56 dst 172.19.145.57
NHRP: Encapsulation succeeded. MAC addr ffff.ffff.ffff.
NHRP: O 86 bytes out Ethernet1 dest 255.255.255.255
NHRP: Recv reply Size 64
NHRP M: id 0 src 172.19.145.56 dst 172.19.145.57
NHRP: Cache update 172.19.145.57 0000.0c14.59d3.
```

Table 2-93 describes the fields shown in the display.

**Table 2-93 Debug NHRP Field Descriptions**

Field	Descriptions
NHRP and NHRP M	NHRP debugging output and mandatory header debugging output.
Cache update	NHRP cache is being revised.
Sent request src dst	NHRP request packet was sent from the specified source address. NHRP packet was sent to the specified destination address.
id	Sequence number of the packet.
src	Sequence number of the source address.
dst	Sequence number of the destination address.
Encapsulation succeeded. MAC addr	NHRP packet was successfully encapsulated. Link layer address used as the destination address for the NHRP packet.
O 86 bytes out Ethernet1 dest	Size of the NHRP packet (in this case, the output was 86 bytes). Interface that the packet was sent out on, and the network layer destination address.
Recv reply Size	Indicates receipt of an NHRP reply packet and the size of the packet excluding the link layer header.

Related Commands

**debug nhrp options**

**debug nhrp rate**

## debug nhrp options

Use the **debug nhrp options** EXEC command to display information about NHRP option processing. The **no** form of this command disables debugging output.

**[no] debug nhrp options**

### Usage Guidelines

Use this command to show you whether there are problems or error situations with NHRP option processing (for example, unknown options).

### Sample Display

Figure 2-196 shows sample **debug nhrp options** output.

**Figure 2-196 Sample Debug NHRP Options Output**

```
Router# debug nhrp options

NHRP-OPT: MASK 4
NHRP-OPT-MASK: FFFFFFFF
NHRP-OPT: NETID 4
NHRP-OPT: RESPONDER 4
NHRP-OPT: RECORD 0
NHRP-OPT: RRECORD 0
```

Table 2-94 describes the fields shown in Figure 2-196.

**Table 2-94 Debug NHRP Options Field Descriptions**

Field	Descriptions
NHRP-OPT	NHRP options debugging output.
MASK 4	Number of bytes of information in the destination prefix option.
NHRP-OPT-MASK	Contents of the destination prefix option.
NETID	Number of bytes of information in the subnetwork identifier option.
RESPONDER	Number of bytes of information in the responder address option.
RECORD	Forward record option.
RRECORD	Reverse record option.

### Related Commands

**debug nhrp**  
**debug nhrp rate**

## debug nhrp rate

Use the **debug nhrp rate** EXEC command to display information about NHRP traffic rate limits. The **no** form of this command disables debugging output.

**[no] debug nhrp rate**

### Usage Guidelines

Use this command to verify that the traffic is consistent with the setting of the NHRP commands (such as **ip nhrp use** and **ip max-send** commands).

### Sample Display

Figure 2-197 shows sample **debug nhrp rate** output.

**Figure 2-197 Sample Debug NHRP Rate Output**

```
Router# debug nhrp rate

NHRP-RATE: Sending initial request
NHRP-RATE: Retransmitting request (retrans ivl 2)
NHRP-RATE: Retransmitting request (retrans ivl 4)
NHRP-RATE: Ethernet1: Used 3
```

Table 2-95 describes the fields shown in Figure 2-197.

**Table 2-95 Debug NHRP Rate Field Descriptions**

Field	Descriptions
NHRP-RATE	NHRP rate debugging output.
Sending initial request	First time an attempt was made to send an NHRP packet to a particular destination.
Retransmitting request	Indicates that the NHRP packet was retransmitted, and shows the time interval (in seconds) to wait before the NHRP packet is retransmitted again.
Ethernet1:	Interface over which the NHRP packet was transmitted.
Used 3	Number of packets sent out of the default maximum 5 (in this case, 3 were sent).

### Related Commands

**debug nhrp**  
**debug nhrp options**

## debug packet

Use the **debug packet** EXEC command to display information on packets that the network can not classify. The **no** form of this command disables debugging output.

**[no] debug packet**

### Sample Display

Figure 2-198 shows sample **debug packet** output.

**Figure 2-198 Sample Debug Packet Output**

```
Router# debug packet

Ethernet0: Unknown ARPA, src 0000.0c00.6fa4, dst ffff.ffff.ffff, type 0x0a0
data 00000c00f23a00000c00ab45, len 60
Serial3: Unknown HDLC, size 64, type 0xaaaa, flags 0x0F00
Serial2: Unknown PPP, size 128
Serial7: Unknown FRAME-RELAY, size 174, type 0x5865, DLCI 7a
Serial0: compressed TCP/IP packet dropped
```

Table 2-96 describes significant fields shown in Figure 2-198.

**Table 2-96 Debug Packet Field Descriptions**

Field	Description
Ethernet0	Name of the Ethernet interface that received the packet.
Unknown	The network could not classify this packet. Examples include packets with unknown link types.
ARPA	<p>This packet uses ARPA-style encapsulation. Possible encapsulation styles vary depending on the media command mode (MCM) and encapsulation style, as follows:</p> <p><b>Ethernet (MCM)—Encapsulation Style</b></p> <ul style="list-style-type: none"> <li>• APOLLO</li> <li>• ARP</li> <li>• ETHERTALK</li> <li>• ISO1</li> <li>• ISO3</li> <li>• LLC2</li> <li>• NOVELL-ETHER</li> <li>• SNAP</li> </ul> <p><b>FDDI (MCM)—Encapsulation Style</b></p> <ul style="list-style-type: none"> <li>• APOLLO</li> <li>• ISO1</li> <li>• ISO3</li> <li>• LLC2</li> <li>• SNAP</li> </ul>

**Table 2-96 Debug Packet Field Descriptions (Continued)**

Field	Description
	<b>Frame Relay—Encapsulation Style</b> <ul style="list-style-type: none"> <li>• BRIDGE</li> <li>• FRAME-RELAY</li> </ul>
	<b>Serial (MCM)—Encapsulation Style</b> <ul style="list-style-type: none"> <li>• BFEX25</li> <li>• BRIDGE</li> <li>• DDN-X25</li> <li>• DDNX25-DCE</li> <li>• ETHERTALK</li> <li>• FRAME-RELAY</li> <li>• HDLC</li> <li>• HDH</li> <li>• LAPB</li> <li>• LAPBDCE</li> <li>• MULTI-LAPB</li> <li>• PPP</li> <li>• SDLC-PRIMARY</li> <li>• SDLC-SECONDARY</li> <li>• SLIP</li> <li>• SMDS</li> <li>• STUN</li> <li>• X25</li> <li>• X25-DCE</li> </ul>
	<b>Token Ring (MCM)—Encapsulation Style</b> <ul style="list-style-type: none"> <li>• 3COM-TR</li> <li>• ISO1</li> <li>• ISO3</li> <li>• MAC</li> <li>• LLC2</li> <li>• NOVELL-TR</li> <li>• SNAP</li> <li>• VINES-TR</li> </ul>
src 0000.0c00.6fa4	MAC address of the node generating the packet.
dst.ffff.ffff.ffff	MAC address of the destination node for the packet.
type 0x0a0	Packet type.
data...	First 12 bytes of the datagram following the MAC header.
len 60	Length of the message in bytes that the interface received from the wire.
size 64	Length of the message in bytes that the interface received from the wire. Equivalent to the len field.
flags 0x0F00	HDLC or PP flags field.

**Table 2-96**      **Debug Packet Field Descriptions (Continued)**

<b>Field</b>	<b>Description</b>
DLCI 7a	The DLCI number on Frame Relay.
compressed TCP/IP packet dropped	This message can occur when TCP header compression is enabled on an interface and the packet does not turn out to be HDLC or X25 after classification.

## debug ppp

Use the **debug ppp** EXEC command to display information on traffic and exchanges in an internetwork implementing the Point-to-Point Protocol (PPP). The **no** form of this command disables debugging output.

**[no] debug ppp {packet | negotiation | error | authentication}**

### Syntax Description

<b>packet</b>	Causes the <b>debug ppp</b> command to display PPP packets being sent and received. (This command displays low-level packet dumps.)
<b>negotiation</b>	Causes the <b>debug ppp</b> command to display PPP packets transmitted during PPP startup, where PPP options are negotiated.
<b>error</b>	Causes the <b>debug ppp</b> command to display protocol errors and error statistics associated with PPP connection negotiation and operation.
<b>authentication</b>	Causes the <b>debug ppp</b> command to display authentication protocol messages, including Challenge Authentication Protocol (CHAP) packet exchanges and Password Authentication Protocol (PAP) exchanges.

### Usage Guidelines

Use the **debug ppp** commands when trying to find the following:

- The Network Control Protocols (NCPs) that are supported on either end of a PPP connection
- Any loops that might exist in a PPP internetwork
- Nodes that are (or are not) properly negotiating PPP connections
- Errors that have occurred over the PPP connection
- Causes for CHAP session failures
- Causes for PAP session failures

Refer to Internet RFCs 1331, 1332, and 1333 for details concerning PPP-related nomenclature and protocol information.

### Sample Displays

Figure 2-199 shows sample **debug ppp packet** output as seen from the Link Quality Monitor (LQM) side of the connection. This display example depicts packet exchanges under normal PPP operation.

**Figure 2-199 Sample Debug PPP Packet Output**

```
Router# debug ppp packet

PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
```

```

PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 3 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 3 len = 12
PPP Serial4: O LCP ECHOREP(A) id 3 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 4 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 4 len = 12
PPP Serial4: O LCP ECHOREP(A) id 4 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 5 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 5 len = 12
PPP Serial4: O LCP ECHOREP(A) id 5 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 6 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 6 len = 12
PPP Serial4: O LCP ECHOREP(A) id 6 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 7 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 7 len = 12
PPP Serial4: O LCP ECHOREP(A) id 7 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48

```

Table 2-97 describes significant fields shown in Figure 2-199.

**Table 2-97 Debug PPP Packet Field Descriptions**

Field	Description
PPP	This is PPP debugging output.
Serial4	Interface number associated with this debugging information.
(o), O	This packet was detected as an output packet.
(i), I	This packet was detected as an input packet.
lcp_slqr()	Procedure name; running LQM, send a Link Quality Report (LQR).
lcp_rlqr()	Procedure name; running LQM, received an LQR.
input (C021)	The router received a packet of the specified packet type (in hexadecimal). A value of C025 indicates packet of type LQM.
state = OPEN	PPP state; normal state is OPEN.
magic = D21B4	Magic Number for indicated node; when output is indicated, this is the Magic Number of the node on which debugging is enabled. The actual Magic Number depends on whether the packet detected is indicated as I or O.
datagramsize = 52	Packet length including header.
code = ECHOREQ(9)	Code identifies the type of packet received. Both forms of the packet, string and hexadecimal, are presented.

**Table 2-97 Debug PPP Packet Field Descriptions (Continued)**

Field	Description
len = 48	Packet length without header.
id = 3	ID number per Link Control Protocol (LCP) packet format.
pkt type 0xC025	Packet type in hexadecimal; typical packet types are C025 for LQM and C021 for LCP.
LCP ECHOREQ (9)	Echo Request; value in parentheses is the hexadecimal representation of the LCP type.
LCP ECHOREP (A)	Echo Reply; value in parentheses is the hexadecimal representation of the LCP type.

To elaborate on the displayed output, consider the partial exchange in Figure 2-200. This sequence shows that one side is using ECHO for its keepalives and the other side is using LQRs.

**Figure 2-200 Sample Debug PPP Packet Output—Keepalive Messages**

```
Router# debug ppp packet

PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt_type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt_type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 3 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 3 len = 12
PPP Serial4: O LCP ECHOREP(A) id 3 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
```

Explanations for each line of Figure 2-200 follow.

The first line states that the router with debugging enabled has sent an LQR to the other side of the PPP connection:

```
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
```

The next two lines indicate that the router has received a packet of type C025 (LQM) and provides details about the packet:

```
PPP Serial4(i): pkt_type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
```

The next two lines indicate that the router received an ECHOREQ of type C021 (LCP). The other side is sending ECHOs. The router on which debugging is configured for LQM but also responds to ECHOs.

```
PPP Serial4(i): pkt_type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 3 (C) magic D3454
```

Next, the router is detected to have responded to the ECHOREQ with an ECHOREP and is preparing to send out an LQR:

```
PPP Serial4: O LCP ECHOREP(A) id 3 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
```

Figure 2-201 shows sample **debug ppp negotiation** output. This is a normal negotiation, where both sides agree on network control program (NCP) parameters. In this case, protocol type IP is proposed and acknowledged.

**Figure 2-201 Sample Debug PPP Negotiation Output**

```
Router# debug ppp negotiation

ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 3D56CAC
ppp: received config for type = 4 (QUALITYTYPE) acked
ppp: received config for type = 5 (MAGICNUMBER) value = 3D567F8 acked (ok)
PPP Serial4: state = ACKSENT fsm_rconfack(C021): rcvd id 5
ppp: config ACK received, type = 4 (CI_QUALITYTYPE), value = C025
ppp: config ACK received, type = 5 (CI_MAGICNUMBER), value = 3D56CAC
ppp: ipcp_reqci: returning CONFACK.
      (ok)
PPP Serial4: state = ACKSENT fsm_rconfack(8021): rcvd id 4
```

Table 2-98 describes significant fields shown in Figure 2-201.

**Table 2-98 Debug PPP Negotiation Field Descriptions**

Field	Description
ppp	This is PPP debugging output.
sending CONFREQ	The router sent a configuration request.
type = 4 (CI_QUALITYTYPE)	The type of LCP configuration option that is being negotiated and a descriptor. A type value of 4 indicates Quality Protocol negotiation; a type value of 5 indicates Magic Number negotiation.
value = C025/3E8	For Quality Protocol negotiation, indicates NCP type and reporting period. In the example, C025 indicates LQM; 3E8 is a hexadecimal value translating to about 10 seconds (in hundredths of a second).
value = 3D56CAC	For Magic Number negotiation, indicates the Magic Number being negotiated.
received config	The receiving node has received the proposed option negotiation for the indicated option type.
acked	Acknowledgment and acceptance of options.
state = ACKSENT	Specific PPP state in the negotiation process.
ipcp_reqci	IPCP notification message; sending CONFACK.
fsm_rconfack (8021)	The procedure fsm_rconfack processes received CONFACKs, and the protocol (8021) is IP.

Explanations for each line in Figure 2-201 follow.

The first two lines in Figure 2-201 indicate that the router is trying to bring up LCP and will use the indicated negotiation options (Quality Protocol and Magic Number). The value fields are the values of the options themselves. C025/3E8 translates to Quality Protocol LQM. 3E8 is the reporting period (in hundredths of a second). 3D56CAC is the value of the Magic Number for the router.

```
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 3D56CAC
```

The next two lines indicate that the other side negotiated for options 4 and 5 as requested and acknowledged both. If the responding end does not support the options, a CONFREJ is sent by the responding node. If the responding end does not accept the value of the option, a CONFNAK is sent with the value field modified.

```
ppp: received config for type = 4 (QUALITYTYPE) acked
ppp: received config for type = 5 (MAGICNUMBER) value = 3D567F8 acked (ok)
```

The next three lines indicate that the router received a CONFACK from the responding side and displays accepted option values. Use the rcvd id field to verify that the CONFREQ and CONFACK have the same id field.

```
PPP Serial4: state = ACKSENT fsm_rconfack(C021): rcvd id 5
ppp: config ACK received, type = 4 (CI_QUALITYTYPE), value = C025
ppp: config ACK received, type = 5 (CI_MAGICNUMBER), value = 3D56CAC
```

The next line indicates that the router has IP routing enabled on this interface and that the IPCP NCP negotiated successfully:

```
ppp: ipcp_reqci: returning CONFACK.
```

In the last line, the router's state is listed as ACKSENT.

```
PPP Serial4: state = ACKSENT fsm_rconfack(C021): rcvd id 5\
```

Figure 2-202 shows sample output when **debug ppp packet** and **debug ppp negotiation** output are enabled at the same time.

Figure 2-202 Sample Debug PPP Output—Packet and Negotiation Options Enabled

```

router# debug ppp negotiation
router# debug ppp packet

ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = D4C64
PPP Serial4: O LCP CONFREQ(1) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
MAGICNUMBER (6) 0 13 76 100
PPP Serial4(i): pkt type 0xC021, datagramsize 22
PPP Serial4: I LCP CONFREQ(1) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
MAGICNUMBER (6) 0 13 84 240
PPP Serial4: input(C021) state = REQSENT code = CONFREQ(1) id = 4 len = 18
ppp: received config for type = 4 (QUALITYTYPE) acked
ppp: received config for type = 5 (MAGICNUMBER) value = D54F0 acked
PPP Serial4: O LCP CONFACK(2) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
MAGICNUMBER (6) 0 13 84 240 (ok)
PPP Serial4(i): pkt type 0xC021, datagramsize 22
PPP Serial4: I LCP CONFACK(2) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
MAGICNUMBER (6) 0 13 76 100
PPP Serial4: input(C021) state = ACKSENT code = CONFACK(2) id = 4 len = 18
PPP Serial4: state = ACKSENT fsm_rconfack(C021): rcvd id 4
ppp: config ACK received, type = 4 (CI_QUALITYTYPE), value = C025
ppp: config ACK received, type = 5 (CI_MAGICNUMBER), value = D4C64
ipcp: sending CONFREQ, type = 3 (CI_ADDRESS), Address = 2.1.1.2
PPP Serial4: O IPCP CONFREQ(1) id 3 (10) Type3 (6) 2 1 1 2
PPP Serial4: I IPCP CONFREQ(1) id 3 (10) Type3 (6) 2 1 1 1
PPP Serial4(i): pkt type 0x8021, datagramsize 14
PPP Serial4: input(8021) state = REQSENT code = CONFREQ(1) id = 3 len = 10
ppp Serial4: Negotiate IP address: her address 2.1.1.1 (ACK)
ppp: ipcp_reqci: returning CONFACK.
PPP Serial4: O IPCP CONFACK(2) id 3 (10) Type3 (6) 2 1 1 1 (ok)
PPP Serial4: I IPCP CONFACK(2) id 3 (10) Type3 (6) 2 1 1 2
PPP Serial4: input(8021) state = ACKSENT code = CONFACK(2) id = 3 len = 10
PPP Serial4: state = ACKSENT fsm_rconfack(8021): rcvd id 3
ipcp: config ACK received, type = 3 (CI_ADDRESS), Address = 2.1.1.2
PPP Serial4(o): lcp_slqr() state = OPEN magic = D4C64, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D54F0, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D54F0, len = 48
PPP Serial4(o): lcp_slqr() state = OPEN magic = D4C64, len = 48

```

This field shows a decimal representation of the Magic Number.

This field shows a decimal representation of the NCP value.

This field shows a decimal representation of the reporting period.

This exchange represents a successful PPP negotiation for support of NCP type IPCP.

S2877

Figure 2-203 shows sample **debug ppp negotiation** output when the remote side of the connection is unable to respond to LQM requests.

Figure 2-203 Sample Debug PPP Negotiation Output—No Response Detected

```

Router# debug ppp negotiation

ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010

```

```

ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44C1488

```

Figure 2-204 shows sample output when no response is detected for configuration requests (with both **debug ppp negotiation** and **debug ppp packet** enabled).

**Figure 2-204 Sample Debug PPP Negotiation and Packet Output—No Response Detected**

```

Router# debug ppp negotiation
Router# debug ppp packet

ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44DFDC8
PPP Serial4: O LCP CONFREQ(1) id 14 (12) QUALITYTYPE (8) 192 37 0 0 3 232
MAGICNUMBER (6) 4 77 253 200
ppp: TIMEOUT: Time= 44E0980 State= 3
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44DFDC8
PPP Serial4: O LCP CONFREQ(1) id 15 (12) QUALITYTYPE (8) 192 37 0 0 3 232
MAGICNUMBER (6) 4 77 253 200
ppp: TIMEOUT: Time= 44E1828 State= 3
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44DFDC8
PPP Serial4: O LCP CONFREQ(1) id 16 (12) QUALITYTYPE (8) 192 37 0 0 3 232
MAGICNUMBER (6) 4 77 253 200
ppp: TIMEOUT: Time= 44E27C8 State= 3
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44DFDC8
PPP Serial4: O LCP CONFREQ(1) id 17 (12) QUALITYTYPE (8) 192 37 0 0 3 232
MAGICNUMBER (6) 4 77 253 200
ppp: TIMEOUT: Time= 44E3768 State= 3

```

Figure 2-205 shows sample **debug ppp error** output. These messages might appear when the Quality Protocol option is enabled on an interface that is already running PPP.

**Figure 2-205 Sample Debug PPP Error Output**

```

Router# debug ppp error

PPP Serial3(i): rlqr receive failure. successes = 15
PPP: myrcvdiffp = 159 peerxmitdiffp = 41091
PPP: myrcvdiffo = 2183 peerxmitdiffo = 1714439
PPP: threshold = 25
PPP Serial4(i): rlqr transmit failure. successes = 15
PPP: myxmitdiffp = 41091 peerrcvdiffp = 159
PPP: myxmitdiffo = 1714439 peerrcvdiffo = 2183
PPP: l->OutLQRs = 1 LastOutLQRs = 1
PPP: threshold = 25
PPP Serial3(i): lqr_protrej() Stop sending LQRs.
PPP Serial3(i): The link appears to be looped back.

```

Table 2-99 describes significant fields shown in Figure 2-205.

**Table 2-99 Debug PPP Error Field Descriptions**

Field	Description
PPP	This is PPP debugging output.
Serial3(i)	Interface number associated with this debugging information; indicates that this is an input packet.
rlqr receive failure	The request to negotiate the Quality Protocol option is not accepted.
myrcvdiffp = 159	Number of packets received over the time period.
peerxmitdiffp = 41091	Number of packets sent by the remote node over this period.
myrcvdiffo = 2183	Number of octets received over this period.
peerxmitdiffo = 1714439	Number of octets sent by the remote node over this period.
threshold = 25	The maximum error percentage acceptable on this interface. This percentage is calculated by the threshold value entered in the <b>ppp quality number</b> interface configuration command. A value of $100 - \text{number}$ (100 minus <i>number</i> ) is the maximum error percentage. In this case, a <i>number</i> of 75 was entered. This means that the local router must maintain a minimum 75 percent non-error percentage, or the PPP link will be considered down.
OutLQRs = 1	Local router's current send LQR sequence number.
LastOutLQRs = 1	The last sequence number that the remote node side has seen from the local node.

Figure 2-206 shows sample **debug ppp authentication** output. Use this **debug** command to determine why an authentication fails.

**Figure 2-206 Sample Debug PPP Authentication Output**

```
Router# debug ppp authentication

Serial0: Unable to authenticate. No name received from peer
Serial0: Unable to validate CHAP response. USERNAME pioneer not found.
Serial0: Unable to validate CHAP response. No password defined for USERNAME pioneer
Serial0: Failed CHAP authentication with remote.
Remote message is Unknown name
Serial0: remote passed CHAP authentication.
Serial0: Passed CHAP authentication with remote.
Serial0: CHAP input code = 4 id = 3 len = 48
```

In general, these messages are self-explanatory. Fields that appear in Figure 2-206 that can show optional output are outlined in Table 2-100.

**Table 2-100     Debug PPP Authentication Field Descriptions**

Field	Description
Serial0	Interface number associated with this debugging information and CHAP access session in question.
USERNAME pioneer not found.	The name <i>pioneer</i> in this example is the name received in the CHAP response. The router looks up this name in the list of usernames that are configured for the router.
Remote message is Unknown name	<p>The following messages can appear:</p> <ul style="list-style-type: none"> <li>• No name received to authenticate</li> <li>• Unknown name</li> <li>• No secret for given name</li> <li>• Short MD5 response received</li> <li>• MD compare failed</li> </ul>
code = 4	<p>Specific CHAP type packet detected. Possible values are as follows:</p> <ul style="list-style-type: none"> <li>• 1 = Challenge</li> <li>• 2 = Response</li> <li>• 3 = Success</li> <li>• 4 = Failure</li> </ul>
id = 3	ID number per Link Control Protocol (LCP) packet format.
len = 48	Packet length without header.

## debug ppp bap

To display general BACP transactions, use the **debug ppp bap** EXEC command. To disable debugging output, use the **no** form of this command.

**[no] debug ppp bap [error | event | negotiation]**

### Syntax Description

<b>error</b>	(Optional) Displays local errors.
<b>event</b>	(Optional) Displays information about protocol actions and transitions between action states (pending, waiting, idle) on the link.
<b>negotiation</b>	(Optional) Displays successive steps in negotiations between peers.

### Usage Guidelines

Do not use this command when memory is scarce or in very high traffic situations.

### Sample Displays

The following types of events generate the debug messages displayed in the figures in this section:

- A dial attempt failed.
- A BACP group was created.
- A BACP group was removed.
- The precedence of the group changed.
- Attempting to dial a number.
- Received a BACP message.
- Discarding a BACP message.
- Received an unknown code.
- Cannot find the appropriate BACP group on input.
- Displaying the response type.
- Incomplete mandatory options notification.
- Invalid outgoing message type.
- Unable to build an output message.
- Sending a BACP message.
- Details about the sent message (type of message, its identifier, the virtual access interface that sent it, and so forth).

Figure 2-207 shows the basic sample output when the **debug ppp bap** command is used.

**Figure 2-207 Sample Debug PPP BAP Output**

```
Router# debug ppp bap

BAP Virtual-Access1: group "laudrup" (2) (multilink) without precedence created

BAP laudrup: sending CallReq, id 2, len 38 on BRI3:1 to remote
BAP Virtual-Access1: received CallRsp, id 2, len 13
BAP laudrup: CallRsp, id 2, ACK
BAP laudrup: attempt1 to dial 19995776677 on BRI3
  ---> reason BAP - Multilink bundle overloaded
BAP laudrup: sending StatusInd, id 2, len 44 on Virtual-Access1 to remote
BAP Virtual-Access1: received StatusRsp, id 2, len 1
BAP laudrup: StatusRsp, id 2, ACK
```

Table 2-101 describes some basic information about the group, the events, and the sent-message details.

**Table 2-101 Debug PPP BAP Field Descriptions**

Field	Description
BAP Virtual-Access1:	Identifier of the virtual access interface in use.
group "laudrup"	Name of the BACP group.
sending CallReq	Action initiated; in this case, sending a call request.
on BRI3:1 to remote	Physical interface being used.
BAP laudrup: attempt1 to dial 19995776677 on BRI3	Call initiated, number being dialed, and physical interface being used.
---> reason BAP - Multilink bundle overloaded	Reason for initiating the BACP call.
BAP laudrup: sending StatusInd, id 2, len 44 on Virtual-Access1 to remote	Details about the sent message: it was a status indication message, had identifier 2, BACP datagram length 44, and was sent on virtual access interface 1. You can display information about the virtual access interface by using the <b>show interfaces virtual-access</b> command. (The length shown at the end of each negotiated option includes the 2-byte type and length header.)

The **debug ppp bap event** command output might show state transitions and protocol actions, in addition to the basic **debug ppp bap** output as displayed in Figure 2-207.

Figure 2-208 shows state transition output that might be displayed for the command when the **event** keyword is used.

**Figure 2-208 Debug PPP BAP Event State Transition Messages**

```
Router# debug ppp bap event

BAP laudrup: Idle --> AddWait
BAP laudrup: AddWait --> AddPending
BAP laudrup: AddPending --> Idle
```

Figure 2-209 shows protocol action output that might be displayed for the command when the **event** keyword is used.

**Figure 2-209 Debug PPP BAP Event Protocol Action Messages**

```
Router# debug ppp bap event

Peer does not support a message type
No response to a particular request
No response to all request retransmissions
Not configured to initiate link addition
Expected action by peer has not occurred
Exceeded number of retries
No links available to call out
Unable to provide phone numbers for callback
Maximum number of links in the group
Minimum number of links in the group
Unable to process link addition at present
Unable to process link removal at present
Not configured/unable to initiate link removal
Link addition completed notification
Link addition failed notification
Determination of location of the group config
Link with specified discriminator not in group
Link removal failed
Call failure with status
Failed to dial specified number
Discarding retransmission
Unable to find received identifier
Received StatusInd when no call pending
Discarding message with no phone delta
Unable to send message in particular state
Received a zero identifier
Request has precedence
```

The error messages displayed in Figure 2-210 might be added to the basic output when the **debug ppp bap error** command is used. Because the errors are very rare, you might never see these messages.

**Figure 2-210 PPP BAP Error Messages**

```
Router# debug ppp bap error

Unable to find appropriate request for received response
Invalid message type of queue
Received request is not part of the group
Add link attempt failed to locate group
Remove link attempt failed to locate group
Unable to inform peer of link addition
Changing of precedence cannot locate group
Received short header/illegal length/short packet
Invalid configuration information length
Unable to NAK incomplete options
Unable to determine current number of links
No interface list to dial on
Attempt to send invalid data
Local link discriminator is not in group
Received response type is incorrect for identifier
```

The messages displayed in Figure 2-211 might be added to the basic output when the **debug ppp bap negotiation** command is used:

**Figure 2-211 Debug PPP BAP Negotiation Messages**

```
Router# debug ppp bap negotiation

BAP laudrup: adding link speed 64 kbps for type 0x1 len 5
BAP laudrup: adding reason "User initiated addition", len 25
BAP laudrup: CallRsp, id 4, ACK
BAP laudrup: link speed 64 kbps for types 0x1, len 5 (ACK)
BAP laudrup: phone number "1: 0 2: ", len 7 (ACK)
BAP laudrup: adding call status 0, action 0 len 4
BAP laudrup: adding 1 phone numbers "1: 0 2: " len 7
BAP laudrup: adding reason "Successfully added link", len 25
BAP laudrup: StatusRsp, id 4, ACK
```

Additional negotiation messages might also be displayed for the following:

```
Received BAP message
Sending message
Decode individual options for send/receive
Notification of invalid options
```

Figure 2-212 shows additional reasons for a particular BAP action that might be displayed in an “adding reason” line of the **debug ppp bap negotiation** command output.

**Figure 2-212 Additional Reasons for a BACP Negotiation Action**

```
"Outgoing add request has precedence"
"Outgoing remove request has precedence"
"Unable to change request precedence"
"Unable to determine valid phone delta"
"Attempting to add link"
"Link addition is pending"
"Attempting to remove link"
"Link removal is pending"
"Precedence of peer marked CallReq for no action"
"Callback request rejected due to configuration"
"Call request rejected due to configuration"
"No links of specified type(s) available"
"Drop request disallowed due to configuration"
"Discriminator is invalid"
"No response to call requests"
"Successfully added link"
"Attempt to dial destination failed"
"No interfaces present to dial out"
"No dial string present to dial out"
"Mandatory options incomplete"
"Load has not exceeded threshold"
"Load is above threshold"
"Currently attempting to dial destination"
"No response to CallReq from race condition"
```

Table 2-102 describes the reasons for a BACP Negotiation Action provided in Figure 2-212.

**Table 2-102 Explanation of Reasons for BACP Negotiation Action**

<b>Reason</b>	<b>Explanation</b>
“Outgoing add request has precedence”	We received a CallRequest or CallbackRequest while we were waiting on a CallResponse or CallbackResponse to a transmitted request. We are the favored peer from the initial BACP negotiation, therefore we are issuing a NAK to our peer request.
“Outgoing remove request has precedence”	We received a LinkDropQueryRequest while we are waiting on a LinkDropQueryResponse to a transmitted request. We are the favored peer from the initial BACP negotiation, therefore we are issuing a NAK to our peer request.
“Unable to change request precedence”	We received a CallRequest, CallbackRequest or LinkDropQueryRequest while we were waiting on a LinkDropQueryResponse to a transmitted request. Our peer is deemed to be the favored peer from the initial BACP negotiation and we were unable to change the status of our outgoing request in response to the favored request so we are issuing a NAK. (This is an internal error and should never be seen.)
“Unable to determine valid phone delta”	We received a CallRequest from our peer but we are unable to provide the required phone delta for the response; therefore we are issuing a NAK. (This is an internal error and should never be seen.)
“Attempting to add link”	We received a LinkDropQueryRequest while we were attempting to add a link; a NAK is issued.
“Link addition is pending”	We received a LinkDropQueryRequest, CallRequest, or CallbackRequest while we were attempting to add a link as the result of a previous operation; a NAK is issued in the response.
“Attempting to remove link”	We received a CallRequest or CallbackRequest while we were attempting to remove a link; a NAK is issued.
“Link removal is pending”	We received a CallRequest, CallbackRequest or LinkDropQueryRequest while we were attempting to remove a link as the result of a previous operation; a NAK is issued in the response.
“Precedence of peer marked CallReq for no action”	We received an ACK to a previously unfavored CallRequest; we are issuing a CallStatusIndication to inform our peer that there will be no further action on our part as per this response.
“Callback request rejected due to configuration”	We received a CallbackRequest but we are configured not to accept them; a REJECT is issued to our peer.
“Call request rejected due to configuration”	We received a CallRequest but we are configured not to accept them; a REJECT is issued to our peer.
“No links of specified type(s) available”	We received a CallRequest but there are no links of the specified type and speed available; a NAK is issued.
“Drop request disallowed due to configuration”	We received a LinkDropQueryRequest but we are configured not to accept them; a NAK is issued to our peer.
“Discriminator is invalid”	We received a LinkDropQueryRequest but the local link discriminator is not contained within the bundle; a NAK is issued.

**Table 2-102 Explanation of Reasons for BACP Negotiation Action (Continued)**

Reason	Explanation
“No response to call requests”	After no response to our CallRequest message, a CallStatusIndication is sent to the peer informing that no more action will be taken on behalf of this operation.
“Successfully added link”	Sent as part of the CallStatusIndication informing our peer that we successfully completed the addition of a link to the bundle as the result of the transmission of a CallRequest or the reception of a CallbackRequest.
“Attempt to dial destination failed”	Sent as part of the CallStatusIndication informing our peer that we failed in an attempt to add a link to the bundle as the result of the transmission of a CallRequest or the reception of a CallbackRequest. The retry field with the CallStatusIndication informs the peer of our intentions.
“No interfaces present to dial out”	There are no available interfaces to dial out on to attempt to add a link to the bundle, and we are not going to retry the dial attempt.
“No dial string present to dial out”	We do not have a dial string to dial out with to attempt to add a link to the bundle, and we are not going to retry the dial attempt. (This is an internal error and should never be seen.)
“Mandatory options incomplete”	We received a CallRequest, CallbackRequest, LinkDropQueryRequest or CallStatusIndication and the mandatory options are not present, thus a NAK is issued in the response. (A CallStatusResponse is an ACK, however).
“Load has not exceeded threshold”	We received a CallRequest or CallbackRequest but we are issuing a NAK in the response. We are monitoring the load of the bundle, thus we determine when links should be added to the bundle.
“Load is above threshold”	We received a LinkDropQueryRequest but we are issuing a NAK in the response. We are monitoring the load of the bundle, and thus we determine when links should be removed from the bundle.
“Currently attempting to dial destination”	We received a CallbackRequest which is a retransmission of one which we previously ACK'd and are currently in the process of dialing the number suggested in the request. We are issuing an ACK because we did so previously, even though our peer never saw the previous response.
“No response to CallReq from race condition”	We issued a CallRequest but failed to receive a response, and we are issuing a CallStatusIndication to inform our peer of our intention not to proceed with the operation.

## debug ppp multilink

Use the **debug ppp multilink EXEC** command to display information about individual multilink fragments and important multilink events. The **no** form of this command disables debugging output.

**[no] debug ppp multilink**

### Usage Guidelines

The **debug ppp multilink** command has some memory overhead and should not be used when memory is scarce or in very high traffic situations.

### Sample Display

Figure 2-213 shows sample **debug ppp multilink** output when this command is used with the **ping** command. The debug output indicates that a multilink PPP packet on interface BRI 0 (on the B channel) is an input (I) or output (O) packet. The output also identifies the sequence number of the packet and the size of the fragment.

**Figure 2-213 Sample Debug PPP Multilink Output**

```
Router# debug ppp multilink
Router# ping 7.1.1.7
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 7.1.1.7, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 32/34/36 ms
Router#
2:00:28: MLP BRI0: B-Channel 1: O seq 80000000: size 58
2:00:28: MLP BRI0: B-Channel 2: O seq 40000001: size 59
2:00:28: MLP BRI0: B-Channel 2: I seq 40000001: size 59
2:00:28: MLP BRI0: B-Channel 1: I seq 80000000: size 58
2:00:28: MLP BRI0: B-Channel 1: O seq 80000002: size 58
2:00:28: MLP BRI0: B-Channel 2: O seq 40000003: size 59
2:00:28: MLP BRI0: B-Channel 2: I seq 40000003: size 59
2:00:28: MLP BRI0: B-Channel 1: I seq 80000002: size 58
2:00:28: MLP BRI0: B-Channel 1: O seq 80000004: size 58
2:00:28: MLP BRI0: B-Channel 2: O seq 40000005: size 59
2:00:28: MLP BRI0: B-Channel 2: I seq 40000005: size 59
2:00:28: MLP BRI0: B-Channel 1: I seq 80000004: size 58
2:00:28: MLP BRI0: B-Channel 1: O seq 80000006: size 58
2:00:28: MLP BRI0: B-Channel 2: O seq 40000007: size 59
2:00:28: MLP BRI0: B-Channel 2: I seq 40000007: size 59
2:00:28: MLP BRI0: B-Channel 1: I seq 80000006: size 58
2:00:28: MLP BRI0: B-Channel 1: O seq 80000008: size 58
2:00:28: MLP BRI0: B-Channel 2: O seq 40000009: size 59
2:00:28: MLP BRI0: B-Channel 2: I seq 40000009: size 59
2:00:28: MLP BRI0: B-Channel 1: I seq 80000008: size 58
```

## debug ppp multilink events

To display information about events affecting multilink groups established for BACP, use the **debug ppp multilink events** EXEC command. The **no** form of this command disables debugging output.

**[no] debug ppp multilink events**

### Usage Guidelines

Do not use this command when memory is scarce or in very high traffic situations.

### Sample Display

Figure 2-214 shows sample **debug ppp multilink events** command output.

**Figure 2-214 Sample Debug PPP Multilink Events Output**

```
Router# debug ppp multilink events

MLP laudrup: established BAP group 4 on Virtual-Access1, physical BRI3:1
MLP laudrup: removed BAP group 4
```

Other event messages include the following:

```
Unable to find bundle for BAP group identifier
Unable to find physical interface to start BAP
Unable to create BAP group
Attempt to start BACP when inactive or running
Attempt to start BACP on non-MLP interface
Link protocol has gone down, removing BAP group
Link protocol has gone down, BAP not running or present
```

Table 2-103 describes the significant fields in Figure 2-214.

**Table 2-103 Debug PPP Multilink Events Field Descriptions**

Field	Description
MLP laudrup	Name of the multilink group.
established BAP group 4	Internal identifier. The same identifiers are used in the <b>show ppp bap group</b> command output.
Virtual-Access1	Dynamic access interface number.
physical BRI3:1	Bundle was established from a call on this interface.
removed BAP group 4	When the bundle is removed, the associated BACP group (with its ID) is also removed.

## debug qlc error

Use the **debug qlc error** EXEC command to display quality link line control (QLLC) errors. The **no** form of this command disables debugging output.

**[no] debug qlc error**

### Usage Guidelines

This command helps you track down errors in the QLLC interactions with X.25 networks. Use **debug qlc error** in conjunction with **debug x25 all** to see the connection. The data shown by this command only flows through the router on the X.25 connection. Some forms of this command can generate lots of output and network traffic.

### Sample Display

Figure 2-215 shows sample **debug qlc error** output.

#### Figure 2-215 Sample Debug QLLC Error Output

```
Router# debug qlc error

%QLLC-3-GENERRMSG: qlc_close - bad qlc pointer Caller 00407116 Caller 00400BD2
QLLC 4000.1111.0002: NO X.25 connection. Dicarding XID and calling out
```

Explanations for individual lines of output from Figure 2-215 follow.

The following line indicates that the QLLC connection was closed:

```
%QLLC-3-GENERRMSG: qlc_close - bad qlc pointer Caller 00407116 Caller 00400BD2
```

The following line shows the virtual MAC address of the failed connection:

```
QLLC 4000.1111.0002: NO X.25 connection. Dicarding XID and calling out
```

## debug qlc event

Use the **debug qlc event** EXEC command to enable debugging of QLLC events. The **no** form of this command disables debugging output.

**[no] debug qlc event**

### Usage Guidelines

Use the **debug qlc event** command to display primitives that might affect the state of a QLLC connection. An example of these events is the allocation of a QLLC structure for a logical channel indicator when an X.25 call has been accepted with the QLLC call user data. Other examples are the receipt and transmission of LAN explorer and XID frames.

### Sample Display

Figure 2-216 shows sample **debug qlc event** output.

**Figure 2-216 Sample Debug QLLC Event Output**

```
Router# debug qlc event

QLLC: allocating new qlc lci 9
QLLC: tx POLLING TEST, da 4001.3745.1088, sa 4000.1111.0001
QLLC: rx explorer response, da 4000.1111.0001, sa c001.3745.1088, rif 08B0.1A91.1901.A040
QLLC: gen NULL XID, da c001.3745.1088, sa 4000.1111.0001, rif 0830.1A91.1901.A040, dsap
4, ssap 4
QLLC: rx XID response, da 4000.1111.0001, sa c001.3745.1088, rif 08B0.1A91.1901.A040
```

Explanations for representative lines of output in Figure 2-216 follow.

The following line indicates a new QLLC data structure has been allocated:

```
QLLC: allocating new qlc lci 9
```

The following lines show transmission and receipt of LAN explorer or test frames:

```
QLLC: tx POLLING TEST, da 4001.3745.1088, sa 4000.1111.0001
QLLC: rx explorer response, da 4000.1111.0001, sa c001.3745.1088, rif 08B0.1A91.1901.A040
```

The following lines show XID events:

```
QLLC: gen NULL XID, da c001.3745.1088, sa 4000.1111.0001, rif 0830.1A91.1901.A040, dsap
4, ssap 4
QLLC: rx XID response, da 4000.1111.0001, sa c001.3745.1088, rif 08B0.1A91.1901.A040
```

## debug qlc packet

Use the **debug qlc packet EXEC** command to display QLLC events and QLLC data packets. The **no** form of this command disables debugging output.

```
[no] debug qlc packet
```

### Usage Guidelines

This command helps you to track down errors in the QLLC interactions with X.25 networks. The data shown by this command only flows through the router on the X25 connection. Use **debug qlc packet** in conjunction with **debug x25 all** to see the connection and the data that flows through the router.

### Sample Display

Figure 2-217 shows sample **debug qlc packet** output.

**Figure 2-217 Sample Debug QLLC Packet Output**

```
Router# debug qlc packet

14:38:05: Serial2/5 QLLC I: Data Packet.-RSP 9 bytes.
14:38:07: Serial2/6 QLLC I: Data Packet.-RSP 112 bytes.
14:38:07: Serial2/6 QLLC O: Data Packet. 128 bytes.
14:38:08: Serial2/6 QLLC I: Data Packet.-RSP 9 bytes.
14:38:08: Serial2/6 QLLC I: Data Packet.-RSP 112 bytes.
14:38:08: Serial2/6 QLLC O: Data Packet. 128 bytes.
14:38:08: Serial2/6 QLLC I: Data Packet.-RSP 9 bytes.
14:38:12: Serial2/5 QLLC I: Data Packet.-RSP 112 bytes.
14:38:12: Serial2/5 QLLC O: Data Packet. 128 bytes.
```

Explanations for individual lines of output from Figure 2-217 follow.

The following lines indicate a packet was received on the interfaces:

```
14:38:05: Serial2/5 QLLC I: Data Packet.-RSP 9 bytes.
14:38:07: Serial2/6 QLLC I: Data Packet.-RSP 112 bytes.
```

The following lines show that a packet was transmitted on the interfaces:

```
14:38:07: Serial2/6 QLLC O: Data Packet. 128 bytes.
14:38:12: Serial2/5 QLLC O: Data Packet. 128 bytes.
```

## debug qlc state

Use the **debug qlc state** EXEC command to enable debugging of the QLLC events. The **no** form of this command disables debugging output.

**[no] debug qlc state**

### Usage Guidelines

Use the **debug qlc state** command to show when the state of a QLLC connection has changed. The typical QLLC connection goes from states ADM to SETUP to NORMAL. The NORMAL state indicates that a QLLC connection exists and is ready for data transfer.

### Sample Display

Figure 2-218 shows sample **debug qlc state** output.

**Figure 2-218 Sample Debug QLLC State Output**

```
Router# debug qlc state

Serial2 QLLC O: QSM-CMD
Serial2: X25 O D1 DATA (5) Q 8 lci 9 PS 4 PR 3
QLLC: state ADM -> SETUP
Serial2: X25 I D1 RR (3) 8 lci 9 PR 5
Serial2: X25 I D1 DATA (5) Q 8 lci 9 PS 3 PR 5
Serial2 QLLC I: QUA-RSPQLLC: addr 00, ctl 73

QLLC: qsetupstate: recvd qua rsp
QLLC: state SETUP -> NORMAL
```

Explanations for representative lines of output in Figure 2-218 follow.

The following line indicates a QLLC connection attempt is changing state from ADM to SETUP:

```
QLLC: state ADM -> SETUP
```

The following line indicates a QLLC connection attempt is changing state from SETUP to NORMAL:

```
QLLC: state SETUP -> NORMAL
```

## debug qlc timer

Use the **debug qlc timer** EXEC command to display QLLC timer events. The **no** form of this command disables debugging output.

**[no] debug qlc timer**

### Usage Guidelines

The QLLC process periodically cycles and checks status of itself and its partner. If the partner is not found in the desired state, a LAPB primitive command is resent until the partner is in the desired state or the timer expires.

### Sample Display

Figure 2-219 shows sample **debug qlc timer** output.

**Figure 2-219 Sample Debug QLLC Timer Output**

```
Router# debug qlc timer

14:27:24: Qllc timer lci 257, state ADM retry count 0 Caller 00407116 Caller 00400BD2
14:27:34: Qllc timer lci 257, state NORMAL retry count 0
14:27:44: Qllc timer lci 257, state NORMAL retry count 1
14:27:54: Qllc timer lci 257, state NORMAL retry count 1
```

Explanations for individual lines of output from Figure 2-219 follow.

The following line of output shows the state of a QLLC partner on a given X.25 logical channel identifier:

```
14:27:24: Qllc timer lci 257, state ADM retry count 0 Caller 00407116 Caller 00400BD2
```

Other messages are informational and appear every ten seconds.

## debug qlc x25

Use the debug qlc x25 EXEC command to display X.25 packets that affect a QLLC connection. The **no** form of this command disables debugging output.

**[no] debug qlc x25**

### Usage Guidelines

This command is helpful to track down errors in the QLLC interactions with X.25 networks. Use **debug qlc x25** in conjunction with **debug x25 events** or **debug x25 all** to see the X.25 events between the router and its partner.

### Sample Display

Figure 2-220 shows sample **debug qlc x25** output.

**Figure 2-220 Sample Debug QLLC X25 Output**

```
Router# debug qlc x25

15:07:23: QLLC X25 notify lci 257 event 1
15:07:23: QLLC X25 notify lci 257 event 5
15:07:34: QLLC X25 notify lci 257 event 3 Caller 00407116 Caller 00400BD2
15:07:35: QLLC X25 notify lci 257 event 4
```

Table 2-104 describes fields of output that appear in Figure 2-220.

**Table 2-104 Debug QLLC X.25 Field Descriptions**

Field	Description
15:07:23	Shows the time of day.
QLLC X25 notify 257	Indicates this is a QLLC X25 message.
event <i>n</i>	Indicates the type of event, <i>n</i> . Values for <i>n</i> can be as follows: <ul style="list-style-type: none"> <li>• 1 – Circuit is cleared</li> <li>• 2 – Circuit has been reset</li> <li>• 3 – Circuit is connected</li> <li>• 4 – Circuit congestion has cleared</li> <li>• 5 – Circuit has been deleted</li> </ul>

## debug radius

Use the **debug radius** EXEC command to display information associated with the Remote Authentication Dial-In User Server (RADIUS). The **no** form of this command disables debugging output.

**[no] debug radius**

### Usage Guidelines

RADIUS is a distributed security system that secures networks against unauthorized access. Cisco supports RADIUS under the Authentication, Authorization, and Accounting (AAA) security system.

Use the **debug aaa authentication** command to get a high-level view of login activity. When RADIUS is used on the router, you can use the **debug radius** command for more detailed debugging information.

### Sample Displays

Figure 2-221 shows part of the **debug aaa authentication** command output for a RADIUS login attempt that failed. The information indicates that RADIUS is the authentication method used.

**Figure 2-221 Sample Debug AAA Authentication Output—RADIUS Login Attempt**

```
Router# debug aaa authentication
14:02:55: AAA/AUTHEN (164826761): Method=RADIUS
14:02:55: AAA/AUTHEN (164826761): status = GETPASS
14:03:01: AAA/AUTHEN/CONT (164826761): continue_login
14:03:01: AAA/AUTHEN (164826761): status = GETPASS
14:03:01: AAA/AUTHEN (164826761): Method=RADIUS
14:03:04: AAA/AUTHEN (164826761): status = FAIL
```

Figure 2-222 shows part of the **debug radius** command output that shows a login attempt that failed because of a key mismatch (that is, a configuration problem).

**Figure 2-222 Sample Debug RADIUS Output—Failed Login (Key Mismatch)**

```
Router# debug radius
13:55:19: Radius: IPC Send 0.0.0.0:1645, Access-Request, id 0x7, len 57
13:55:19:      Attribute 4 6 AC150E5A
13:55:19:      Attribute 5 6 0000000A
13:55:19:      Attribute 1 7 62696C6C
13:55:19:      Attribute 2 18 19D66483
13:55:22: Radius: Received from 171.69.1.152:1645, Access-Reject, id 0x7, len 20
13:55:22: Radius: Reply for 7 fails decrypt
```

Figure 2-223 shows part of the **debug radius** command output that shows a successful login attempt as indicated by an Access-Accept message.

**Figure 2-223 Sample Debug RADIUS Output—Successful Login**

```
Router# debug radius

13:59:02: Radius: IPC Send 0.0.0.0:1645, Access-Request, id 0xB, len 56
13:59:02:      Attribute 4 6 AC150E5A
13:59:02:      Attribute 5 6 0000000A
13:59:02:      Attribute 1 6 62696C6C
13:59:02:      Attribute 2 18 0531FEA3
13:59:04: Radius: Received from 171.69.1.152:1645, Access-Accept, id 0xB, len 26
13:59:04:      Attribute 6 6 00000001
```

Figure 2-224 shows part of the **debug radius** command output that shows an unsuccessful login attempt as indicated by the Access-Reject message.

**Figure 2-224 Sample Debug RADIUS Output—Failed Login**

```
Router# debug radius

13:57:56: Radius: IPC Send 0.0.0.0:1645, Access-Request, id 0xA, len 57
13:57:56:      Attribute 4 6 AC150E5A
13:57:56:      Attribute 5 6 0000000A
13:57:56:      Attribute 1 7 62696C6C
13:57:56:      Attribute 2 18 49C28F6C
13:57:59: Radius: Received from 171.69.1.152:1645, Access-Reject, id 0xA, len 20
```

### Related Commands

**debug aaa accounting**  
**debug aaa authentication**

## debug rif

Use the **debug rif** EXEC command to display information on entries entering and leaving the routing information field (RIF) cache. The **no** form of this command disables debugging output.

**[no] debug rif**

### Usage Guidelines

In order to use the **debug rif** command to display traffic source-routed through an interface, fast switching of source route bridging (SRB) frames must first be disabled with the **no source-bridge route-cache** interface configuration command.

### Sample Display

Figure 2-225 shows sample **debug rif** output.

**Figure 2-225 Sample Debug RIF Output**

```

router# debug rif
SDLLC or Local-Ack entry — RIF: U chk da=9000.5a59.04f9,sa=0110.2222.33c1 [4880.3201.00A1.0050] type 8 on
                             static/remote/0
                             RIF: U chk da=0000.3080.4aed,sa=0000.0000.0000 [] type 8 on TokenRing0/0
Non-SDLLC or non-Local-Ack entry — RIF: U add 1000.5a59.04f9 [4880.3201.00A1.0050] type 8
                                     RIF: L checking da=0000.3080.4aed, sa=0000.0000.0000
                                     RIF: rcvd TEST response from 9000.5a59.04f9
                                     RIF: U upd da=1000.5a59.04f9,sa=0110.2222.33c1 [4880.3201.00A1.0050]
                                     RIF: rcvd XID response from 9000.5a59.04f9
                                     SR1: sent XID response to 9000.5a59.04f9

```

Explanations for representative lines of **debug rif** output in Figure 2-225 follow.

The first line of output is an example of a RIF entry for an interface configured for SDLLC or Local-Ack. Table 2-105 describes significant fields shown in this line of **debug rif** output.

**Table 2-105 Debug RIF Field Descriptions—Part 1**

Field	Description
RIF:	This message describes RIF debugging output.
U chk	Update checking. The entry is being updated; the timer is set to zero (0).
da = 9000.5a59.04f9	Destination MAC address.
sa = 0110.2222.33c1	Source MAC address. This field contains values of zero (0000.0000.0000) in a non-SDLLC or non-Local-ack entry.
[4880.3201.00A1.0050]	RIF string. This field is blank (null RIF) in a non-SDLLC or non-Local-Ack entry.

**Table 2-105 Debug RIF Field Descriptions—Part 1 (Continued)**

Field	Description
type 8	Possible values follow: <ul style="list-style-type: none"> <li>• 0—Null entry</li> <li>• 1—This entry was learned from a particular Token Ring port (interface)</li> <li>• 2—Statically configured</li> <li>• 4—Statically configured for a remote interface</li> <li>• 8—This entry is to be aged</li> <li>• 16—This entry (which has been learned from a remote interface) is to be aged</li> <li>• 32—This entry is not to be aged</li> <li>• 64 —This interface is to be used by LAN Network Manager (and is not to be aged)</li> </ul>
on static/remote/0	This route was learned from a real Token Ring port, in contrast to a virtual ring.

The following line of output is an example of a RIF entry for an interface that is not configured for SDLLC or Local-Ack:

```
RIF: U chk da=0000.3080.4aed,sa=0000.0000.0000 [] type 8 on TokenRing0/0
```

Notice that the source address contains only zero values (0000.0000.0000), and that the RIF string is null ([ ]). The last element in the entry indicates that this route was learned from a virtual ring, rather than a real Token Ring port.

The following line shows that a new entry has been added to the RIF cache:

```
RIF: U add 1000.5a59.04f9 [4880.3201.00A1.0050] type 8
```

The following line shows that a RIF cache lookup operation has taken place:

```
RIF: L checking da=0000.3080.4aed, sa=0000.0000.0000
```

The following line shows that a TEST response from address 9000.5a59.04f9 was inserted into the RIF cache:

```
RIF: rcvd TEST response from 9000.5a59.04f9
```

The following line shows that the RIF entry for this route has been found and updated:

```
RIF: U upd da=1000.5a59.04f9,sa=0110.2222.33c1 [4880.3201.00A1.0050]
```

The following line shows that an XID response from this address was inserted into the RIF cache:

```
RIF: rcvd XID response from 9000.5a59.04f9
```

The following line shows that the router sent an XID response to this address:

```
SR1: sent XID response to 9000.5a59.04f9
```

Table 2-106 explains the other possible lines of **debug rif** output.

**Table 2-106 Debug RIF Field Descriptions—Part 2**

<b>Field</b>	<b>Description</b>
RIF: L Sending XID for <i>address</i>	The router/bridge wanted to send a packet to <i>address</i> but did not find it in the RIF cache. It sent an XID explorer packet to determine which RIF it should use. The attempted packet is dropped.
RIF: L No buffer for XID to <i>address</i>	Similar to the previous description; however, a buffer in which to build the XID packet could not be obtained.
RIF: U remote rif too small [ <i>rif</i> ]	A packet's RIF was too short to be valid.
RIF: U rej <i>address</i> too big [ <i>rif</i> ]	A packet's RIF exceeded the maximum size allowed and was rejected. The maximum size is 18 bytes.
RIF: U upd interface <i>address</i>	The RIF entry for this router/bridge's interface has been updated.
RIF: U ign <i>address</i> interface update	A RIF entry that would have updated an interface corresponding to one of this router's interfaces.
RIF: U add <i>address</i> [ <i>rif</i> ]	The RIF entry for <i>address</i> has been added to the RIF cache.
RIF: U no memory to add rif for <i>address</i>	No memory to add a RIF entry for <i>address</i> .
RIF: removing rif entry for <i>address</i> , <i>type code</i>	The RIF entry for <i>address</i> has been forcibly removed.
RIF: flushed <i>address</i>	The RIF entry for <i>address</i> has been removed because of a RIF cache flush.
RIF: expired <i>address</i>	The RIF entry for <i>address</i> has been aged out of the RIF cache.

**Related Command****debug list**

## debug rtr error

Use the **debug rtr error** EXEC command to enable logging of response time reporter runtime errors. The **no** form of this command disables debugging output—including the **debug rtr trace** command.

**[no] debug rtr error** [*probe*]

### Syntax Description:

*probe* (Optional) Number of the probe in the range 0 to 31.

### Usage Guidelines

The response time reporter feature allows you to monitor network performance and network resources by measuring response times and availability. With this feature you can perform troubleshooting, problem notifications, and preproblem analysis based on response time reporter statistics.

The **debug rtr error** command displays runtime errors. When a probe number other than 0 is specified, all runtime errors for that probe are displayed when the probe is active. When the probe number is 0, all runtime errors relating to the response time reporter scheduler process are displayed. When no probe number is specified, all runtime errors for all active probes configured on the router and probe control are displayed. The response time reporter scheduler process is responsible for starting and stopping probes and managing the database.

---

**Note** Use the **debug rtr error** command before using the **debug rtr trace** command because the **debug rtr error** command generates a smaller amount of traffic.

---

Because the trace output generated by the **debug rtr trace** command uses the same control mechanism as **debug rtr error**, the **no debug rtr error** command disables both logging and tracing.

### Sample Display

Figure 2-226 shows sample **debug rtr error** output. All debug output for the response time reporter (including **debug rtr trace**) has the format shown in Figure 2-226.

**Figure 2-226 Sample Debug RTR Error Output**

```
Router# debug rtr error 1

RTR 1: Error Return Code - LU0 RTR Probe 1
      in echoTarget on call luReceive
      LuApiReturnCode of InvalidHandle - invalid host name or API handle
```

Table 2-107 shows describes the fields and messages shown in Figure 2-226.

**Table 2-107**    **Debug RTR Error Field Descriptions**

<b>Field</b>	<b>Description</b>
RTR 1	Number of the probe generating the message.
Error Return Code	Message identifier indicating the error type (or error itself).
LU0 RTR Probe 1	Name of the process generating the message.
in echoTarget on call luReceive LuApiReturnCode of InvalidHandle - invalid host name or API handle	Supplemental messages that pertain to the message identifier.

**Related Command****debug rtr trace**

## debug rtr trace

Use the **debug rtr trace** EXEC command to trace the execution of a response time reporter probe. The **no** form of this command disables trace debugging output (but not **debug rtr error** output).

[no] **debug rtr trace** [*probe*]

### Syntax Description:

*probe* (Optional) Number of the probe in the range 0 to 31.

### Usage Guidelines

---

**Note** The **debug rtr trace** command can generate a large number of debug messages. First use the **debug rtr error** command, and then use the **debug rtr trace** on a per probe basis.

---

When a probe number other than 0 is specified, execution for that probe is traced. When the probe number is 0, the response time reporter scheduler process is traced. When no probe number is specified, all active probes and every probe control is traced. The response time reporter scheduler process is responsible for starting and stopping probes and managing the database.

The **debug rtr trace** command also enables **debug rtr error** for the specified probe. However, the **no debug rtr trace** command does not disable the **debug rtr error** command. You must manually disable the command by using the **no debug rtr error** command.

All debug output (including **debug rtr error**) has the format shown in the **debug rtr error** output example in Figure 2-226.

### Sample Display

Figure 2-227 shows a partial sample of **debug rtr trace** output. In this example, a probe is traced through a single operation attempt: the setup of a connection to the target (LU0), the attempt at an echo, and the closing of the connection on the echo attempt.

**Figure 2-227 Sample Debug RTR Trace Output**

```
Router# debug rtr trace

RTR 1: Calling getRttMonOperState (check pending) - LU0 RTR Probe 1
RTR 1: Calling getRttMonOperState (check death) - LU0 RTR Probe 1
RTR 1: Starting An Echo Operation - LU0 RTR Probe 1
RTR 1: setting receiveFinished to FALSE - LU0 RTR Probe 1
        in dependLuEchoApplication
RTR 1: openConnection called - LU0 RTR Probe 1
        Calling rttMonHopConnected
RTR 1: calling luT0orT2Open - LU0 RTR Probe 1
RTR 1: Dumping luT0orT2Open Result - LU0 RTR Probe 1
        applicationHandle: inRttMonCtrlAdminQItem = 13920808
        receiveFinished = FALSE currentLUHandle = 14199576
        maxRespBufferLen = 52 receivedBufferLen = 0
        aHostName = CWBC02 locAddr = 1 eApplNameLen = 7
        ApplName = D5E2D7C5C3C8D60 eModeName = 4040404040404040 userDataLen = 14
```

```

        userData = D5E2D7C5C3C8D61 sysSense = 0 userSense = 0 bindDataLen = 64
        bindData = D5E2D7C5C3C8D61 bindDataCount = 14
RTR 1: Calling rttMonSetConnectionHandle - LU0 RTR Probe 1
RTR 1: Calling rttMonSetHopConnectedState to TRUE - LU0 RTR Probe 1
RTR 1: Calling rttMonSetDiagText - LU0 RTR Probe 1
RTR 1: setupPathInfo called - LU0 RTR Probe 1
        Calling rttMonStartUpdatePath
RTR 1: Calling rttMonUpdatePath - LU0 RTR Probe 1
        for Target CWBC02 NSPECHO
RTR 1: Calling rttMonEndUpdatePath - LU0 RTR Probe 1
RTR 1: Calling rttMonUpdateNumberOfEchosAttempted - LU0 RTR Probe 1
RTR 1: performEcho called - LU0 RTR Probe 1
        applicationHandle: inRttMonCtrlAdminQItem = 13920808
        receiveFinished = FALSE currentLUHandle = 14199576
        maxRespBufferLen = 52 receivedBufferLen = 0
        echoTimeout (milliseconds) = 5000 sequenceNum = 886
        rttMonEchoAdminTargetAddress is CWBC02 NSPECHO
        verifyDataFlag = False
RTR 1: Calling rttMonGetFirstStoredHopAddress - LU0 RTR Probe 1
RTR 1: echoTarget called - LU0 RTR Probe 1
        applicationHandle: inRttMonCtrlAdminQItem = 13920808
        receiveFinished = FALSE currentLUHandle = 14199576
        maxRespBufferLen = 52 receivedBufferLen = 0
        echoTimeout (milliseconds) = 5000
        Data:
            E6 A7 E8 A9 01 02 03 77 00 00 00 00 C1
        calling luReceive...
RTR 1: calling luSend - LU0 RTR Probe 1
RTR 1: receiveFinished is FALSE - LU0 RTR Probe 1
        in echoTarget calling
        process_wait_for_event_timed(5000 milliseconds)
RTR 1: rtt_closeIndication called - DSPU Msg Proc
        applicationHandle: inRttMonCtrlAdminQItem = 13920808
        receiveFinished = FALSE currentLUHandle = 14199576
        maxRespBufferLen = 52 receivedBufferLen = 0
RTR 1: setting receiveFinished to TRUE - DSPU Msg Proc
        in rtt_closeIndication
RTR 1: Woke on receive event - LU0 RTR Probe 1
RTR 1: returned from echoTarget - LU0 RTR Probe 1
        with D_echoTarget_connectionLost return_code
        and responseTime (milliseconds) = 196
...
RTR 1: Calling getRttMonOperState (check active) - LU0 RTR Probe 1
RTR 1: Going to Sleep - LU0 RTR Probe 1
        until next frequency time (delta milliseconds 60000)

```

## Related Command

**debug rtr error**

## debug sdlc

Use the **debug sdlc** EXEC command to display information on Synchronous Data Link Control (SDLC) frames received and sent by any router serial interface involved in supporting SDLC end station functions. The **no** form of this command disables debugging output.

**[no] debug sdlc**

### Usage Guidelines

---

**Note** Because the **debug sdlc** command can generate many messages and alter timing in the network node, use it only when instructed by authorized support personnel.

---

### Sample Display

Figure 2-228 shows sample **debug sdlc** output.

**Figure 2-228 Sample Debug SDLC Output**

```
Router# debug sdlc

SDLC: Sending RR at location 4
Serial3: SDLC O (12495952) C2 CONNECT (2) RR P/F 6
Serial3: SDLC I (12495964) [C2] CONNECT (2) RR P/F 0 (R) [VR: 6 VS: 0]
Serial3: SDLC T [C2] 12496064 CONNECT 12496064 0
SDLC: Sending RR at location 4
Serial3: SDLC O (12496064) C2 CONNECT (2) RR P/F 6
Serial3: SDLC I (12496076) [C2] CONNECT (2) RR P/F 0 (R) [VR: 6 VS: 0]
Serial3: SDLC T [C2] 12496176 CONNECT 12496176 0
```

Explanations for individual lines of output from Figure 2-228 follow.

The following line of output indicates that the router is sending a Receiver Ready packet at location 4 in the code:

```
SDLC: Sending RR at location 4
```

The following line of output describes a frame input event:

```
Serial3: SDLC O (12495952) C2 CONNECT (2) RR P/F 6
```

Table 2-108 describes the fields in this line of output.

**Table 2-108 Debug SDLC Field Descriptions for a Frame Output Event**

Field	Description
SDLC	Protocol providing the information.
Serial3	Interface type and unit number reporting the frame event.
O	Command mode of frame event. Possible values follow: <ul style="list-style-type: none"> <li>• I—Frame input</li> <li>• O—Frame output</li> <li>• T—T1 timer expired</li> </ul>

**Table 2-108 Debug SDLC Field Descriptions for a Frame Output Event (Continued)**

Field	Description
(12495952)	Current timer value.
C2	SDLC address of the SDLC connection.
CONNECT	State of the protocol when the frame event occurred. Possible values follow: <ul style="list-style-type: none"> <li>• CONNECT</li> <li>• DISCONNECT</li> <li>• DISCSENT (disconnect sent)</li> <li>• ERROR (FRMR frame sent)</li> <li>• REJSENT (reject frame sent)</li> <li>• SNRMSSENT (SNRM frame sent)</li> <li>• USBUSY</li> <li>• THEMBUSY</li> <li>• BOTHBUSY</li> </ul>
(2)	Size of the frame (in bytes).
RR	Frame type name. Possible values follow: <ul style="list-style-type: none"> <li>• DISC—Disconnect</li> <li>• DM—Disconnect mode</li> <li>• FRMR—Frame reject</li> <li>• IFRAME—Information frame</li> <li>• REJ—Reject</li> <li>• RNR—Receiver not ready</li> <li>• RR—Receiver ready</li> <li>• SIM—Set Initialization mode command</li> <li>• SNRM—Set Normal Response Mode</li> <li>• TEST—Test frame</li> <li>• UA—Unnumbered acknowledgment</li> <li>• XID—EXchange ID</li> </ul>
P/F	Poll/Final bit indicator. Possible values follow: <ul style="list-style-type: none"> <li>• F—Final (printed for Response frames)</li> <li>• P—Poll (printed for Command frames)</li> <li>• P/F—Poll/Final (printed for RR, RNR and REJ frames, which can be either Command or Response frames)</li> </ul>
6	Receive count; range: 0–7.

The following line of output describes a frame input event:

```
Serial3: SDLC I (12495964) [C2] CONNECT (2) RR P/F 0 (R) [VR: 6 VS: 0] rfp: P
```

In addition to the fields described in Table 2-108, output for a frame input event also includes the additional fields described in Table 2-109.

**Table 2-109 Debug SDLC Field Descriptions Unique to a Frame Input Event**

Field	Description
(R)	Frame Type: <ul style="list-style-type: none"> <li>• C—Command</li> <li>• R—Response</li> </ul>
VR: 6	Receive count; range: 0–7.
VS: 0	Send count; range: 0–7.
rfp: P	Ready for poll; <ul style="list-style-type: none"> <li>• P—Idle poll (keepalive) timer is on.</li> <li>• T—Data acknowledgment timer is on.</li> </ul> These timers are based on the TI timer.
VS: 0	Send count; range: 0–7.

The following line of output describes a frame timer event:

```
Serial3: SDLC T [C2] 12496064 CONNECT 12496064 0
```

Table 2-110 describes the fields in this line of output.

**Table 2-110 Debug SDLC Field Descriptions for a Timer Event**

Field	Description
Serial3:	Interface type and unit number reporting the frame event.
SDLC	Protocol providing the information.
T	The timer has expired.
[C2]	SDLC address of this SDLC connection.
12496064	System clock.
CONNECT	State of the protocol when the frame event occurred. Possible values follow: <ul style="list-style-type: none"> <li>• BOTHBUSY</li> <li>• CONNECT</li> <li>• DISCONNECT</li> <li>• DISCSENT (disconnect sent)</li> <li>• ERROR (FRMR frame sent)</li> <li>• REJSENT (reject frame sent)</li> <li>• SNRMSSENT (SNRM frame sent)</li> <li>• THEMBUSY</li> <li>• BOTHBUSY</li> </ul>
12496064	Top timer.
0	Retry count; default: 0.

**Related Command**  
**debug list**

## debug sdlc local-ack

Use the **debug sdlc local-ack** EXEC command to display information on the local acknowledgment feature. The **no** form of this command disables debugging output.

**[no] debug sdlc local-ack** *[number]*

### Syntax Description

*number* (Optional) Frame type that you want to monitor. Refer to the Usage Guidelines section.

### Usage Guidelines

You can select the frame types you want to monitor; the frame types correspond to bit flags. You can select 1, 2, 4, or 7, which is the decimal value of the bit flag settings. If you select 1, the octet is set to 00000001. If you select 2, the octet is set to 0000010. If you select 4, the octet is set to 00000100. If you want to select all frame types, select 7; the octet is 00000111. The default is 7 for all events. Table 2-111 defines these bit flags.

**Table 2-111 Debug SDLC Local-Ack Debugging Levels**

Debug Command	Meaning
<b>debug sdlc local-ack 1</b>	Only U-Frame events
<b>debug sdlc local-ack 2</b>	Only I-Frame events
<b>debug sdlc local-ack 4</b>	Only S-Frame events
<b>debug sdlc local-ack 7</b>	All SDLC Local-Ack events (default setting)



**Caution** Because using this command is processor intensive, it is best to use it after hours, rather than in a production environment. It is also best to use this command by itself, rather than in conjunction with other **debugging** commands.

### Sample Display

Figure 2-229 shows sample **debug sdlc local-ack** output.

**Figure 2-229 Sample Debug SDLC Local-Ack Output**

```

router# debug sdlc local-ack 1
SLACK (Serial3): Input      = Network, LinkupRequest
SLACK (Serial3): Old State = AwaitSdlcOpen           New State = AwaitSdlcOpen

SLACK (Serial3): Output    = SDLC, SNRM

SLACK (Serial3): Input      = SDLC, UA
SLACK (Serial3): Old State = AwaitSdlcOpen           New State = Active

SLACK (Serial3): Output    = Network, LinkResponse

```

Group of associated operations

S2560

Explanations for individual lines of output from Figure 2-229 follow.

The first line shows the input to the SDLC local acknowledgment state machine:

```
SLACK (Serial3): Input      = Network, LinkupRequest
```

Table 2-112 describes the fields in this line of output.

**Table 2-112 Debug SDLC Local-Ack Field Descriptions**

Field	Description
SLACK	The SDLC local acknowledgment feature is providing the information.
(Serial3):	Interface type and unit number reporting the event.
Input = Network	The source of the input.
LinkupRequest	The op code. A LinkupRequest is an example of possible values.

The second line shows the change in the SDLC local acknowledgment state machine. In this case the AwaitSdlcOpen state is an internal state that has not changed while this display was captured.

```
SLACK (Serial3): Old State = AwaitSdlcOpen          New State = AwaitSdlcOpen
```

The third line shows the output from the SDLC local acknowledgment state machine:

```
SLACK (Serial3): Output      = SDLC, SNRM
```

## debug sdlc packet

Use the **debug sdlc packet** EXEC command to display packet information on Synchronous Data Link Control (SDLC) frames received and sent by any router serial interface involved in supporting SDLC end station functions. The **no** form of this command disables debugging output.

**[no] debug sdlc packet** [*max-bytes*]

### Syntax Description

*max-byte* (Optional) Limits the number of bytes of data that are printed to the display.

### Usage Guidelines

This command requires intensive CPU processing; therefore, we recommend not using it when the router is expected to handle normal network loads, such as in a production environment. Instead, use this command when network response is non-critical. We also recommend that you use this command by itself, rather than in conjunction with other **debug** commands.

### Sample Display

Figure 2-230 shows sample **debug sdlc packet** output with the packet display limited to 20 bytes of data.

**Figure 2-230 Sample Debug SDLC Packet Output**

```
Router# debug sdlc packet 20

Serial3 SDLC Output
00000 C3842C00 02010010 019000C5 C5C5C5C5 Cd.....EEEEEE
00010 C5C5C5C5                               EEEE
Serial3 SDLC Output
00000 C3962C00 02010011 039020F2          Co.....2
Serial3 SDLC Output
00000 C4962C00 0201000C 039020F2          Do.....2
Serial3 SDLC Input
00000      C491                               Dj
```

## debug sdllc

Use the **debug sdllc EXEC** command to display information about data link layer frames transferred between a device on a Token Ring and a device on a serial line via a router configured with the SDLLC feature. The **no** form of this command disables debugging output.

**[no] debug sdllc**

### Usage Guidelines

The SDLLC feature translates between the SDLC link layer protocol used to communicate with devices on a serial line and the LLC2 link layer protocol used to communicate with devices on a Token Ring.

The router configured with the SDLLC feature must be attached to the serial line. The router sends and receives frames on behalf of the serial device on the attached serial line but acts as an SDLC station.

The topology between the router configured with the SDLLC feature and the Token Ring is network dependent and is not limited by the SDLLC feature.

### Sample Display

Figure 2-231 shows sample **debug sdllc** output between link layer peers from the perspective of the SDLLC-configured router.

**Figure 2-231 Sample Debug SDLLC Output**

```
Router# debug sdllc

SDLLC: rx explorer rsp, da 4000.2000.1001, sa C000.1020.1000, rif
      8840.0011.00A1.0050
SDLLC: tx short xid, sa 4000.2000.1001, da C000.1020.1000, rif
      88C0.0011.00A1.0050, dsap 4 ssap 4
SDLLC: tx long xid, sa 4000.2000.1001, da C000.1020.1000, rif
      88C0.0011.00A1.0050, dsap 4 ssap 4
Rcvd SABME/LINKUP_REQ pak from TR host
SDLLCERR: not from our partner, pak dropped, da 4000.2000.1001,
sa C000.1020.1000, rif 8840.0011.00A1.0050, partner = 5000.1040.1003
```

Table 2-113 describes significant fields shown in Figure 2-231.

**Table 2-113 Debug SDLLC Field Descriptions**

Field	Description
rx	Router receives message from the FEP.
explorer rsp	Response to an explorer (TEST) frame previously sent by the router to FEP.
da	Destination address. This is the address of the router receiving the response.
sa	Source address. This is the address of the FEP sending the response to the router.
rif	Routing information field.
tx	Router sent message to the FEP.

**Table 2-113 Debug SDLLC Field Descriptions (Continued)**

Field	Description
short xid	Router sent the null XID to the FEP.
dsap	Destination service access point
ssap	Source service access point.
tx long xid	Router sent the XID type 2 to the FEP.
Rcvd	Router received Layer 2 message from the FEP.
SABME/LINKUP_REQ	Set asynchronous Balanced Mode Extended command.
partner =	Partner address.

The following line indicates that an explorer frame response was received by the router at address 4000.2000.1001 from the FEP at address C000.1020.1000 with the specified RIF. The original explorer sent to the FEP from the router is not monitored as part of the **debug sdlc** command.

```
SDLLC: rx explorer rsp, da 4000.2000.1001, sa C000.1020.1000, rif
      8840.0011.00A1.0050
```

The following line indicates that the router sent the null XID (Type 0) to the FEP. The debugging information does not include the response to the XID message sent by the FEP to the router.

```
SDLLC: tx short xid, sa 4000.2000.1001, da C000.1020.1000, rif
      88C0.0011.00A1.0050, dsap 4 ssap 4
```

The following line indicates that the router sent the XID command (Format 0 Type 2) to the FEP:

```
SDLLC: tx long xid, sa 4000.2000.1001, da C000.1020.1000, rif
      88C0.0011.00A1.0050, dsap 4 ssap 4
```

The following line is the SABME response to the XID command previously sent by the router to the FEP:

```
Rcvd SABME/LINKUP_REQ pak from TR host
```