



Debug Commands

This chapter contains an alphabetical listing of the **debug** commands and their descriptions. Documentation for each command includes a brief description of its use, command syntax, usage guidelines, sample output, and a description of that output.

Output formats vary with each **debug** command. Some commands generate a single line of output per packet, whereas others generate multiple lines of output per packet. Some generate large amounts of output; others generate only occasional output. Some generate lines of text, and others generate information in field format. Thus, the way **debug** command output is documented also varies. For example, the output for **debug** commands that generate lines of text is usually described line by line, and the output for **debug** commands that generate information in field format is usually described in tables.

By default, the network server sends the output from the **debug** commands to the console. Sending output to a terminal (virtual console) produces less overhead than sending it to the console. Use the **terminal monitor** privileged EXEC command to send output to a terminal. For more information about redirecting output, see the “Using Debug Commands” chapter.

debug aaa accounting

To display information on accountable events as they occur, use the **debug aaa accounting** command in privileged EXEC mode. To disable debugging output, use the **no** form of the command.

debug aaa accounting

no debug aaa accounting

Syntax Description This command has no arguments or keywords.

Command Modes Privileged EXEC

Usage Guidelines The information displayed by the **debug aaa accounting** command is independent of the accounting protocol used to transfer the accounting information to a server. Use the **debug tacacs** and **debug radius** protocol-specific commands to get more detailed information about protocol-level issues.

You can also use the **show accounting** command to step through all active sessions and to print all the accounting records for actively accounted functions. The **show accounting** command allows you to display the active “accountable events” on the system. It provides systems administrators a quick look at what is happening, and may also be useful for collecting information in the event of a data loss of some kind on the accounting server. The **show accounting** command displays additional data on the internal state of the authentication, authorization, and accounting (AAA) security system if **debug aaa accounting** is turned on as well.

Examples The following is sample output from the **debug aaa accounting** command:

```
Router# debug aaa accounting

16:49:21: AAA/ACCT: EXEC acct start, line 10
16:49:32: AAA/ACCT: Connect start, line 10, glare
16:49:47: AAA/ACCT: Connection acct stop:
task_id=70 service=exec port=10 protocol=telnet address=172.31.3.78 cmd=glare bytes_in=308
bytes_out=76 paks_in=45 paks_out=54 elapsed_time=14
```

| Related Commands | Command | Description |
|------------------|--|---|
| | debug aaa authentication | Displays information on accountable events as they occur. |
| | debug aaa authorization | Displays information on AAA/TACACS+ authorization. |
| | debug radius | Displays information associated with the RADIUS. |
| | debug tacacs | Displays information associated with the TACACS. |

debug aaa authentication

To display information on AAA/Terminal Access Controller Access Control System Plus (TACACS+) authentication, use the **debug aaa authentication** command in privileged EXEC mode. To disable debugging output, use the **no** form of the command.

debug aaa authentication

no debug aaa authentication

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

Use this command to learn the methods of authentication being used and the results of these methods.

Examples

The following is sample output from the **debug aaa authentication** command. A single EXEC login that uses the “default” method list and the first method, TACACS+, is displayed. The TACACS+ server sends a GETUSER request to prompt for the username and then a GETPASS request to prompt for the password, and finally a PASS response to indicate a successful login. The number 50996740 is the session ID, which is unique for each authentication. Use this ID number to distinguish between different authentications if several are occurring concurrently.

```
Router# debug aaa authentication

6:50:12: AAA/AUTHEN: create_user user='' ruser='' port='tty19' rem_addr='172.31.60.15'
authen_type=1 service=1 priv=1
6:50:12: AAA/AUTHEN/START (0): port='tty19' list='' action=LOGIN service=LOGIN
6:50:12: AAA/AUTHEN/START (0): using "default" list
6:50:12: AAA/AUTHEN/START (50996740): Method=TACACS+
6:50:12: TAC+ (50996740): received authen response status = GETUSER
6:50:12: AAA/AUTHEN (50996740): status = GETUSER
6:50:15: AAA/AUTHEN/CONT (50996740): continue_login
6:50:15: AAA/AUTHEN (50996740): status = GETUSER
6:50:15: AAA/AUTHEN (50996740): Method=TACACS+
6:50:15: TAC+: send AUTHEN/CONT packet
6:50:15: TAC+ (50996740): received authen response status = GETPASS
6:50:15: AAA/AUTHEN (50996740): status = GETPASS
6:50:20: AAA/AUTHEN/CONT (50996740): continue_login
6:50:20: AAA/AUTHEN (50996740): status = GETPASS
6:50:20: AAA/AUTHEN (50996740): Method=TACACS+
6:50:20: TAC+: send AUTHEN/CONT packet
6:50:20: TAC+ (50996740): received authen response status = PASS
6:50:20: AAA/AUTHEN (50996740): status = PASS
```

debug aaa authorization

To display information on AAA/TACACS+ authorization, use the **debug aaa authorization** command in privileged EXEC mode. To disable debugging output, use the **no** form of the command.

debug aaa authorization

no debug aaa authorization

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

Use this command to learn the methods of authorization being used and the results of these methods.

Examples

The following is sample output from the **debug aaa authorization** command. In this display, an EXEC authorization for user “carrel” is performed. On the first line, the username is authorized. On the second and third lines, the attribute value (AV) pairs are authorized. The debug output displays a line for each AV pair that is authenticated. Next, the display indicates the authorization method used. The final line in the display indicates the status of the authorization process, which, in this case, has failed.

```
Router# debug aaa authorization

2:23:21: AAA/AUTHOR (0): user='carrel'
2:23:21: AAA/AUTHOR (0): send AV service=shell
2:23:21: AAA/AUTHOR (0): send AV cmd*
2:23:21: AAA/AUTHOR (342885561): Method=TACACS+
2:23:21: AAA/AUTHOR/TAC+ (342885561): user=carrel
2:23:21: AAA/AUTHOR/TAC+ (342885561): send AV service=shell
2:23:21: AAA/AUTHOR/TAC+ (342885561): send AV cmd*
2:23:21: AAA/AUTHOR (342885561): Post authorization status = FAIL
```

The **aaa authorization** command causes a request packet containing a series of AV pairs to be sent to the TACACS daemon as part of the authorization process. The daemon responds in one of the following three ways:

- Accepts the request as is
- Makes changes to the request
- Refuses the request, thereby refusing authorization

[Table 2](#) describes AV pairs associated with the **debug aaa authorization** command that may show up in the debug output.

Table 2 Attribute Value Pairs for Authorization

| Attribute Value | Description |
|-----------------|--|
| service=arap | Authorization for the ARA protocol is being requested. |
| service=shell | Authorization for EXEC startup and command authorization is being requested. |
| service=ppp | Authorization for PPP is being requested. |
| service=slip | Authorization for SLIP is being requested. |

Table 2 Attribute Value Pairs for Authorization (continued)

| Attribute Value | Description |
|------------------|--|
| protocol=lcp | Authorization for LCP is being requested (lower layer of PPP). |
| protocol=ip | Used with service=slip and service=ppp to indicate which protocol layer is being authorized. |
| protocol=ipx | Used with service=ppp to indicate which protocol layer is being authorized. |
| protocol=atalk | Used with service=ppp or service=arap to indicate which protocol layer is being authorized. |
| protocol=vines | Used with service=ppp for VINES over PPP. |
| protocol=unknown | Used for undefined or unsupported conditions. |
| cmd=x | Used with service=shell, if cmd=NULL, this is an authorization request to start an EXEC. If cmd is not NULL, this is a command authorization request and will contain the name of the command being authorized. For example, cmd=telnet. |
| cmd-arg=x | Used with service=shell. When performing command authorization, the name of the command is given by a cmd=x pair for each argument listed. For example, cmd-arg=archie.sura.net. |
| acl=x | Used with service=shell and service=arap. For ARA, this pair contains an access list number. For service=shell, this pair contains an access class number. For example, acl=2. |
| inacl=x | Used with service=ppp and protocol=ip. Contains an IP input access list for SLIP or PPP/IP. For example, inacl=2. |
| outacl=x | Used with service=ppp and protocol=ip. Contains an IP output access list for SLIP or PPP/IP. For example, outacl=4. |
| addr=x | Used with service=slip, service=ppp, and protocol=ip. Contains the IP address that the remote host should use when connecting via SLIP or PPP/IP. For example, addr=172.30.23.11. |
| routing=x | Used with service=slip, service=ppp, and protocol=ip. Equivalent in function to the /routing flag in SLIP and PPP commands. Can either be true or false. For example, routing=true. |
| timeout=x | Used with service=arap. The number of minutes before an ARA session disconnects. For example, timeout=60. |
| autocmd=x | Used with service=shell and cmd=NULL. Specifies an autocommand to be executed at EXEC startup. For example, autocmd=telnet yxz.com. |
| noescape=x | Used with service=shell and cmd=NULL. Specifies a noescape option to the username configuration command. Can be either true or false. For example, noescape=true. |
| nohangup=x | Used with service=shell and cmd=NULL. Specifies a nohangup option to the username configuration command. Can be either true or false. For example, nohangup=false. |

Table 2 *Attribute Value Pairs for Authorization (continued)*

| Attribute Value | Description |
|---------------------|---|
| priv-lvl= <i>x</i> | Used with service=shell and cmd=NULL. Specifies the current privilege level for command authorization as a number from 0 to 15. For example, priv-lvl=15. |
| zonelist= <i>x</i> | Used with service=arap. Specifies an AppleTalk zonelist for ARA. For example, zonelist=5. |
| addr-pool= <i>x</i> | Used with service=ppp and protocol=ip. Specifies the name of a local pool from which to get the address of the remote host. |

debug aaa cache filterserver

To help troubleshoot your filter cache configurations, use the **debug aaa cache filterserver** command in privileged EXEC mode. To disable debugging, use the **no** form of this command.

debug aaa cache filterserver

no debug aaa cache filterserver

Syntax Description This command has no arguments or keywords.

Command Modes Privileged EXEC

| Command History | Release | Modification |
|-----------------|-----------|------------------------------|
| | 12.2(13)T | This command was introduced. |

Examples The following is sample output from the **debug aaa cache filterserver** command:

```
Router# debug aaa cache filterserver

AAA/FLTSV: need "myfilter" (fetch), call 0x612DAC64
AAA/FLTSV: send req, call 0x612DAC50
AAA/FLTSV: method SERVER_GROUP myradius
AAA/FLTSV: recv reply, call 0x612DAC50 (PASS)
AAA/FLTSV: create cache
AAA/FLTSV: add attr "call-inacl"
AAA/FLTSV: add attr "call-inacl"
AAA/FLTSV: add attr "call-inacl"
AAA/FLTSV: skip attr "filter-cache-refresh"
AAA/FLTSV: skip attr "filter-cache-time"
AAA/CACHE: set "AAA filtserv cache" entry "myfilter" refresh? no
AAA/CACHE: set "AAA filtserv cache" entry "myfilter" cachetime 15
AAA/FLTSV: add attr to list "call-inacl" call 0x612DAC64
AAA/FLTSV: add attr to list "call-inacl" call 0x612DAC64
AAA/FLTSV: add attr to list "call-inacl" call 0x612DAC64
AAA/FLTSV: PASS call 0x612DAC64
AAA/CACHE: timer "AAA filtserv cache", next in 10 secs (0 entries)
AAA/CACHE: timer "AAA filtserv cache", next in 10 secs (1 entry)
AAA/CACHE: destroy "AAA filtserv cache" entry "myfilter"
AAA/CACHE: timer "AAA filtserv cache", next in 10 secs (0 entries)
```

| Related Commands | Command | Description |
|------------------|---|---|
| | aaa authorization cache filterserver | Enables AAA authorization caches and the downloading of ACL configurations from a RADIUS filter server. |

debug aaa pod

To display debug messages related to Packet Of Disconnect (POD) packets, use the **debug aaa pod** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug aaa pod

no debug aaa pod

Syntax Description This command has no keywords or arguments.

Defaults Debugging for POD packets is not enabled.

Command Modes Privileged EXEC

| Command History | Release | Modification |
|-----------------|------------|---|
| | 12.1(3)T | This command was introduced. |
| | 12.2(2)XB | Support for the voice applications as well as support for the Cisco AS5350, Cisco AS5400 and the Cisco 3600 series was added. |
| | 12.2(2)XB1 | Support for the Cisco AS5800 was added. |
| | 12.2(11)T | This command was integrated into Cisco IOS Release 12.2(11)T, and support for the Cisco AS5850 was added. |

Examples The following example shows output from a successful POD request, when using the **show debug** command:

```
Router# debug aaa pod

AAA POD packet processing debugging is on

Router# show debug

General OS:
  AAA POD packet processing debugging is on
Router#
Apr 25 17:15:59.318:POD:172.19.139.206 request queued
Apr 25 17:15:59.318:voice_pod_request:
Apr 25 17:15:59.318:voip_populate_pod_attr_list:
Apr 25 17:15:59.318:voip_pod_get_guid:
Apr 25 17:15:59.318:voip_pod_get_vsa_attr_val:
Apr 25 17:15:59.318:voip_pod_get_vsa_attr_val:attr_len=50
Apr 25 17:15:59.318:voip_pod_get_vsa_attr_val:attr=h323-conf-id
Apr 25 17:15:59.318:voip_pod_get_vsa_attr_val:attr_len=50 value_len=35
Apr 25 17:15:59.318:voip_pod_get_guid:conf-id=FFA7785F F7F607BB
00000000 993FB1F4 n_bytes=35
Apr 25 17:15:59.318:voip_pod_get_guid:GUID = FFA7785F F7F607BB 00000000
```

```
993FB1F4
Apr 25 17:15:59.318:voip_populate_pod_attr_list:
Apr 25 17:15:59.318:voip_pod_get_vsa_attr_val:
Apr 25 17:15:59.318:voip_pod_get_vsa_attr_val:attr_len=23
Apr 25 17:15:59.318:voip_pod_get_vsa_attr_val:attr=h323-originate
Apr 25 17:15:59.318:voip_pod_get_vsa_attr_val:attr_len=23 value_len=6
Apr 25 17:15:59.318:voip_get_call_direction:
Apr 25 17:15:59.318:voip_get_call_direction:returning answer
Apr 25 17:15:59.318:voip_eval_pod_attr:
Apr 25 17:15:59.318:cc_api_trigger_disconnect:
Apr 25 17:15:59.322:POD:Sending ACK to 172.19.139.206/1700
Apr 25 17:15:59.322:voip_pod_clean:
```

Related Commands

| Command | Description |
|-----------------------|--------------------------|
| aaa pod server | Enables the POD feature. |

debug acircuit

To display errors and events that occur on the attachment circuits (the circuits between the provider edge (PE) and customer edge (CE) routers), use the **debug acircuit** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug acircuit {error | event}

no debug acircuit {error | event}

Syntax Description

| | |
|--------------|---|
| error | Displays any errors that occurred on any of the attachment circuits. |
| event | Displays any event messages for the attachment circuits, including messages about state transitions, interface transitions, and message events. |

Command Modes

Privileged EXEC

Command History

| Release | Modification |
|-----------|---|
| 12.0(23)S | This command was introduced. |
| 12.2(14)S | This command was integrated into Cisco IOS Release 12.2(14)S. |
| 12.2(15)T | This command was integrated into Cisco IOS Release 12.2(15)T. |

Usage Guidelines

An attachment circuit connects a PE router to a CE router. A router can have many attachment circuits. The attachment circuit manager controls all the attachment circuits from one central location. Therefore, when you enable the debug messages for the attachment circuit, you receive information about all the attachment circuits.

Examples

The following is sample output from the **debug acircuit event** command when you enable an interface:

```
Router# debug acircuit event
```

```
*Jan 28 15:19:03.070: ACLIB: ac_cstate() Handling circuit UP for interface Se2/0
*Jan 28 15:19:03.070: ACLIB [11.0.1.1, 200]: pthru_intf_handle_circuit_up() calling
acmgr_circuit_up
*Jan 28 15:19:03.070: ACLIB [11.0.1.1, 200]: Setting new AC state to Ac-Connecting
*Jan 28 15:19:03.070: ACMGR: Receive <Circuit Up> msg
*Jan 28 15:19:03.070: Se2/0 ACMGR: circuit up event, SIP state chg down to connecting,
action is service request
*Jan 28 15:19:03.070: Se2/0 ACMGR: Sent a sip service request
*Jan 28 15:19:03.070: ACLIB [11.0.1.1, 200]: AC updating switch context.
*Jan 28 15:19:03.070: Se2/0 ACMGR: Rcv SIP msg: resp connect forwarded, hdl 9500001D,
l2ss_hdl 700001E
*Jan 28 15:19:03.070: Se2/0 ACMGR: service connected event, SIP state chg connecting to
connected, action is respond forwarded
*Jan 28 15:19:03.070: ACLIB: pthru_intf_response hdl is 9500001D, response is 1
*Jan 28 15:19:03.070: ACLIB [11.0.1.1, 200]: Setting new AC state to Ac-Connected
```

The following is sample output from the **debug acircuit event** command when you disable an interface:

Router# **debug acircuit event**

```
*Jan 28 15:25:57.014: ACLIB: SW AC interface INTF-DOWN for interface Se2/0
*Jan 28 15:25:57.014: ACLIB [11.0.1.1, 200]: Setting new AC state to Ac-Idle
*Jan 28 15:25:57.014: ACLIB: SW AC interface INTF-DOWN for interface Se2/0
*Jan 28 15:25:57.014: Se2/0 ACMGR: Receive <Circuit Down> msg
*Jan 28 15:25:57.014: Se2/0 ACMGR: circuit down event, SIP state chg connected to end,
action is service disconnect
*Jan 28 15:25:57.014: Se2/0 ACMGR: Sent a sip service disconnect
*Jan 28 15:25:57.014: ACLIB [11.0.1.1, 200]: AC deleting switch context.
*Jan 28 15:25:59.014: %LINK-5-CHANGED: Interface Serial2/0, changed state to
administratively down
*Jan 28 15:25:59.014: ACLIB: ac_cstate() Handling circuit DOWN for interface Se2/0
*Jan 28 15:26:00.014:%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial2/0, changed
state to down
```

debug alarm-interface

To show real-time activities in the data channel or the management channel of the Alarm Interface Controller (AIC), use the **debug alarm-interface** command in privileged EXEC mode. To turn off the output, use the **undebug alarm-interface** command.

```
debug alarm-interface slot-number { data | management }
```

```
undebug alarm-interface slot-number { data | management }
```

| Syntax Description | <i>slot-number</i> | Router chassis slot where the AIC network module is installed. |
|--------------------|--------------------|--|
| | data | Displays AIC serial data channel and asynchronous craft port communication activity. |
| | management | Displays IOS-to-AIC communication activity. |

Defaults No default behavior or values.

Command Modes Privileged EXEC

| Command History | Release | Modification |
|-----------------|-----------|--|
| | 12.2(2)XG | This command was introduced for the Cisco 2600 series and the Cisco 3600 series. |
| | 12.2(8)T | This command was integrated into Cisco IOS Release 12.2(8)T. |

Usage Guidelines This command allows you to observe the management channel activity from the AIC in the specified slot. Such activity shows that the software running on the AIC CPU has reached a minimum level of working order.

Examples The following example shows sample output for the **debug alarm-interface 1 management** command:

```
AIC Slot 1:STATUS received
```

The following is sample output for the **debug alarm-interface 1 data** command:

```
AIC Slot 1:STATUS received
aic_fastsend:particle count=1, len=1504
aic_pak_to_txring:scattered particle count=1, tx bytes=1504, leftover=0
aic_interrupt:# 30419 gstar=0x1000000
aic_safe_start:particle count=1, len=524
aic_pak_to_txring:scattered particle count=1, tx bytes=524, leftover=0
aic_process_TXivq:ivq - 0x42040000 at 15, slice 1
aic_interrupt:# 30420 gstar=0x1000000
aic_process_TXivq:ivq - 0x42040000 at 16, slice 1
aic_interrupt:# 30421 gstar=0x10000000
aic_scc_rx_intr:sts_dlen=0xC5E10000, len=1504, RSTA=0xA0
aic_serial_RX_interrupt:rxttype=1, len=1504, aic_scc_rx_intr:last_rxbd has aged, 2
```

```

aic_process_RXivq:ivq - 0x60000    at 13, slice 1
aic_interrupt:# 30422  gstar=0x10000000
aic_scc_rx_intr:sts_dlen=0xC20D0000, len=524, RSTA=0xA0
aic_serial_RX_interrupt:rxttype=1, len=524, aic_process_RXivq:ivq - 0x60000    at 14, slice
1
aic_interrupt:# 30423  gstar=0x20000000
aic_scc_rx_intr:sts_dlen=0xC00D0000, len=12, RSTA=0xA0
aic_mgmt_RX_interrupt:len=12
aic_mgmt_fastsend:particle count=1, len=20 / 20
aic_pak_to_txring:scattered particle count=1, tx bytes=20, leftover=0
aic_scc_rx_intr:last_rxbd has aged, 2
aic_process_RXivq:ivq - 0x10060000 at 37, slice 1
aic_interrupt:# 30424  gstar=0x20000000
aic_process_TXivq:ivq - 0x52040000 at 24, slice 1

```

Related Commands

| Command | Description |
|------------------------|---|
| alarm-interface | Enters the alarm interface mode and configures the AIC. |
| reset | Resets the AIC CPU. |

debug alps ascu

To enable debugging for ALPS ASCUs, use the **debug alps ascu** command in privileged EXEC mode. To disable debugging, use the **no** form of this command.

```
debug alps ascu { event | packet | detail | all | format { ipars | router | both } } [interface [ascu id]]
```

```
no debug alps ascu { event | packet | detail | all | format { ipars | router | both } } [interface [ascu id]]
```

Syntax Description

| | |
|---|---|
| event | Displays ASCU events or protocol errors. |
| packet | Displays sent or received packets. |
| detail | Displays all ASCU protocol events. |
| all | Enables event, packet, and detail debugging. |
| format { ipars router both } | Specifies how to display ASCU addresses and the hexadecimal data in the debug output: <ul style="list-style-type: none"> ipars—Displays the IPARS hexadecimal output, only. router—Displays the router hexadecimal output, only. both—Displays both the IPARS and router hexadecimal output. <p>The only difference between the IPARS output and the router output is the format of the hexadecimal data.</p> |
| <i>interface</i> | (Optional) Enables debugging on a specified interface. Applies only to the event , packet , detail , and all keywords. |
| <i>ascu id</i> | (Optional) Enables debugging for a specified ASCU. |

Defaults

Debugging is off.

Command History

| Release | Modification |
|----------|---|
| 11.3(6)T | This command was introduced for limited availability. |
| 12.0(1) | This command was available for general release. |
| 12.0(5)T | This command was modified. |
| 12.1(2)T | The format , ipars , router , and both keywords were added. The output for this command was modified to include IPARS and router formats. |

Usage Guidelines

To enable debugging for a group of ASCUs, enter a separate command for each ASCU interface and IA combination.

The *interface* option applies only to the **event**, **packet**, **detail**, and **all** keywords.

**Note**

To specify the particular debug tracing level (**event**, **packet**, **detail** or **all**) and the format (router, pairs or both), you must configure the **debug alps ascu** command two times: once to configure the debug tracing level and once to configure the format.

**Note**

To log messages to an internal buffer, use the **logging buffered** global configuration command. To display the state of logging (syslog), use the **show logging** command in privileged EXEC mode. For information on these commands and other commands used to customize logs, refer to the [Cisco IOS Configuration Fundamentals Configuration Guide](#) and [Cisco IOS Configuration Fundamentals Command Reference](#).

Examples

The following output is from the **debug alps ascu event** command, showing events or protocol errors in **router** format for ASCU 42 on interface Serial7:

```
Router# debug alps ascu format router

Router# debug alps ascu event Serial7 42

ALPS ASCU: T1 expired for ascu 42 on i/f Serial7
ALPS ASCU: DOWN event while UP for ascu 42 on i/f Serial7 : C1 count = 1
```

**Note**

If you specify the **ipars** or **both** format for the **event** or **detail** tracing level, both the IPARS and router formats will be displayed.

The following output is from the **debug alps ascu event** command, showing events or protocol errors in **ipars** format for ASCU 42 on interface Serial7:

```
Router# debug alps ascu format ipars

Router# debug alps ascu event Serial7 42

ALPS ASCU: T1 expired for ascu 42/2F on i/f Serial7
ALPS ASCU: DOWN event while UP for ascu 42/2F on i/f Serial7 : C1 count = 1
```

The following output is from the **debug alps ascu detail** command, showing all protocol events in **router** format for ASCU 42 on interface Serial6:

```
Router# debug alps ascu format router

Router# debug alps ascu detail Serial6 42

ALPS ASCU: Tx ALC POLL MSG (+ 0 pad bytes) to ascu 42 on i/f Serial6
ALPS ASCU: ALC GO AHD MSG rcvd from ascu 42 on i/f Serial6
ALPS ASCU: Tx ALC POLL MSG (+ 0 pad bytes) to ascu 42 on i/f Serial6
ALPS ASCU: ALC GO AHD MSG rcvd from ascu 42 on i/f Serial6
ALPS ASCU: Tx ALC POLL MSG (+ 0 pad bytes) to ascu 42 on i/f Serial6
ALPS ASCU: Rx ALC DATA MSG (14 bytes + CCC) from ascu 42 on i/f Serial6, fwd to ckt
RTP_MATIP
ALPS ASCU: ALC GO AHD MSG rcvd from ascu 42 on i/f Serial6
ALPS ASCU: Tx ALC DATA MSG (14 bytes + CCC + 0 pad bytes) to ascu 42 on i/f Serial6
ALPS ASCU: Tx ALC POLL MSG (3 bytes + CCC + 0 pad bytes) to ascu 42 on i/f Serial6
```

**Note**

If you specify the **ipars** or **both** format for the **event** or **detail** tracing level, both the IPARS and router formats will be displayed.

The following output is from the **debug alps ascu detail** command, showing all protocol events in **both** format for ASCU 42 on interface Serial6:

```
Router# debug alps ascu format both

Router# debug alps ascu detail Serial6 42

ALPS ASCU: Tx ALC POLL MSG (+ 0 pad bytes) to ascu 42/2F on i/f Serial6
ALPS ASCU: ALC GO AHD MSG rcvd from ascu 42/2F on i/f Serial6
ALPS ASCU: Tx ALC POLL MSG (+ 0 pad bytes) to ascu 42/2F on i/f Serial6
ALPS ASCU: ALC GO AHD MSG rcvd from ascu 42/2F on i/f Serial6
ALPS ASCU: Tx ALC POLL MSG (+ 0 pad bytes) to ascu 42/2F on i/f Serial6
ALPS ASCU: Rx ALC DATA MSG (14 bytes + CCC) from ascu 42/2F on i/f Serial6, fwd to ckt
RTP_MATIP
ALPS ASCU: ALC GO AHD MSG rcvd from ascu 42/2F on i/f Serial6
ALPS ASCU: Tx ALC DATA MSG (14 bytes + CCC + 0 pad bytes) to ascu 42/2F on i/f Serial6
ALPS ASCU: Tx ALC POLL MSG (3 bytes + CCC + 0 pad bytes) to ascu 42/2F on i/f Serial6
```

The following output is from the **debug alps ascu packet** command, showing all packets sent or received in **router** format for ASCU 42 on interface Serial6:

```
Router# debug alps ascu packet Serial6 42

ALPS ASCU: Tx ALC SERVICE MSG (18 bytes + CCC + 0 pad bytes) to ascu 42 on i/f Serial6
02321D26 0C261616
140C0D18 26163135 0611C6
ALPS ASCU: Rx ALC DATA MSG (14 bytes + CCC) from ascu 42 on i/f Serial6, fwd ckt
RTP_MATIP
42607866 65717866
65717966 755124
ALPS ASCU: Tx ALC DATA MSG (14 bytes + CCC + 0 pad bytes) to ascu 42 on i/f Serial6
022038 26253138
26253139 263511E4
```

The following output is from the **debug alps ascu packet** command, showing all packets sent or received in **ipars** format for ASCU 42 on interface Serial6:

```
Router# debug alps ascu packet Serial6 42

ALPS ASCU: Tx ALC SERVICE MSG (18 bytes + CCC + 0 pad bytes) to ascu 42/2F on i/f Serial6
ALPS IPARS Format:
2F2C1126 33262525
35331339 26251C14 271DC6
ALPS ASCU: Rx ALC DATA MSG (14 bytes + CCC) from ascu 42/2F on i/f Serial6, fwd ckt
RTP_MATIP
ALPS IPARS Format:
2F3E3826 161C3826
161C1826 141D24
ALPS ASCU: Tx ALC DATA MSG (14 bytes + CCC + 0 pad bytes) to ascu 42/2F on i/f Serial6
ALPS IPARS Format:
2F3E38 26161C38
26161C18 26141DE4
```

The following output is from the **debug alps ascu packet** command, showing all packets sent or received in **both** format for ASCU 42 on interface Serial6:

```
Router# debug alps ascu packet Serial6 42
```

```
ALPS ASCU: Tx ALC SERVICE MSG (18 bytes + CCC + 0 pad bytes) to ascu 42/2F on i/f Serial6
ALPS Router Format:
02321D26 0C261616
140C0D18 26163135 0611C6
ALPS IPARS Format:
2F2C1126 33262525
35331339 26251C14 271DC6
ALPS ASCU: Rx ALC DATA MSG (14 bytes + CCC) from ascu 42/2F on i/f Serial6, fwd ckt
RTP_MATIP
ALPS Router Format:
42607866 65717866
65717966 755124
ALPS IPARS Format:
2F3E3826 161C3826
161C1826 141D24
ALPS ASCU: Tx ALC DATA MSG (14 bytes + CCC + 0 pad bytes) to ascu 42/2F on i/f Serial6
ALPS Router Format:
022038 26253138
26253139 263511E4
ALPS IPARS Format:
2F3E38 26161C38
26161C18 26141DE4
```

debug alps circuit event

To enable event debugging for ALPS circuits, use the **debug alps circuit event** command in privileged EXEC mode. To disable debugging, use the **no** form of this command.

debug alps circuit event [*name*]

no debug alps circuit event [*name*]

| Syntax Description | <i>name</i> (Optional) Name given to identify an ALPS circuit on the remote CPE. |
|--------------------|--|
|--------------------|--|

| Defaults | If no circuit name is specified, then debugging is enabled for every ALPS circuit. |
|----------|--|
|----------|--|

| Command History | Release | Modification |
|-----------------|---------|------------------------------|
| | 11.3 T | This command was introduced. |

| Usage Guidelines | To enable debugging for a single ALPS circuit, specify the name of the circuit. To enable debugging for a group of circuits, enter a separate command for each circuit name. |
|------------------|---|
|------------------|---|

| Examples | The following is sample output from the debug alps circuit event command for circuit RTP_AX25: |
|----------|---|
|----------|---|

```
alps-rcpe# debug alps circuit event RTP_AX25

ALPS P1024 CKT: FSM - Ckt= RTP_AX25, State= OPEN, Event= DISABLE:
(CloseAndDisable)->DISC
ALPS P1024 CKT: FSM - Ckt= RTP_AX25, State= DISC, Event= ENABLE:
(TmrStartNullRetry)->INOP
ALPS P1024 CKT: Ckt= RTP_AX25, Open - peer set to 200.100.40.2
ALPS P1024 CKT: Ckt= RTP_AX25, Open - peer open.
ALPS P1024 CKT: FSM - Ckt= RTP_AX25, State= INOP, Event= RETRY_TIMEOUT:
(Open)->OPNG
ALPS P1024 CKT: FSM - Ckt= RTP_AX25, State= OPNG, Event= CKT_OPEN_CFM:
(CacheAndFwdAscuData)->OPEN

alps-ccpe# debug alps circuit event RTP_AX25

ALPS AX.25 FSM: Ckt= RTP_AX25, State= OPEN, Event= CktClose, Rsn= 12:
(PvcKill,CktRemove,TmrStartClose)->INOP
ALPS AX.25 FSM: Ckt= RTP_AX25, State= INOP, Event= X25PvcInact, Rsn= 0:
(-,-,-)->INOP
ALPS AX.25 FSM: Ckt= RTP_AX25, State= INOP, Event= X25VcDeleted, Rsn= 0:
(-, CktDestroy,TmrStop)->INOP
ALPS AX.25 FSM: Ckt= RTP_AX25, State= INOP, Event= CktOpReq, Rsn= 4:
(PvcMake,CktAdd,TmrStartOpen)->OPNG
ALPS AX.25 FSM: Ckt= RTP_AX25, State= OPNG, Event= X25ResetTx, Rsn= 0:
(-,-,-)->OPNG
ALPS AX.25 FSM: Ckt= RTP_AX25, State= OPNG, Event= X25VcUp, Rsn= 0:
(-,OpnCfm,TmrStop)->OPEN
```

debug alps peer

To enable event or packet debugging for ALPS peers, use the **debug alps peer** command in privileged EXEC mode. To disable debugging, use the **no** form of this command.

```
debug alps peer { event | packet } [ip-address]
```

```
no debug alps peer { event | packet } [ip-address]
```

| Syntax Description | | |
|--------------------|-------------------|------------------------------------|
| | event | Specifies debugging for an event. |
| | packet | Specifies debugging for a packet. |
| | <i>ip-address</i> | (Optional) Remote peer IP address. |

Defaults If no IP address is specified, then debugging is enabled for every peer connection.

| Command History | Release | Modification |
|-----------------|----------|--|
| | 11.3(6)T | This command was introduced for limited availability. |
| | 12.0(1) | This command was available for general release. |
| | 12.0(5)T | The packet keyword was added. The format for the output was modified for consistency. |

Usage Guidelines To enable debugging for a single remote ALPS peer, specify the peer IP address.
To enable debugging for a set of remote peers, enter the command for each peer IP address.

Examples The following output is from the **debug alps peer packet** command:

```
Router# debug alps peer packet

ALPS PEER:Peer (10.227.50.106, MATIP_A_CKT-1) - TX Peer Data Msg (18 bytes)
040A5320:                01 00001241
040A5330:45546B5F 6F4F7757 67477B5B 51
ALPS PEER:Peer (10.227.50.106, MATIP_A_CKT-1) - RX Peer Data Msg (18 bytes)
04000550:                01000012 4145546B 5F6F4F77
04000560:5767477B 5B51
ALPS PEER:Peer (10.227.50.106, MATIP_A_CKT-1) - TX Peer Data Msg (18 bytes)
0409F6E0:                01 00001241 45546B5F
0409F6F0:6F4F7757 67477B5B 51
ALPS PEER:Peer (10.227.50.106, MATIP_A_CKT-1) - RX Peer Data Msg (18 bytes)
04000680:                01000012 4145546B
04000690:5F6F4F77 5767477B 5B51
```

debug alps peer event

To enable event debugging for ALPS peers, use the **debug alps peer event** command in privileged EXEC mode. To disable debugging, use the **no** form of this command.

debug alps peer event *ipaddr*

no debug alps peer event *ipaddr*

| | | |
|---------------------------|---------------|------------------|
| Syntax Description | <i>ipaddr</i> | Peer IP address. |
|---------------------------|---------------|------------------|

| | |
|-----------------|---|
| Defaults | If no IP address is specified, then debugging is enabled for every peer connection. |
|-----------------|---|

| Command History | Release | Modification |
|------------------------|----------------|------------------------------|
| | 11.3 T | This command was introduced. |

| | |
|-------------------------|---|
| Usage Guidelines | To enable debugging for a single remote ALPS peer, specify the peer IP address. To enable debugging for a set of remote peers, enter the command for each peer IP address. |
|-------------------------|---|

| | |
|-----------------|---|
| Examples | The following is sample output from the debug alps peer event command: |
|-----------------|---|

```
Router# debug alps peer event

ALPS PEER: FSM - Peer 200.100.25.2, Event ALPS_CLOSED_IND, State OPENED
ALPS PEER: peer 200.100.25.2 closed - closing peer circuits.
ALPS PEER: Promiscuous peer created for 200.100.25.2
ALPS PEER: TCP Listen - passive open 200.100.25.2(11003) -> 10000
ALPS PEER: FSM - Peer 200.100.25.2, Event ALPS_OPEN_IND, State DISCONN
ALPS PEER: peer 200.100.25.2 opened OK.
```

debug alps snmp

To enable debugging for ALPS SNMP agents, use the **debug alps snmp** command in privileged EXEC mode. To disable debugging, use the **no** form of this command.

debug alps snmp

no debug alps snmp

Syntax Description This command has no arguments or keywords.

Defaults Debugging for SNMP agents is not enabled.

| Command History | Release | Modification |
|-----------------|----------|---|
| | 11.3(6)T | This command was introduced for limited availability. |
| | 12.0(1)T | This command was available for general release. |
| | 12.0(5)T | This command was added to the documentation. |
| | 12.1(2)T | The output for this command was modified to reflect MIB and SNMP changes. |

Examples The following output is from the **debug alps snmp** command. The first line shows a circuit event status change. The second line shows an ASCU status change. The third line shows a peer connection status change.

```
ALPS CktStatusChange Notification for circuit CKT-1
ALPS AscuParamChange Notification for ascu (Serial3, 41)
PeerConnStatusChange Notification for peer (10.227.50.106, MATIP_A_CKT-1)
```

The following output is from the **debug alps snmp** command, showing that an open failure has occurred on circuit 1:

```
ALPS CktOpenFailure Notification for circuit CKT1
```

The following output is from the **debug alps snmp** command, showing that a partial rejection to an ALPS circuit peer open request has occurred on circuit 1:

```
ALPS CktPartialReject Notification for ascu (Serial2, 41) on circuit CKT1
```

debug apple arp

To enable debugging of the AppleTalk Address Resolution Protocol (ARP), use the **debug apple arp** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug apple arp [*type number*]

no debug apple arp [*type number*]

| Syntax Description | |
|--------------------|------------------------------|
| <i>type</i> | (Optional) Interface type. |
| <i>number</i> | (Optional) Interface number. |

Usage Guidelines This command is helpful when you experience problems communicating with a node on the network you control (a neighbor). If the **debug apple arp** display indicates that the router is receiving ARP probes, you can assume that the problem does not reside at the physical layer.

Examples The following is sample output from the **debug apple arp** command:

```
Router# debug apple arp

Ether0: AARP: Sent resolve for 4160.26
Ether0: AARP: Reply from 4160.26(0000.0c00.0453) for 4160.154(0000.0c00.8ea9)
Ether0: AARP: Resolved waiting request for 4160.26(0000.0c00.0453)
Ether0: AARP: Reply from 4160.19(0000.0c00.0082) for 4160.154(0000.0c00.8ea9)
Ether0: AARP: Resolved waiting request for 4160.19(0000.0c00.0082)
Ether0: AARP: Reply from 4160.19(0000.0c00.0082) for 4160.154(0000.0c00.8ea9)
```

Explanations for representative lines of output follow.

The following line indicates that the router has requested the hardware MAC address of the host at network address 4160.26:

```
Ether0: AARP: Sent resolve for 4160.26
```

The following line indicates that the host at network address 4160.26 has replied, giving its MAC address (0000.0c00.0453). For completeness, the message also shows the network address to which the reply was sent and its hardware MAC address (also in parentheses).

```
Ether0: AARP: Reply from 4160.26(0000.0c00.0453) for 4160.154(0000.0c00.8ea9)
```

The following line indicates that the MAC address request is complete:

```
Ether0: AARP: Resolved waiting request for 4160.26(0000.0c00.0453)
```

debug apple domain

To enable debugging of the AppleTalk domain activities, use the **debug apple domain** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug apple domain

no debug apple domain

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

Use the **debug apple domain** command to observe activity for domains and subdomains. Use this command in conjunction with the **debug apple remap** command to observe interaction between remapping and domain activity. Messages are displayed when the state of a domain changes, such as creating a new domain, deleting a domain, and updating a domain.

Examples

The following is sample output from the **debug apple domain** command intermixed with output from the **debug apple remap** command; the two commands show related events:

```
Router# debug apple domain

Router# debug apple remap

AT-REMAP: RemapProcess for net 30000 domain AURP Domain 1
AT-REMAP: ReshuffleRemapList for subdomain 1
AT-REMAP: Could not find a remap for cable 3000-3001
AT-DOMAIN: atdomain_DisablePort for Tunnel0
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: Disabling interface Ethernet1
AT-DOMAIN: atdomain_DisablePort for Ethernet1
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: CleanSubDomain for inbound subdomain 1
AT-REMAP: Remap for net 70 inbound subdomain 1 has been deleted
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: CleanSubDomain for outbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-REMAP: RemapProcess for net 30000 domain AURP Domain 1 Remapped Net 10000
AT-REMAP: Remap for net 50 outbound subdomain 1 has been deleted
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: CleanSubDomain for inbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: CleanSubDomain for outbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
```

Related Commands

| Command | Description |
|-----------------------------------|--|
| debug apple remap | Enables debugging of the AppleTalk remap activities. |

debug apple errors

To display errors occurring in the AppleTalk network, use the **debug apple errors** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug apple errors [*type number*]

no debug apple errors [*type number*]

Syntax Description

| | |
|---------------|------------------------------|
| <i>type</i> | (Optional) Interface type. |
| <i>number</i> | (Optional) Interface number. |

Usage Guidelines

In a stable AppleTalk network, the **debug apple errors** command produces little output.

To solve encapsulation problems, enable **debug apple errors** and **debug apple packet** together.

Examples

The following is sample output from the **debug apple errors** command when a router is brought up with a zone that does not agree with the zone list of other routers on the network:

```
Router# debug apple errors

%AT-3-ZONEDISAGREES: Ethernet0: AppleTalk port disabled; zone list incompatible with
4160.19
%AT-3-ZONEDISAGREES: Ethernet0: AppleTalk port disabled; zone list incompatible with
4160.19
%AT-3-ZONEDISAGREES: Ethernet0: AppleTalk port disabled; zone list incompatible with
4160.19
```

As the output suggests, a single error message indicates zone list incompatibility; this message is sent out periodically until the condition is corrected or the **debug apple errors** command is turned off.

Most of the other messages that the **debug apple errors** command can generate are obscure or indicate a serious problem with the AppleTalk network. Some of these other messages follow.

In the following message, RTMPReq, RTMPReq, ATP, AEP, ZIP, ADSP, or SNMP could replace NBP, and “llap dest not for us” could replace “wrong encapsulation”:

```
Packet discarded, src 4160.12-254,dst 4160.19-254,NBP,wrong encapsulation
```

In the following message, in addition to an invalid echo packet error, other possible errors are unsolicited AEP echo reply, unknown echo function, invalid ping packet, unknown ping function, and bad responder packet type:

```
Ethernet0: AppleTalk packet error; no source address available
AT: pak_reply: dubious reply creation, dst 4160.19
AT: Unable to get a buffer for reply to 4160.19

Processing error, src 4160.12-254,dst 4160.19-254,AEP, invalid echo packet
```

The **debug apple errors** command can print out additional messages when other debugging commands are also turned on. When you turn on both the **debug apple errors** and **debug apple events** commands, the following message can be generated:

```
Proc err, src 4160.12-254,dst 4160.19-254,ZIP,NetInfo Reply format is invalid
```

In the preceding message, in addition to the NetInfo Reply format is invalid error, other possible errors are NetInfoReply not for me, NetInfoReply ignored, NetInfoReply for operational net ignored, NetInfoReply from invalid port, unexpected NetInfoReply ignored, cannot establish primary zone, no primary has been set up, primary zone invalid, net information mismatch, multicast mismatch, and zones disagree.

When you turn on both the **debug apple errors** and **debug apple nbp** commands, the following message can be generated:

```
Processing error,...,NBP,NBP name invalid
```

In the preceding message, in addition to the NBP name invalid error, other possible errors are NBP type invalid, NBP zone invalid, not operational, error handling brrq, error handling proxy, NBP fwdreq unexpected, No route to srcnet, Proxy to "*" zone, Zone "*" from extended net, No zone info for "*", and NBP zone unknown.

When you turn on both the **debug apple errors** and **debug apple routing** commands, the following message can be generated:

```
Processing error,...,RTMPReq, unknown RTMP request
```

In the preceding message, in addition to an unknown RTMP request error, other possible errors are RTMP packet header bad, RTMP cable mismatch, routed RTMP data, RTMP bad tuple, and Not Req or Rsp.

debug apple events

To display information about AppleTalk special events, neighbors becoming reachable or unreachable, and interfaces going up or down, use the **debug apple events** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug apple events [*type number*]

no debug apple events [*type number*]

Syntax Description

| | |
|---------------|------------------------------|
| <i>type</i> | (Optional) Interface type. |
| <i>number</i> | (Optional) Interface number. |

Usage Guidelines

Only significant events (for example, neighbor and route changes) are logged.

The **debug apple events** command is useful for solving AppleTalk network problems because it provides an overall picture of the stability of the network. In a stable network, the **debug apple events** command does not return any information. If the command generates numerous messages, those messages can indicate possible sources of the problems.

When configuring or making changes to a router or interface for AppleTalk, enable the **debug apple events** command to alert you to the progress of the changes or to any errors that might result. Also use this command periodically when you suspect network problems.

The **debug apple events** command is also useful to determine whether network flapping (nodes toggling online and offline) is occurring. If flapping is excessive, look for routers that only support 254 networks.

When you enable the **debug apple events** command, you will see any messages that the **apple event-logging** configuration command normally displays. Turning on the **debug apple events** command, however, does not cause the **apple event-logging** command to be maintained in nonvolatile memory. Only turning on the **apple event-logging** command explicitly stores it in nonvolatile memory. Furthermore, if the **apple event-logging** command is already enabled, turning on or off the **debug apple events** command does not affect the **apple event-logging** command.

Examples

The following is sample output from the **debug apple events** command that describes a nonseed router coming up in discovery mode:

```

router# debug apple events

Discovery mode state changes
Ether0: AT: Resetting interface address filters
%AT-5-INTRESTART: Ether0: AppleTalk port restarting; protocol restarted
Ether0: AppleTalk state changed; unknown -> restarting
Ether0: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 65401.148
Ether0: AppleTalk state changed; probing -> acquiring
%AT-6-ACQUIREMODE: Ether0: AT port initializing; acquiring net configuration
Ether0: AppleTalk state changed; acquiring -> restarting
Ether0: AppleTalk state changed; restarting -> line down
Ether0: AppleTalk state changed; line down -> restarting
Ether0: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 4160.148
Ether0: AppleTalk state changed; probing -> acquiring
%AT-6-ACQUIREMODE: Ether0: AT port initializing; acquiring net configuration
Ether0: AppleTalk state changed; acquiring -> requesting zones
Ether0: AT: Resetting interface address filters
%AT-5-INTRESTART: Ether0: AppleTalk port restarting; protocol restarted
Ether0: AppleTalk state changed; requesting zones -> verifying
AT: Sent GetNetInfo request broadcast on Ethernet0
Ether0: AppleTalk state changed; verifying -> checking zones
Ether0: AppleTalk state changed; checking zones -> operational

```

As the output shows, the **debug apple events** command is useful in tracking the discovery mode state changes through which an interface progresses. When no problems are encountered, the state changes progress as follows:

1. Line down.
2. Restarting.
3. Probing (for its own address [node ID] using AARP).
4. Acquiring (sending out GetNetInfo requests).
5. Requesting zones (the list of zones for its cable).
6. Verifying (that the router's configuration is correct. If not, a port configuration mismatch is declared).
7. Checking zones (to make sure its list of zones is correct).
8. Operational (participating in routing).

Explanations for individual lines of output follow.

The following message indicates that a port is set. In this case, the zone multicast address is being reset.

```
Ether0: AT: Resetting interface address filters
```

The following messages indicate that the router is changing to restarting mode:

```
%AT-5-INTRESTART: Ether0: AppleTalk port restarting; protocol restarted
Ether0: AppleTalk state changed; unknown -> restarting
```

The following message indicates that the router is probing in the startup range of network numbers (65280 to 65534) to discover its network number:

```
Ether0: AppleTalk state changed; restarting -> probing
```

The following message indicates that the router is enabled as a nonrouting node using a provisional network number within its startup range of network numbers. This type of message only appears if the network address the router will use differs from its configured address. This is always the case for a discovery-enabled router; it is rarely the case for a nondiscovery-enabled router.

```
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 65401.148
```

The following messages indicate that the router is sending out GetNetInfo requests to discover the default zone name and the actual network number range in which its network number can be chosen:

```
Ether0: AppleTalk state changed; probing -> acquiring
%AT-6-ACQUIREMODE: Ether0: AT port initializing; acquiring net configuration
```

Now that the router has acquired the cable configuration information, the following message indicates that it restarts using that information:

```
Ether0: AppleTalk state changed; acquiring -> restarting
```

The following messages indicate that the router is probing for its actual network address:

```
Ether0: AppleTalk state changed; restarting -> line down
Ether0: AppleTalk state changed; line down -> restarting
Ether0: AppleTalk state changed; restarting -> probing
```

The following message indicates that the router has found an actual network address to use:

```
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 4160.148
```

The following messages indicate that the router is sending out GetNetInfo requests to verify the default zone name and the actual network number range from which its network number can be chosen:

```
Ether0: AppleTalk state changed; probing -> acquiring
%AT-6-ACQUIREMODE: Ether0: AT port initializing; acquiring net configuration
```

The following message indicates that the router is requesting the list of zones for its cable:

```
Ether0: AppleTalk state changed; acquiring -> requesting zones
```

The following messages indicate that the router is sending out GetNetInfo requests to make sure its understanding of the configuration is correct:

```
Ether0: AppleTalk state changed; requesting zones -> verifying
AT: Sent GetNetInfo request broadcast on Ethernet0
```

The following message indicates that the router is rechecking its list of zones for its cable:

```
Ether0: AppleTalk state changed; verifying -> checking zones
```

The following message indicates that the router is now fully operational as a routing node and can begin routing:

```
Ether0: AppleTalk state changed; checking zones -> operational
```

The following shows sample **debug apple events** output that describes a nondiscovery-enabled router coming up when no other router is on the wire.

```
router# debug apple events

Ethernet1: AT: Resetting interface address filters
%AT-5-INTRESTART: Ethernet1: AppleTalk port restarting; protocol restarted
Ethernet1: AppleTalk state changed; unknown -> restarting
Ethernet1: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: Ethernet1: AppleTalk node up; using address 4165.204
Ethernet1: AppleTalk state changed; probing -> verifying
AT: Sent GetNetInfo request broadcast on Ethernet1
Ethernet1: AppleTalk state changed; verifying -> operational
%AT-6-ONLYROUTER: Ethernet1: AppleTalk port enabled; no neighbors found
```

Indicates a nondiscovery-enabled router with no other router on the wire

S2543

As the output shows, a nondiscovery-enabled router can come up when no other router is on the wire; however, it must assume that its configuration (if accurate syntactically) is correct, because no other router can verify it. Notice that the last line indicates this situation.

The following is sample output from the **debug apple events** command that describes a discovery-enabled router coming up when there is no seed router on the wire:

```
Router# debug apple events

Ether0: AT: Resetting interface address filters
%AT-5-INTRESTART: Ether0: AppleTalk port restarting; protocol restarted
Ether0: AppleTalk state changed; unknown -> restarting
Ether0: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 65401.148
Ether0: AppleTalk state changed; probing -> acquiring
AT: Sent GetNetInfo request broadcast on Ether0
AT: Sent GetNetInfo request broadcast on Ether0
AT: Sent GetNetInfo request broadcast on Ether0
AT: Sent GetNetInfo request broadcast on Ether0
AT: Sent GetNetInfo request broadcast on Ether0
```

As the output shows, when you attempt to bring up a nonseed router without a seed router on the wire, it never becomes operational; instead, it hangs in the acquiring mode and continues to send out periodic GetNetInfo requests.

The following is sample output from the **debug apple events** command when a nondiscovery-enabled router is brought up on an AppleTalk internetwork that is in compatibility mode (set up to accommodate extended as well as nonextended AppleTalk) and the router has violated internetwork compatibility:

```
router# debug apple events

E0: AT: Resetting interface address filters
%AT-5-INTRESTART: E0: AppleTalk port restarting; protocol restarted
E0: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: E0: AppleTalk node up; using address 41.19
E0: AppleTalk state changed; probing -> verifying
AT: Sent GetNetInfo request broadcast on Ethernet0
%AT-3-ZONEDISAGREES: E0: AT port disabled; zone list incompatible with 41.19
AT: Config error for E0, primary zone invalid
E0: AppleTalk state changed; verifying -> config mismatch
```

Indicates configuration mismatch

S2545

The following three configuration command lines indicate the part of the configuration of the router that caused the configuration mismatch:

```
lestat(config)#interface ethernet 0  
lestat(config-if)#apple cab 41-41  
lestat(config-if)#apple zone Marketing
```

The router shown had been configured with a cable range of 41-41 instead of 40-40, which would have been accurate. Additionally, the zone name was configured incorrectly; it should have been “Marketing,” rather than being misspelled as “Markting.”

debug apple nbp

To display debugging output from the Name Binding Protocol (NBP) routines, use the **debug apple nbp** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug apple nbp [*type number*]

no debug apple nbp [*type number*]

Syntax Description

| | |
|---------------|------------------------------|
| <i>type</i> | (Optional) Interface type. |
| <i>number</i> | (Optional) Interface number. |

Usage Guidelines

To determine whether the router is receiving NBP lookups from a node on the AppleTalk network, enable **debug apple nbp** at each node between the router and the node in question to determine where the problem lies.



Caution

Because the **debug apple nbp** command can generate many messages, use it only when the CPU utilization of the router is less than 50 percent.

Examples

The following is sample output from the **debug apple nbp** command:

```
Router# debug apple nbp

AT: NBP ctrl = LkUp, ntuples = 1, id = 77
AT: 4160.19, skt 2, enum 0, name: =:ciscoRouter@Low End SW Lab
AT: LkUp =:ciscoRouter@Low End SW Lab

AT: NBP ctrl = LkUp-Reply, ntuples = 1, id = 77
AT: 4160.154, skt 254, enum 1, name: lestat.Ether0:ciscoRouter@Low End SW Lab

AT: NBP ctrl = LkUp, ntuples = 1, id = 78
AT: 4160.19, skt 2, enum 0, name: =:IPADDRESS@Low End SW Lab
AT: NBP ctrl = LkUp, ntuples = 1, id = 79
AT: 4160.19, skt 2, enum 0, name: =:IPGATEWAY@Low End SW Lab
AT: NBP ctrl = LkUp, ntuples = 1, id = 83
AT: 4160.19, skt 2, enum 0, name: =:ciscoRouter@Low End SW Lab
AT: LkUp =:ciscoRouter@Low End SW Lab

AT: NBP ctrl = LkUp, ntuples = 1, id = 84
AT: 4160.19, skt 2, enum 0, name: =:IPADDRESS@Low End SW Lab

AT: NBP ctrl = LkUp, ntuples = 1, id = 85
AT: 4160.19, skt 2, enum 0, name: =:IPGATEWAY@Low End SW Lab
AT: NBP ctrl = LkUp, ntuples = 1, id = 85
AT: 4160.19, skt 2, enum 0, name: =:IPGATEWAY@Low End SW Lab
```

The first three lines describe an NBP lookup request:

```
AT: NBP ctrl = LkUp, ntuples = 1, id = 77
AT: 4160.19, skt 2, enum 0, name: =:ciscoRouter@Low End SW Lab
AT: LkUp =:ciscoRouter@Low End SW Lab
```

Table 3 describes the fields in the first line of output.

Table 3 *debug apple nbp Field Descriptions—First Line of Output*

| Field | Description |
|-------------|---|
| AT: NBP | Indicates that this message describes an AppleTalk NBP packet. |
| ctrl = LkUp | Identifies the type of NBP packet. Possible values are as follows: <ul style="list-style-type: none"> LkUp—NBP lookup request. LkUp-Reply—NBP lookup reply. |
| ntuples = 1 | Indicates the number of name-address pairs in the lookup request packet. Range: 1 to 31 tuples. |
| id = 77 | Identifies an NBP lookup request value. |

Table 4 describes the fields in the second line of output.

Table 4 *debug apple nbp Field Descriptions—Second Line of Output*

| Field | Description |
|------------------------------------|--|
| AT: | Indicates that this message describes an AppleTalk packet. |
| 4160.19 | Indicates the network address of the requester. |
| skt 2 | Indicates the internet socket address of the requester. The responder will send the NBP lookup reply to this socket address. |
| enum 0 | Indicates the enumerator field. Used to identify multiple names registered on a single socket. Each tuple is assigned its own enumerator, incrementing from 0 for the first tuple. |
| name: =:ciscoRouter@Low End SW Lab | Indicates the entity name for which a network address has been requested. The AppleTalk entity name includes three components: <ul style="list-style-type: none"> Object (in this case, a wildcard character [=], indicating that the requester is requesting name-address pairs for all objects of the specified type in the specified zone). Type (in this case, ciscoRouter). Zone (in this case, Low End SW Lab). |

The third line in the output essentially reiterates the information in the two lines above it, indicating that a lookup request has been made regarding name-address pairs for all objects of the ciscoRouter type in the Low End SW Lab zone.

Because the router is defined as an object of type ciscoRouter in zone Low End SW Lab, the router sends an NBP lookup reply in response to this NBP lookup request. The following two lines of output show the response of the router:

```
AT: NBP ctrl = LkUp-Reply, ntuples = 1, id = 77
AT: 4160.154, skt 254, enum 1, name: lestat.Ether0:ciscoRouter@Low End SW Lab
```

In the first line, `ctrl = LkUp-Reply` identifies this NBP packet as an NBP lookup request. The same value in the `id` field (`id = 77`) associates this lookup reply with the previous lookup request. The second line indicates that the network address associated with the entity name of the router (`lestat.Ether0:ciscoRouter@Low End SW Lab`) is `4160.154`. The fact that no other entity name/network address is listed indicates that the responder only knows about itself as an object of type `ciscoRouter` in zone `Low End SW Lab`.

debug apple packet

To display per-packet debugging output, use the **debug apple packet** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug apple packet [*type number*]

no debug apple packet [*type number*]

Syntax Description

| | |
|---------------|------------------------------|
| <i>type</i> | (Optional) Interface type. |
| <i>number</i> | (Optional) Interface number. |

Usage Guidelines

With this command, you can monitor the types of packets being slow switched. It displays at least one line of debugging output per AppleTalk packet processed.

The output reports information online when a packet is received or a transmission is attempted.

When invoked in conjunction with the **debug apple routing**, **debug apple zip**, and **debug apple nbp** commands, the **debug apple packet** command adds protocol processing information in addition to generic packet details. It also reports successful completion or failure information.

When invoked in conjunction with the **debug apple errors** command, the **debug apple packet** command reports packet-level problems, such as those concerning encapsulation.



Caution

Because the **debug apple packet** command can generate many messages, use it only when the CPU utilization of the router is less than 50 percent.

Examples

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

```
Router# debug apple packet

Ether0: AppleTalk packet: enctype SNAP, size 60, encaps000000000000000000000000
AT: src=Ethernet0:4160.47, dst=4160-4160, size=10, 2 rtes, RTMP pkt sent
AT: ZIP Extended reply rcvd from 4160.19
AT: ZIP Extended reply rcvd from 4160.19
AT: src=Ethernet0:4160.47, dst=4160-4160, size=10, 2 rtes, RTMP pkt sent
Ether0: AppleTalk packet: enctype SNAP, size 60, encaps000000000000000000000000
Ether0: AppleTalk packet: enctype SNAP, size 60, encaps000000000000000000000000
```

[Table 5](#) describes the fields in the first line of output.

Table 5 *debug apple packet* Field Descriptions—First Line of Output

| Field | Description |
|------------------|---|
| Ether0: | Name of the interface through which the router received the packet. |
| AppleTalk packet | Indicates that this is an AppleTalk packet. |
| enctype SNAP | Encapsulation type for the packet. |

Table 5 *debug apple packet Field Descriptions—First Line of Output (continued)*

| Field | Description |
|--------------------------------|--------------------------------|
| size 60 | Size of the packet (in bytes). |
| encaps000000000000000000000000 | Encapsulation. |

Table 6 describes the fields in the second line of output.

Table 6 *debug apple packet Field Descriptions—Second Line of Output*

| Field | Description |
|-----------------------|--|
| AT: | Indicates that this is an AppleTalk packet. |
| src=Ethernet0:4160.47 | Name of the interface sending the packet and its AppleTalk address. |
| dst=4160-4160 | Cable range of the destination of the packet. |
| size=10 | Size of the packet (in bytes.) |
| 2 rtes | Indicates that two routes in the routing table link these two addresses. |
| RTMP pkt sent | Type of packet sent. |

The third line indicates the type of packet received and its source AppleTalk address. This message is repeated in the fourth line because AppleTalk hosts can send multiple replies to a given GetNetInfo request.

debug apple remap

To enable debugging of the AppleTalk remap activities, use the **debug apple remap** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug apple remap

no debug apple remap

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

Use the **debug apple remap** command with the **debug apple domain** command to observe activity between domains and subdomains. Messages from the **debug apple remap** command are displayed when a particular remapping function occurs, such as creating remaps or deleting remaps.

Examples

The following is sample output from the **debug apple remap** command intermixed with output from the **debug apple domain** command; the two commands show related events.

```
Router# debug apple remap
```

```
Router# debug apple domain
```

```
AT-REMAP: RemapProcess for net 30000 domain AURP Domain 1
AT-REMAP: ReshuffleRemapList for subdomain 1
AT-REMAP: Could not find a remap for cable 3000-3001
AT-DOMAIN: atdomain_DisablePort for Tunnel0
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: Disabling interface Ethernet1
AT-DOMAIN: atdomain_DisablePort for Ethernet1
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: CleanSubDomain for inbound subdomain 1
AT-REMAP: Remap for net 70 inbound subdomain 1 has been deleted
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: CleanSubDomain for outbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-REMAP: RemapProcess for net 30000 domain AURP Domain 1 Remaped Net 10000
AT-REMAP: Remap for net 50 outbound subdomain 1 has been deleted
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: CleanSubDomain for inbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: CleanSubDomain for outbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
```

Related Commands

| Command | Description |
|------------------------------------|---|
| debug apple domain | Enables debugging of the AppleTalk domain activities. |

debug apple routing

To enable debugging output from the Routing Table Maintenance Protocol (RTMP) routines, use the **debug apple routing** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug apple routing [*type number*]

no debug apple routing [*type number*]

Syntax Description

| | |
|---------------|------------------------------|
| <i>type</i> | (Optional) Interface type. |
| <i>number</i> | (Optional) Interface number. |

Usage Guidelines

This command can be used to monitor acquisition of routes, aging of routing table entries, and advertisement of known routes. It also reports conflicting network numbers on the same network if the network is misconfigured.



Caution

Because the **debug apple routing** command can generate many messages, use it only when router CPU utilization is less than 50 percent.

Examples

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

```
Router# debug apple routing

AT: src=Ethernet0:4160.41, dst=4160-4160, size=19, 2 rtes, RTMP pkt sent
AT: src=Ethernet1:41069.25, dst=41069, size=427, 96 rtes, RTMP pkt sent
AT: src=Ethernet2:4161.23, dst=4161-4161, size=427, 96 rtes, RTMP pkt sent
AT: Route ager starting (97 routes)
AT: Route ager finished (97 routes)
AT: RTMP from 4160.19 (new 0,old 94,bad 0,ign 0, dwn 0)
AT: RTMP from 4160.250 (new 0,old 0,bad 0,ign 2, dwn 0)
AT: RTMP from 4161.236 (new 0,old 94,bad 0,ign 1, dwn 0)
AT: src=Ethernet0:4160.41, dst=4160-4160, size=19, 2 rtes, RTMP pkt sent
```

[Table 7](#) describes the fields in the first line of sample **debug apple routing** output.

Table 7 debug apple routing Field Descriptions

| Field | Description |
|-----------------------|---|
| AT: | Indicates that this is AppleTalk debugging output. |
| src=Ethernet0:4160.41 | Indicates the source router interface and network address for the RTMP update packet. |
| dst=4160-4160 | Indicates the destination network address for the RTMP update packet. |

Table 7 *debug apple routing Field Descriptions (continued)*

| Field | Description |
|---------------|--|
| size=19 | Displays the size of this RTMP packet (in bytes). |
| 2 rtes | Indicates that this RTMP update packet includes information on two routes. |
| RTMP pkt sent | Indicates that this type of message describes an RTMP update packet that the router has sent (rather than one that it has received). |

The following two messages indicate that the ager has started and finished the aging process for the routing table and that this table contains 97 entries:

```
AT: Route ager starting (97 routes)
AT: Route ager finished (97 routes)
```

Table 8 describes the fields in the following line of the **debug apple routing** command output:

```
AT: RTMP from 4160.19 (new 0,old 94,bad 0,ign 0, dwn 0)
```

Table 8 *debug apple routing Field Descriptions*

| Field | Description |
|-------------------|--|
| AT: | Indicates that this is AppleTalk debugging output. |
| RTMP from 4160.19 | Indicates the source address of the RTMP update the router received. |
| new 0 | Displays the number of routes in this RTMP update packet that the router did not already know about. |
| old 94 | Displays the number of routes in this RTMP update packet that the router already knew about. |
| bad 0 | Displays the number of routes the other router indicates have gone bad. |
| ign 0 | Displays the number of routes the other router ignores. |
| dwn 0 | Displays the number of poisoned tuples included in this packet. |

debug apple zip

To display debugging output from the Zone Information Protocol (ZIP) routines, use the **debug apple zip** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug apple zip [*type number*]

no debug apple zip [*type number*]

Syntax Description

| | |
|---------------|------------------------------|
| <i>type</i> | (Optional) Interface type. |
| <i>number</i> | (Optional) Interface number. |

Usage Guidelines

This command reports significant events such as the discovery of new zones and zone list queries. It generates information similar to that generated by the debug apple routing command, but generates it for ZIP packets instead of RTMP packets.

You can use the **debug apple zip** command to determine whether a ZIP storm is taking place in the AppleTalk network. You can detect the existence of a ZIP storm when you see that no router on a cable has the zone name corresponding to a network number that all the routers have in their routing tables.

Examples

The following is sample output from the **debug apple zip** command:

```
Router# debug apple zip
AT: Sent GetNetInfo request broadcast on Ether0
AT: Recvd ZIP cmd 6 from 4160.19-6
AT: 3 query packets sent to neighbor 4160.19
AT: 1 zones for 31902, ZIP XReply, src 4160.19
AT: net 31902, zonelen 10, name US-Florida
```

The first line indicates that the router has received an RTMP update that includes a new network number and is now requesting zone information:

```
AT: Sent GetNetInfo request broadcast on Ether0
```

The second line indicates that the neighbor at address 4160.19 replies to the zone request with a default zone:

```
AT: Recvd ZIP cmd 6 from 4160.19-6
```

The third line indicates that the router responds with three queries to the neighbor at network address 4160.19 for other zones on the network:

```
AT: 3 query packets sent to neighbor 4160.19
```

The fourth line indicates that the neighbor at network address 4160.19 responds with a ZIP extended reply, indicating that one zone has been assigned to network 31902:

```
AT: 1 zones for 31902, ZIP XReply, src 4160.19
```

The fifth line indicates that the router responds that the zone name of network 31902 is US-Florida, and the zone length of that zone name is 10:

```
AT: net 31902, zonelen 10, name US-Florida
```

debug appn all

To turn on all possible debugging messages for Advanced Peer-to-Peer Networking (APPN), use the **debug appn all** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug appn all

no debug appn all



Note

Refer to the other forms of the **debug appn** command to enable specific debug output selectively.

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

This command shows all APPN events. Use other forms of the **debug appn** command to display specific types of events.



Caution

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



Caution

Debugging output takes priority over other network traffic. The **debug appn all** command generates more output than any other **debug appn** command and can alter timing in the network node. This command can severely diminish router performance or even render it unusable. In virtually all cases, it is best to use specific **debug appn** commands.

Examples

Refer to the documentation for specific **debug appn** commands for examples and explanations.

Related Commands

| Command | Description |
|--------------------------------|--|
| debug appn cs | Displays the APPN CS component activity. |
| debug appn ds | Displays debugging information on APPN DS component activity. |
| debug appn hpr | Displays information related to HPR code execution. |
| debug appn ms | Displays debugging information on APPN MS component activity. |
| debug appn nof | Displays information on APPN NOF component activity. |
| debug appn pc | Displays debugging information on APPN PC component activity. |
| debug appn ps | Displays debugging information on APPN PS component activity. |
| debug appn scm | Displays debugging information on APPN SCM component activity. |
| debug appn ss | Displays SS events. |
| debug appn trs | Displays debugging information on APPN TRS component activity. |

debug appn cs

To display APPN Configuration Services (CS) component activity, use the **debug appn cs** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug appn cs

no debug appn cs

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The CS component is responsible for defining link stations, ports, and connection networks. It is responsible for the activation and deactivation of ports and link stations and handles status queries for these resources.

Examples

The following is sample output from the **debug appn cs** command. In this example a link station is being stopped.

```
Router# debug appn cs

Turned on event 008000FF

Router# appn stop link PATTY

APPN: ----- CS ----- Deq STOP_LS message
APPN: ----- CS ----- FSM LS: 75 17 5 8
APPN: ----- CS ----- Sending DEACTIVATE_AS - station PATTY
APPN: ----- CS ----- deactivate_as_p->ips_header.lpid = A80A60
APPN: ----- CS ----- deactivate_as_p->ips_header.lpid = A80A60
APPN: ----- CS ----- Sending DESTROY_TG to PC - station PATTY - lpid=A80A60
APPN: ----- CS ----- Deq DESTROY_TG - station PATTY
APPN: ----- CS ----- FSM LS: 22 27 8 0
APPN: ----- CS ----- Sending TG update for LS PATTY to TRS
APPN: ----- CS ----- ENTERING XID_PROCESSING: 4
%APPN-6-APPNSENDMSG: Link Station PATTY stopped
```

[Table 9](#) describes the significant fields and messages shown in the display.

Table 9 *debug appn cs* Field Descriptions

| Field | Description |
|---------|--|
| APPN | APPN debugging output. |
| CS | CS component output. |
| Deq | CS received a message from another component. |
| FSM LS | Link station finite state machine is being referenced. |
| Sending | CS is sending a message to another component. |

■ debug appn cs

Related Commands

| Command | Description |
|--------------------------------|--|
| debug appn all | Turns on all possible debugging messages for APPN. |

debug appn ds

To display debugging information on APPN Directory Services (DS) component activity, use the **debug appn ds** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug appn ds

no debug appn ds

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The DS component manages searches for resources in the APPN network. DS is also responsible for registration of resources within the network.

Examples

The following is sample output from the **debug appn ds** command. In this example a search has been received.

```
Router# debug appn ds

Turned on event 080000FF
APPN: NEWDS: LS: search from: NETA.PATTY
APPN: NEWDS: pcid: DD3321E8B5667111
APPN: NEWDS: Invoking FSM NNSolu
APPN: NEWDS: LSfsm_NNSolu: 00A67AA0 pcid: DD3321E8B5667111 row: 0 col: 0 inp: 80200000
APPN: NEWDS: LSfsm_parent: 00A89940 row: 0 col: 0 inp: 80000000
APPN: NEWDS: Rcvd a LMRQ
APPN: NEWDS: LSfsm_NNSolu: 00A67AA0 pcid: DD3321E8B5667111 row: 12 col: 1 inp: 40000000
APPN: NEWDS: LSfsm_parent: 00A89940 row: 8 col: 1 inp: 40000000
APPN: NEWDS: LSfsm_child: 00A89BE8 row: 0 col: 0 inp: 80000080
APPN: NEWDS: PQenq REQUEST_ROUTE (RQ) to TRS
APPN: NEWDS: LSfsm_child: 00A8A1C0 row: 1 col: 0 inp: 80000008
APPN: NEWDS: LSfsm_NNSolu: 00A67AA0 pcid: DD3321E8B5667111 row: 5 col: 1 inp: 80C04000
APPN: NEWDS: LSfsm_child: 00A8A1C0 row: 7 col: 1 inp: 80844008
APPN: NEWDS: Rcvd a LMRQ
APPN: NEWDS: LSfsm_NNSolu: 00A67AA0 pcid: DD3321E8B5667111 row: 16 col: 6 inp: 40800000
APPN: NEWDS: LSfsm_child: 00A8A1C0 row: 14 col: 5 inp: 40800000
APPN: NEWDS: LSfsm_parent: 00A89940 row: 3 col: 1 inp: 80840000
APPN: NEWDS: send locate to node: NETA.PATTY
```

Table 10 describes the significant fields in the display.

Table 10 *debug appn ds Field Descriptions*

| Field | Description |
|-------------|---|
| APPN | APPN debugging output. |
| NEWDS | DS component output. |
| search from | Locate was received from NETA.PATTY. |
| LSfsm_ | Locate Search finite state machine is being referenced. |
| PQenq | Message was sent to another component. |

Table 10 debug appn ds Field Descriptions (continued)

| Field | Description |
|-------------|--|
| Rcvd | Message was received from another component. |
| send locate | Locate will be sent to NETA.PATY. |

Related Commands

| Command | Description |
|-----------------------|--|
| debug appn all | Turns on all possible debugging messages for APPN. |

debug appn hpr

To display debugging information related to High Performance Routing (HPR) code execution, use the **debug appn hpr** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug appn hpr

no debug appn hpr

Syntax Description

This command has no arguments or keywords.

Examples

The following is sample output from the **debug appn hpr** command:

```
Router# debug appn hpr

APPN: -- ncl.ncl_map_dlc_type() -- mapping TOKEN_RING(4) to NCL_TR(3)
APPN: -- ncl.ncl_port() -- called with port_type:3, cisco_idb:893A14, hpr_ssap:C8
APPN: -- ncl.process_port_change() -- port coming up
APPN: -- ncl.process_port_change() -- PORT_UP
APPN: -- ncl.ncl_port_fsm -- FSM Invoked: Input:0, State:0->1, Action:0
APPN: -- ncl.ncl_port_fsm -- FSM Invoked: Input:1, State:1->2, Action:1
APPN: -- ncl.ncl_unmap_dlc_type() -- mapping NCL(3) to CLS(3)
APPN: ----- ANR ----- Sending ACTIVATE_SAP.req
APPN: -- cswncsnd.main() -- received LSA_IPS ips.
APPN: -- ncl.ncl_port_fsm -- FSM Invoked: Input:3, State:2->3, Action:4
APPN: -- ncl.ncl_assign_anr() -- Assigned ANR,anr:8002
APPN: -- ncl.ncl_map_dlc_type() -- mapping TOKEN_RING(4) to NCL_TR(3)
APPN: -- ncl.ncl_populate_anr() -- anr:8002, dlc_type:3, idb 893A14
APPN: -- ncl.ncl_populate_anr() -- send anr_tbl_update to owning cswncsnd
APPN: -- ncl.ncl_ls_fsm -- FSM Invoked: Input:0, State:0->1, Action:0
APPN: ncl.ncl_send_reqopn_stn_req
APPN: -- ncl.ncl_unmap_dlc_type() -- mapping NCL(3) to CLS(3)
APPN: -- ncl.ncl_ls_fsm() -- send anr_tbl_update to owning cswncsnd
APPN: -- cswncsnd.main() -- received ANR_TBL_UPDATE ips.
APPN: -- cswncsnd.apply_anr_table_update() -- ANR:8002
APPN: -- cswncsnd.main() -- received ANR_TBL_UPDATE ips.
APPN: -- cswncsnd.apply_anr_table_update() -- ANR:8002
APPN: -- cswncsnd.main() -- received LSA_IPS ips.
APPN: -- ncl.ncl_ls_fsm -- FSM Invoked: Input:1, State:1->2, Action:1
APPN: -- ncl.ncl_ls_fsm -- P_CEP_ID:AAF638
APPN: -- ncl.ncl_ls_fsm() -- send anr_tbl_update to owning cswncsnd
APPN: -- cswncsnd.main() -- received ANR_TBL_UPDATE ips.
APPN: -- cswncsnd.apply_anr_table_update() -- ANR:8002
APPN: rtpm: rtp_send() sent data over connection B9D5E8
APPN: hpr timer: rtt start time clocked at 135952 ms
APPN: -- cswncsnd.main() -- received NCL_SND_MSG ips.
APPN: -- cswncsnd.process_nlp_from_rtp() -- label: 8002, send to p_cep 00AAF638.
APPN: hpr timer: rtt end time clocked at 135972 ms
APPN: hpr timer: round trip time measured at 20 ms
```

Table 11 describes the significant fields shown in the display.

Table 11 *debug appn hpr Field Descriptions*

| Field | Description |
|----------------------|--|
| APPN | APPN debugging output. |
| NCL | Network control layer debugging output. Network control layer is the component that handles ANR packets. |
| ncl_port_fsm | Network control layer port finite state machine has been invoked. |
| ncl_assign_anr | ANR label has been assigned to an activating link station. |
| ncl_populate_anr | System is updating the ANR record with information specific to the link station. |
| ncl_ls_fsm | Network control layer link finite state machine has been invoked. |
| rtp_send | RTP is about to send a packet. |
| hpr timer | Debugging output related to an HPR timer. |
| rtt start time | RTP is measuring the round-rip time for an HPR status request packet. This is the start time. |
| NCL_SND_MSG | Network control layer has been requested to send a packet. |
| process_nlp_from_rtp | Network control layer has been requested by RTP to send a packet. |
| rtt end time | RTP is measuring the round trip time for an HPR status request packet. This is the time. |
| round trip time | Round-trip time for this HPR status exchange has been computed. |

Related Commands

| Command | Description |
|-----------------------|--|
| debug appn all | Turns on all possible debugging messages for APPN. |

debug appn ms

To display debugging information on APPN Management Services (MS) component activity, use the **debug appn ms** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug appn ms

no debug appn ms

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The MS component is responsible for generating, sending, and forwarding network management information in the form of traps and alerts to a network management focal point, such as Netview, in the APPN network.

Examples

The following is sample output from the **debug appn ms** command. In this example an error occurred that caused an alert to be generated.

```
Router# debug appn ms

APPN: ----- MSS00 ----- Deq ALERT_MSU msg
APPN: --- MSP70 --- ALERT MV FROM APPN WITH VALID LGTH
APPN: --- MSCPL --- Find Active FP
APPN: --- MSP30 --- Entering Build MS Transport
APPN: --- MSP31 --- Entering Building Routing Info.
APPN: --- MSP34 --- Entering Build GDS
APPN: --- MSP32 --- Entering Building UOW correlator
APPN: --- MSP34 --- Entering Build GDS
APPN: --- MSP30 --- Building GDS 0x1310
APPN: --- MSP30 --- Building MS Transport
APPN: --- MSP72 --- ACTIVE FP NOT FOUND, SAVE ONLY
APPN: --- MSUTL --- UOW <= 60, ALL COPIED in extract_uow
APPN: --- MSCAT --- by enq_cached_ms QUEUE SIZE OF QUEUE after enq 4
```

[Table 12](#) describes the significant fields in the display.

Table 12 *debug appn ms Field Descriptions*

| Field | Description |
|-------|---|
| APPN | Indicates that this is APPN debugging output. |
| MSP | Indicates that this is MS component output. |

Related Commands

| Command | Description |
|--------------------------------|--|
| debug appn all | Turns on all possible debugging messages for APPN. |

debug appn nof

To display debugging information on APPN Node Operator Facility (NOF) component activity, use the **debug appn nof** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug appn nof

no debug appn nof

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The NOF component is responsible for processing commands entered by the user such as start, stop, show, and configuration commands. NOF forwards these commands to the proper component and waits for the response.

Examples

The following is sample output from the **debug appn nof** command. In this example, an APPN connection network is being defined.

```
Router# debug appn nof

Turned on event 010000FF

Router# config term

Enter configuration commands, one per line. End with CNTL/Z.
Router(config)# appn connection-network NETA.CISCO
Router(config-appn-cn)# port TR0
Router(config-appn-cn)# complete
router(config)#

APPN: ----- NOF ----- Define Connection Network Verb Received
APPN: ----- NOF ----- send define_cn_t ips to cs
APPN: ----- NOF ----- waiting for define_cn rsp from cs
router(config)#
```

[Table 13](#) describes the significant fields in the display.

Table 13 *debug appn nof Field Descriptions*

| Field | Description |
|----------|------------------------------------|
| APPN | APPN debugging output. |
| NOF | NOF component output. |
| Received | Configuration command was entered. |
| send | Message was sent to CS. |
| waiting | Response was expected from CS. |

Related Commands

| Command | Description |
|--------------------------------|--|
| debug appn all | Turns on all possible debugging messages for APPN. |

debug appn pc

To display debugging information on APPN Path Control (PC) component activity, use the **debug appn pc** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug appn pc

no debug appn pc

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The PC component is responsible for passing Message Units (MUs) between the Data Link Control (DLC) layer and other APPN components. PC implements transmission priority by passing higher priority MUs to the DLC before lower priority MUs.

Examples

The following is sample output from the **debug appn pc** command. In this example an MU is received from the network:

```
Router# debug appn pc

Turned on event 040000FF
APPN: ----- PC-----PC Deq REMOTE msg variant_name 2251
APPN: --PC-- mu received to PC lpid: A80AEC
APPN: --PC-- mu received from p_cep_id: 67C6F8
APPN: ----- PC-----PC Deq LSA_IPS from DLC
APPN: --PCX dequeued a DATA.IND
APPN: --- PC processing DL_DATA.ind
APPN: --PC-- mu_error_checker with no error, calling frr
APPN: --PC-- calling frr for packet received on LFSID: 1 2 3
APPN: ----- PC-----PC is sending MU to SC A90396
APPN: ----- SC-----send mu: A90396, rpc: 0, nws: 7, rh.b1: 90
APPN: SC: Send mu.snf: 8, th.b0: 2E, rh.b1: 90, dcf: 8
```

[Table 14](#) describes the significant fields in the display.

Table 14 *debug appn pc* Field Descriptions

| Field | Description |
|-------------|---|
| APPN | APPN debugging output. |
| PC | PC component output. |
| Deq REMOTE | Message was received from the network. |
| mu received | Message is an MU. |
| DATA.IND | MU contains data. |
| sending MU | MU is session traffic for an ISR session. The MU is forwarded to the Session Connector component for routing. |

Related Commands

| Command | Description |
|-----------------------|--|
| debug appn all | Turns on all possible debugging messages for APPN. |

debug appn ps

To display debugging information on APPN Presentation Services (PS) component activity, use the **debug appn ps** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug appn ps

no debug appn ps

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The PS component is responsible for managing the Transaction Programs (TPs) used by APPN. TPs are used for sending and receiving searches, receiving resource registration, and sending and receiving topology updates.

Examples

The following is sample output from the **debug appn ps** command. In this example a CP capabilities exchange is in progress.

```
Router# debug appn ps

Turned on event 200000FF
APPN: ---- CCA --- CP_CAPABILITIES_TP has started
APPN: ---- CCA --- About to wait for Partner to send CP_CAP
APPN: ---- CCA --- Partner LU name: NETA.PATTY
APPN: ---- CCA --- Mode Name: CPSVCMG
APPN: ---- CCA --- CGID: 78
APPN: ---- CCA --- About to send cp_cp_session_act to SS
APPN: ---- CCA --- Waiting for cp_cp_session_act_rsp from SS
APPN: ---- CCA --- Received cp_cp_session_act_rsp from SS
APPN: ---- CCA --- About to send CP_CAP to partner
APPN: ---- CCA --- Send to partner completed with rc=0, 0
APPN: ---- RCA --- Allocating conversation
APPN: ---- RCA --- Sending CP_CAPABILITIES
APPN: ---- RCA --- Getting conversation attributes
APPN: ---- RCA --- Waiting for partner to send CP_CAPABILITIES
APPN: ---- RCA --- Normal processing complete with cgid = 82
APPN: ---- RCA --- Deallocating CP_Capabilities conversation
```

[Table 15](#) describes the significant fields in the display.

Table 15 *debug appn ps* Field Descriptions

| Field | Description |
|-------|------------------------------------|
| APPN | APPN debugging output. |
| CCA | CP Capabilities TP output. |
| RCA | Receive CP Capabilities TP output. |

Related Commands

| Command | Description |
|--------------------------------|--|
| debug appn all | Turns on all possible debugging messages for APPN. |

debug appn scm

To display debugging information on APPN Session Connector Manager (SCM) component activity, use the **debug appn scm** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug appn scm

no debug appn scm

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The SCM component is responsible for the activation and deactivation of the local resources that route an intermediate session through the router.

Examples

The following is sample output from the **debug appn scm** command. In this example an intermediate session traffic is being routed.

```
Router# debug appn scm

Turned on event 020000FF
Router#
APPN: ----- SCM-----SCM Deq a MU
APPN: ----- SCM-----SCM send ISR_INIT to SSI
APPN: ----- SCM----- (i05) Enter compare_fqpcid()
APPN: ----- SCM-----Adding new session_info table entry. addr=A93160
APPN: ----- SCM-----SCM Deq ISR_CINIT message
APPN: ----- SCM----- (i05) Enter compare_fqpcid()
APPN: ----- SCM-----SCM sends ASSIGN_LFSID to ASM
APPN: ----- SCM-----SCM Rcvd sync ASSIGN_LFSID from ASM
APPN: ----- SCM-----SCM PQenq a MU to ASM
APPN: ----- SCM-----SCM Deq a MU
APPN: ----- SCM----- (i05) Enter compare_fqpcid()
APPN: ----- SCM-----SCM PQenq BIND rsp to ASM
```

[Table 16](#) describes the significant fields in the display.

Table 16 *debug appn scm Field Descriptions*

| Field | Description |
|-------|------------------------|
| APPN | APPN debugging output. |
| SCM | SCM component output. |

Related Commands

| Command | Description |
|-----------------------|--|
| debug appn all | Turns on all possible debugging messages for APPN. |

debug appn ss

To display session services (SS) events, use the **debug appn ss** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug appn ss

no debug appn ss

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The SS component generates unique session identifiers, activates and deactivates control point-to-control point (CP-CP) sessions, and assists LUs in initiating and activating LU-LU sessions.

Examples

The following is sample output from the **debug appn ss** command. In this example CP-CP sessions between the router and another node are being activated.

```
Router# debug appn ss

Turned on event 10000FF
APPN: ----- SS ----- Deq ADJACENT_CP_CONTACTED message
APPN: ----- SS ----- Deq SESSST_SIGNAL message
APPN: ----- SS ----- Deq CP_CP_SESSION_ACT message
APPN: Sending ADJACENT_NN_1015 to SCM, adj_node_p=A6B980,cp_name=NETA.PATTY
APPN: ----- SS ----- Sending REQUEST_LAST_FRSN message to TRS
APPN: ----- SS ----- Receiving REQUEST_LAST_FRSN_RSP from TRS
APPN: ----- SS ----- Sending ACTIVE_CP_STATUS_CONLOSER message to DS
APPN: ----- SS ----- Sending ACTIVE_CP_STATUS_CONLOSER message to MS
APPN: ----- SS ----- Sending ACTIVE_CP_STATUS_CONLOSER message to TRS
APPN: ----- SS ----- Sending CP_CP_SESSION_ACT_RSP message to CCA TP
APPN: ----- SS ----- Sending PENDING_ACTIVE_CP_STATUS_CONWINNER message to DS
APPN: ----- SS ----- Sending REQUEST_LAST_FRSN message to TRS
APPN: ----- SS ----- Receiving REQUEST_LAST_FRSN_RSP from TRS
APPN: ----- SS ----- Sending ACT_CP_CP_SESSION message to RCA TP
APPN: ----- SS ----- Deq ASSIGN_PCID message
APPN: ----- SS ----- Sending ASSIGN_PCID_RSP message to someone
APPN: ----- SS ----- Deq INIT_SIGNAL message
APPN: ----- SS ----- Sending REQUEST_COS_TPF_VECTOR message to TRS
APPN: ----- SS ----- Receiving an REQUEST_COS_TPF_VECTOR_RSP from TRS
APPN: ----- SS ----- Sending REQUEST_SINGLE_HOP_ROUTE message to TRS
APPN: ----- SS ----- Receiving an REQUEST_SINGLE_HOP_ROUTE_RSP from TRS
APPN: ----- SS ----- Sending ACTIVATE_ROUTE message to CS
APPN: ----- SS ----- Deq ACTIVATE_ROUTE_RSP message
APPN: ----- SS ----- Sending CINIT_SIGNAL message to SM
APPN: ----- SS ----- Deq ACT_CP_CP_SESSION_RSP message
APPN: -- SS----SS ssp00, act_cp_cp_session_rsp received, sense_code=0, cgid=5C,
ips@=A93790
APPN: Sending ADJACENT_NN_1015 to SCM, adj_node_p=A6B980,cp_name=18s
APPN: ----- SS ----- Sending ACTIVE_CP_STATUS_CONWINNER message to DS
APPN: ----- SS ----- Sending ACTIVE_CP_STATUS_CONWINNER message to MS
APPN: ----- SS ----- Sending ACTIVE_CP_STATUS_CONWINNER message to TRS
```

Table 17 describes the significant fields in the display.

Table 17 *debug appn ss Field Descriptions*

| Field | Description |
|-------|------------------------|
| APPN | APPN debugging output. |
| SS | SS component output. |

Related Commands

| Command | Description |
|-----------------------|--|
| debug appn all | Turns on all possible debugging messages for APPN. |

debug appn trs

To display debugging information on APPN Topology and Routing Services (TRS) component activity, use the **debug appn trs** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

debug appn trs

no debug appn trs

Syntax Description

This command has no arguments or keywords.

Usage Guidelines

The TRS component is responsible for creating and maintaining the topology database, creating and maintaining the class of service database, and computing and caching optimal routes through the network.

Examples

The following is sample output from the **debug appn trs** command:

```
Router# debug appn trs

Turned on event 40000FF
APPN: ----- TRS ----- Received a QUERY_CPNAME
APPN: ----- TRS ----- Received a REQUEST_ROUTE
APPN: ----- TRS ----- check_node node_name=NETA.LISA
APPN: ----- TRS ----- check_node node_index=0
APPN: ----- TRS ----- check_node node_weight=60
APPN: ----- TRS ----- add index 484 to origin description list
APPN: ----- TRS ----- add index 0 to dest description list
APPN: ----- TRS ----- origin tg_vector is NULL
APPN: ----- TRS ----- weight_to_origin = 0
APPN: ----- TRS ----- weight_to_dest = 0
APPN: ----- TRS ----- u_b_s_f weight = 30
APPN: ----- TRS ----- u_b_s_f prev_weight = 2147483647
APPN: ----- TRS ----- u_b_s_f origin_index = 484
APPN: ----- TRS ----- u_b_s_f dest_index = 0
APPN: ----- TRS ----- b_r_s_f weight = 30
APPN: ----- TRS ----- b_r_s_f origin_index = 484
APPN: ----- TRS ----- b_r_s_f dest_index = 0
APPN: ----- TRS ----- Received a REQUEST_ROUTE
APPN: ----- TRS ----- check_node node_name=NETA.LISA
APPN: ----- TRS ----- check_node node_index=0
APPN: ----- TRS ----- check_node node_weight=60
APPN: ----- TRS ----- check_node node_name=NETA.BART
APPN: ----- TRS ----- check_node node_index=484
APPN: ----- TRS ----- check_node node_weight=60
APPN: ----- TRS ----- add index 484 to origin description list
APPN: ----- TRS ----- add index 0 to dest description list
APPN: ----- TRS ----- origin_tg_weight to non-VN=30
APPN: ----- TRS ----- origin_node_weight to non-VN=60
APPN: ----- TRS ----- weight_to_origin = 90
APPN: ----- TRS ----- weight_to_dest = 0
APPN: ----- TRS ----- u_b_s_f weight = 120
APPN: ----- TRS ----- u_b_s_f prev_weight = 2147483647
APPN: ----- TRS ----- u_b_s_f origin_index = 484
APPN: ----- TRS ----- u_b_s_f dest_index = 0
```

```
APPN: ----- TRS ----- b_r_s_f weight = 120
APPN: ----- TRS ----- b_r_s_f origin_index = 484
APPN: ----- TRS ----- b_r_s_f dest_index = 0
```

Table 18 describes the significant fields in the display.

Table 18 *debug appn trs Field Descriptions*

| Field | Description |
|-------|------------------------|
| APPN | APPN debugging output. |
| TRS | TRS component output. |