

Document Name		USR8550 – Console commands	
Document Type		Documentation	
Product or Technology related		ADSL	
Document Number	EMEA CSO 0013	Document Date	23/04/02

USR8550 – Console commands

Document Name.....	1	dhcpserver version.....	11
Document Number.....	1	Event Console Commands.....	11
Document Date	1	event.....	11
General Console Commands.....	3	ISFS/Flashfs Console Commands	11
Home	3	cat	11
restart.....	3	fsck.....	12
uptime.....	3	help	12
version	3	id.....	12
Bridge Console Commands.....	3	info	12
device add	3	ls	13
device delete.....	4	rewrite.....	13
device list	4	rm	13
ethertype.....	4	trace	14
filter	4	update.....	14
filterage.....	5	version	14
flush.....	5	IP Console Commands.....	14
info	5	Summary.....	14
interface	5	abort.....	16
portfilter	6	arp.....	16
spanning.....	6	arprounting.....	17
status.....	6	autoloop	17
version	6	config.....	17
Config Console Commands.....	7	device	18
4.1 list.....	7	disable.....	19
4.2 print	7	enable	20
4.3 reset.....	7	errors	20
4.4 save	7	etherfiles	20
4.5 resource.....	7	files.....	20
DHCP Client Console Commands.....	7	get.....	21
dhcpclient config.....	7	help	21
dhcpclient help	7	ipatm abort	22
dhcpclient pool	8	ipatm arp.....	22
dhcpclient status.....	8	ipatm arpserver	22
dhcpclient trace	8	ipatm files.....	22
DHCP-related IP process commands	9	ipatm help.....	23
DHCP Server Console Commands	9	ipatm lifetime	23
dhcpserver config.....	9	ipatm pvc.....	23
dhcpserver help	10	iphostname	24
dhcpserver pool	10	noerrors.....	24
dhcpserver status	10	norelay	24
dhcpserver trace.....	10	ping	25

portname	25	<channel> info	47
protocols	25	<channel> interface	47
relay	26	<channel> lcpmaxconfigure	47
restart	26	<channel> lcpmaxfailure	47
rip accept	26	<channel> lcpmaxterminate	48
rip allowed	27	<channel> llc	48
rip boot	27	<channel> pvc	48
rip help	27	<channel> qos	49
rip hostroutes	28	<channel> remoteip	49
rip killrelay	28	<channel> svc	49
rip poison	28	<channel> theylogin	50
rip relay	28	<channel> tunnel <n> <tunnel protocol>	
rip rxstatus	29	<dial direction>	50
rip send	29	<channel> welogin	50
rip trigger	29	bcp	50
route	29	interface <n> localip	51
routeflush	30	interface <n> stats	51
routes	31	user	51
snmp	31	version	51
stats	31	Spanning Tree Console Commands	52
subnet	31	disable	52
trace	32	enable	52
untrace	33	forwarddelay	52
uptime	33	hellotime	53
version	33	info	53
NAT Console Commands	33	maxage	53
ip nat	33	port <number>	53
nat event	34	port <number> disable	53
nat interfaces	34	port <number> enable	54
nat inbound	34	port <number> pathcost	54
nat stats	36	port <number> priority	54
nat version	36	priority	54
nat dump	36	status	55
nat fragments	37	version	55
nat hashtable	37	SNMP Console Commands	55
OAM Console Commands	37	access	55
PPP Console commands	43	config	56
Console object types	43	help	56
Console examples	44	trap	56
Console commands	45	TFTP Console Commands	57
<channel> clear	45	connect	57
<channel> disable	45	get	57
<channel> discard	45	init	58
<channel> echo	45	list	58
<channel> echo every	46	put	58
<channel> enable	46	trace	58
<channel> event	46	version	58
<channel> hdlc	46		

The commands list are as below:

The shortage of list are in red as follows.

ap	atm	bridge	bsp	buffer
bun	chips	config	dhcpclient	dhcpserver
edd	ethernet	event	flashfs	ip
isfs	nat	oamloop	portcli	ppp

restart snmp tftp uptime version

General Console Commands

Home

Syntax:

home

Description:

Return to root directory of command system.

restart

Syntax:

restart

Description:

Reboots the system.

uptime

Syntax:

Uptime

Description:

Displays the time for which the system has been up.

version

Syntax:

version

Description:

Displays the system type and version.

Bridge Console Commands

Console commands should be prefixed with `bridge` in order to direct them to the **bridge** process.

device add

Syntax:

device add <device>

Description:

This command adds a device to the bridge configuration. Attempts to add the bridge itself or an existing device to the bridge are rejected. Attempts to add devices which don't support the Cyan interface are rejected. There is a limit on the number of devices that can be attached to the bridge. If a device is successfully added to the bridge, it will only become active after the configuration is saved and the system is rebooted. If the device being added is from a process which supports multiple devices, the /DEVICE attribute must be specified as part of the device name. The table below shows devices which may be attached to the bridge, although not all systems may support all devices.

lec1 Forum LAN emulation

edd Ethernet driver

r1483 RFC1483 protocol (PVC)

ppp Point-to-Point protocol

Configuration saving saves this information. See the section implementation constraints for details of which devices are added by default.

Example:

```
device add edd
device add ppp/DEVICE=2
```

See also:

device delete, device list

device delete

Syntax:

```
device delete <device>
```

Description:

This command deletes a device from the bridge configuration. The changes will only take place after the configuration is saved and the system is rebooted. The syntax of the device name is the same as that for the `device add` command. Configuration saving saves this information.

Example:

```
device delete r1483
```

See also:

device add, device list

device list

Syntax:

```
device list
```

Description:

This command lists all the devices that are currently attached to the bridge. It does not show the stored configuration (which can be seen with the `config print` command).

Example:

```
device list
```

See also:

device add, device delete

ethertype

Syntax:

```
ethertype [<port> any|ip|pppoe]
```

Description:

This command enables filtering of Ethernet packets according to the `ETHER_TYPE` field in the header. Only packets of the type specified using this command will be **sent** on the port specified; packets of all types will always be **received**. By default, all bridge ports are set to "any", which means that the type of the packet will never be checked. The meaning of the other options is as follows:

Option Permitted ETHER_TYPE values

"ip" 0x0800 – IP

 0x0806 – ARP

"pppoe" 0x8863, 0x8864 – PPP Over Ethernet (RFC 2516)

The port is specified as an integer, as displayed by the `device list` command. When using this command in the `initbridge` configuration file, ports are numbered in the order in which the `device add` commands are given, starting from 1.

If no arguments are given, the current settings for each port are displayed.

Example:

```
ethertype 2 any
```

See also:

filter

Syntax:

```
filter
```

Description:

This command shows the current contents of the bridge's filter table. The MAC entries for each device are shown in turn together with the time that the MAC address was last seen by the bridge. The command also shows the current filter ageing time, in seconds, and the number of creation failures since the system was started. Creation failures occur when there is no room left in the filter table for a new entry.

Example:

```
filter
```

See also:

```
Filterage
```

filterage**Syntax:**

```
filterage [<age>]
```

Description:

This command sets, or displays if no arguments are given, the filter table ageing time. The ageing time is the time after which MAC addresses are removed from the filter table when there has been no activity. The time is specified in seconds and may be any integer value in the range 10...100,000 seconds. This value may also be changed through SNMP. Changing the value of filterage has immediate effect.

Configuration saving saves this information. By default the filter ageing time is set to 300 seconds.

Example:

```
filterage
```

See also:

```
Filter
```

flush**Syntax:**

```
flush [<port>]
```

Description:

This command allows the MAC entries for a specified port, or all ports, to be removed from the filter table. The port number for a device may be determined using the `device list` or `status` commands. If the port number is omitted, all entries for all ports are removed from the filter table.

Example:

```
flush
```

See also:

```
filter, device list, status
```

info**Syntax:**

```
info
```

Description:

This command displays build information about the **bridge** process. The `version` command is a synonym.

Example:

```
info
```

See also:

```
Version
```

interface**Syntax:**

```
interface [sub-command].
```

Description:

This command accesses the ethernet support library sub-commands for the bridge itself, not for the devices which are attached to it. The ethernet support commands are documented in the "ATMOS Ethernet Support Library" specification.

Example:

```
interface stats
```

See also:

portfilter

Syntax:

```
portfilter [<source port> all|<destination ports>]
```

Description:

The `portfilter` command allows control over the bridge's forwarding and broadcasting behaviour. By default, when a multicast or an unknown packet is received on a port (referred to above as the source port), it will be forwarded to all other bridge ports (referred to above as the destination ports).

Each bridge port may have its behaviour modified separately. The first example below configures the bridge so that packets arriving on port 2 will only be forwarded to ports 3, 4 and 5, and packets arriving on port 3 will only be forwarded to port 1. All other ports retain their default behaviour.

Note that this command does not force packets arriving on the source port to be sent to all specified destination ports. The bridge retains its learning behaviour, so unicast packets, once their destination is known to the bridge, will still only be sent to one port. Note also that the bridge itself (for example when attached to the IP router) will always forward to all ports, and will always be forwarded to by all ports. The default behaviour can be restored by calling this command with the argument "all", as shown in the second example.

The ports are specified as integers, as displayed by the `device list` command. When using this command in the `initbridge` configuration file, ports are numbered in the order in which the `device add` commands are given, starting from 1.

If no arguments are given, the current settings for each port are displayed.

Example 1:

```
portfilter 2 3 4 5
portfilter 3 1
```

Example 2:

```
portfilter 2 all
portfilter 3 all
```

See also:

spanning

Syntax:

```
spanning [sub-command]
```

Description:

The spanning tree commands are only available if it has been compiled in to the bridge. The spanning tree commands are specified in this document later part.

status

Syntax:

```
status
```

Description:

This command shows the status of the bridge and its ports. The status information for a port includes the SNMP type information about time exceeded packets, packets discarded, etc. It also includes the broadcast history of the port over the last five seconds and the high water mark of packets queued on the bridge for this device.

Example:

```
status
```

See also:

version

Syntax:

```
version
```

Description:

This command displays build information about the **bridge** process. The `info` command is a synonym.

Example:

```
version
```

See also:

info

Config Console Commands

4.1 list

4.2 print

4.3 reset

4.4 save

4.5 resource

Syntax:

```
config esource <number>
```

Description:

Read or set the configured settings for resources used by UNI Signalling. This call is notprefixed with q93b, and only takes effect when configuration is saved and the system restarted. The size in bytes of each configured resource is shown in the example output, in the 'unit size' column. See the section on 'Implementation Constraints' for full details.

Example:

```
00:20:2b:00:04:03> config resource q93b
process resource number unit size space occupied
-----
q93b workspace 64000 1 64000
q93b localcalls 32 164 5248
q93b throughcalls 0 136 0
00:20:2b:00:04:03>
```

DHCP Client Console Commands

This section describes console commands provided by the `dhcpclient` process.

dhcpclient config

Syntax:

```
dhcpclient config
```

Description:

This command displays the current configuration of the DHCP client, including selected DHCP options.

Example:

```
mymachine> dhcpclient config
---
DHCP client configuration file: `//isfs/dhclient.conf'
timeout 60;
retry 60;
reboot 10;
backoff-cutoff 40;
interface "ethernet" {
send dhcp-lease-time 5000;
send dhcp-client-identifier "Galapagos";
}
```

dhcpclient help

Syntax:

```
dhcpclient help <command|all>
```

Description:

This command provides help on the various console commands provided by the ATMOS DHCP client. Specifying the command name gives detailed help, and specifying the argument "all"

gives detailed help on all commands.

Example:

```
mymachine> dhcpclient help
Help is available on the following commands:
config help pool status trace untrace
```

dhcpclient pool

Syntax:

```
dhcpclient pool [verbose]
```

Description:

This command displays the state of the memory pool being used by the DHCP client. Should the client ever run out of memory, use of this command is helpful in determining the optimum memory pool size for the client. For example, supporting DHCP client functionality on several interfaces simultaneously will require proportionately more memory. The default pool size specified in the system file `dhcpclient` is 40000 bytes.

The verbose option lists all allocated and freed memory chunks.

Example:

```
mymachine> dhcpclient pool
DHCP Client Memory Pool Status
total pool size 39968
free 21392
allocated 18576
mean alloc chunk 67
max free chunk 13904
```

dhcpclient status

Syntax:

```
dhcpclient status [all]
```

Description:

This command provides DHCP status information for the active bound lease associated with each valid interface in turn, including IP address, time until lease renewal, subnet mask and DHCP server address. Including the "all" flag shows, for each valid interface, the active lease, leases which are being, or have been offered to the interface, and any leases which are still being held by the client which are not currently active (since a single interface can only have one active lease at a time).

Example:

```
bd2000> dhcpclient status
DHCP Client Lease Status (active lease only)
Interface 'ethernet'
  Status | Server ID          | IP address          | Subnet mask          | Renewal
-----+-----+-----+-----+-----
*ACTIVE* | 192.168.219.151    | 192.168.219.1      | 255.255.255.0        | 31 seconds
```

dhcpclient trace

Syntax:

```
dhcpclient trace <trace option>
```

Description:

This command enables or disables tracing for the DHCP client. If no arguments are given the command lists the current tracing options enabled.

The following trace options are available:

- lease** Report changes in lease status (any interface)
- bootp** Report any bootp interoperation
- error** Report all errors (fatal events)
- warn** Report "warn" level events (important events)
- note** Report "note" level events (minor/frequent events)
- all** All trace options

Tracing options are disabled by using the "untrace" command with the option names to be disabled.

Saving configuration does not preserve the current tracing options that are enabled. By default tracing of **error**, **warn** and **note** are enabled.

Example:

```
mymachine> dhcpcclient trace
No tracing options currently enabled.
mymachine> dhcpcclient trace error warn note
Currently tracing: error warn note
```

DHCP-related IP process commands

The following commands are not provided by the DHCP client process but by the IP process `ip` (see [2]).

ip device

Syntax:

```
ip device add <i/f> <type> <file> [mtu <size>] [<IP address>|dhcp]
ip device
```

Description:

The **ip device add** command adds an interface to the configuration of the IP stack. The last parameter of the command would normally the IP address of the interface; use of the string **dhcp** causes the IP address to be discovered by the DHCP client software. Note that using the flag **dhcp** on an interface precludes running a DHCP server on that interface!

The **ip device** command lists the current configuration of any devices attached to the IP stack. A

device configured to use DHCP will show "dhcp" in the "IP address" column, followed by the actual IP address discovered and bound by DHCP, if any.

For interfaces configured to use DHCP, saving configuration only marks the interface as using DHCP; it does not save the actual IP address discovered by DHCP, which must be renewed.

A useful method of automatically configuring suitable IP devices is to put a "device add" statement into the file "`//isfs/resolve`" and downloading it upon booting the image.

Example:

```
bd2000> ip device add ethernet ether //edd dhcp
..DHCP then discovers the IP address for the interface..
bd2000> ip device
# type dev file IP address
device ethernet ether //edd mtu 1500 dhcp
```

DHCP Server Console Commands

This section describes console commands provided by the `dhcpserver` process.

dhcpserver config

Syntax:

```
dhcpserver config [add <text>|confirm|delete|flush]
```

Description:

This command displays or edits the current configuration of the DHCP server. To display current configuration, provide no arguments to the command. Use of the "add" argument adds the line `<text>` to the configuration file. Use of the "confirm" argument reparses the configuration file, confirming the changes made if the parse is successful. Use of the "delete" argument deletes the last line from the configuration file. Use of the "flush" argument deletes the whole configuration.

Following any change to the configuration file, it is necessary to "confirm" the changes, issue a "flashfs update" to commit the change to FLASH, and then restart the system before the changes can take effect.

Example:

```
bd2000> dhcpserver config
---
Current DHCP server configuration
---
allow unknown-clients;
allow bootp;
subnet 192.168.219.0 netmask 255.255.255.0 {
range 192.168.219.10 192.168.219.30;
max-lease-time 5000;
}
bd2000> dhcpserver config flush
Configuration file flushed.
```

```
bd2000> dhcpserver config
---
Current DHCP server configuration
(Issue "dhcpserver config confirm" followed by "flashfs update"
to confirm new configuration)
---
bd2000>
```

dhcpserver help

Syntax:

```
dhcpserver help <command|all>
```

Description:

This command provides help on the various console commands provided by the ATMOS DHCP server. Specifying a command name gives detailed help on the command. Specifying "all" gives

detailed help on all available commands.

Example:

```
bd2000> dhcpserver help
Help is available on the following commands:
config help pool status trace untrace
```

dhcpserver pool

Syntax:

```
dhcpserver pool [verbose]
```

Description:

This command gives a summary of DHCP server memory usage. The verbose option shows the entire memory allocation/free list.

Example:

```
bd2000> dhcpserver pool
DHCP Server Memory Pool Status
total pool size 79968
free 52448
allocated 27520
mean alloc chunk 59
max free chunk 30416
```

dhcpserver status

Syntax:

```
dhcpserver status
```

Description:

This command provides a summary of all leases known to the server on each interface in turn. It

also shows remaining available IP addresses (i.e. those with no specified lease time, or client identifier).

Example:

```
bd2000> dhcpserver status
DHCP Server Lease Status
Interface "ethernet"
IP address      | Client UID                | Expiry
-----|-----|-----
192.168.219.1   | 01:00:20:af:20:6f:59      | 11 hours
192.168.219.2   | 01:00:20:af:11:2a:ac      | 8 hours
192.168.219.3   | Myclient                  | 140 seconds
192.168.219.4   | 00:20:af:20:00:2b         | 2 days
192.168.219.5   | <unknown>                 | Never
192.168.219.6   | <unknown>                 | Never
192.168.219.7   | <unknown>                 | Never
192.168.219.8   | <unknown>                 | Expired
192.168.219.9   | <unknown>                 | Expired
192.168.219.10 | Foobarbozzle             | Expired
```

dhcpserver trace

Syntax:

```
dhcpserver trace <trace option>
```

Description:

This command enables or disables tracing for the DHCP server. If no arguments are given the command lists the current tracing options enabled.

The following trace options are available:

lease Report changes in lease status (any device)
bootp Report any BOOTP interoperation/emulation
error Report all errors (fatal events)
warn Report all warnings
note Report all note-level (minor) events
all All trace options

Tracing options are disabled by using the `untrace` command in the same way.

Saving configuration does not preserve the current tracing options that are enabled. By default,

only tracing of **error** is enabled.

Example:

```
bd2000> dhcpserver trace
No tracing options currently enabled.
bd2000> dhcpserver trace error warn note
Currently tracing: error warn note
```

dhcpserver version**Syntax:**

```
dhcpserver version
```

Description:

This command displays the current version number of the ATMOS DHCP software.

Example:

```
bd2000> dhcpserver version
ATMOS DHCP Version 1.02
bd2000>
```

Event Console Commands

event**Syntax:**

```
event help|next|previous|recent|show|unshow
```

Description:

Control tracing level at the console. The default setting is 1 (least verbose). If no argument is supplied the current setting is displayed. Other settings allow more output to appear:

- 1 Only solicited messages or fatal errors
 - 2 Major protocol or resource errors reported
 - 3 Unusual (but non-fatal) protocol events reported; port up/down events reported
 - 4 Brief messages on all signalling interactions and state changes
 - 5 Full decode of all arriving and leaving signalling messages
- The format command gives more flexibility over the decoding at event level 5.

See also:

```
format
port <p> event
```

ISFS/Flashfs Console Commands

The two processes provide a number of identical console commands. Some of the commands are specific to one process and in the following, the scope of each command is indicated.

cat**Syntax:**

```
cat <file>
```

Scope:

ISFS and FLASHFS.

Description:

The `cat` command allows a console user to view the contents of the specified file. Only printable characters are displayed, non-printable characters are represented by a `.'` character. Printable characters include all standard printable characters together with carriage return, line feed, and tab.

No output formatting is performed, and no scroll lock function implemented.

Example:

```
cat ipaddresses
```

See also:

Ls

fsck

Syntax:

```
fsck
```

Scope:

FLASHFS only.

Description:

The `fsck` command causes FLASHFS to perform a complete file system integrity check on demand.

If any part of the test fails, appropriate console output will be displayed. A flag is maintained by FLASHFS to indicate if FLASH contents are good or bad and this is set or cleared depending on the result of the test. This test is automatically carried out during process initialisation and also after each update, without foreground output (except in the event of an error).

Example:

```
Fsck
```

help

Syntax:

```
help
```

Scope:

ISFS and FLASHFS.

Description:

The `help` command displays the help text which lists the commands supported by the process. The maximum trace levels are displayed in the relevant part of the help text.

Example:

```
help
```

See also:

Version

id

Syntax:

```
id <x>
```

Scope:

FLASHFS only.

Description:

The `id` command displays the identity of the specified FLASH device. The identity value is a 16bit number (displayed in Hex) that allows the programming software to identify the type(s) of device(s) in the board. If the device ID is matched with a supported type (and it should be if the user expects to program the FLASH!) the vendor and part number will be displayed as well. Devices are numbered from 0, and typically a system will contain 1 or 2 devices.

Example:

```
id 0
```

info

Syntax:

```
info
```

Scope:

FLASHFS only.

Description:

The 'info' command displays information about the current FLASHFS configuration. The start and end address of the current FLASHFS filesystem is printed, followed by the total filesystem size. The addresses are FLASHFS logical addresses, where address 0 is the start of the first flash device.

Additionally, the number of FLASH devices that FLASHFS was built for is displayed..

Example:

info

See also:

Version

ls

Syntax:

ls

Scope:

ISFS and FLASHFS, FLASHFS supports an '-l' option.

Description:

The `ls` command allows a console user to list the files present in the filesystem. The FLASHFS '-l' option displays more detailed information (logical address within FLASH and linked list information).

Example:

ls

See also:

Cat

rewrite

Syntax:

rewrite <file>

Scope:

FLASHFS

Description:

The `rewrite` command allows the boot area of the flash chip to be rewritten with the specified ISFS file. Note that the file must be present in ISFS, not in FLASHFS. The only checks that are applied to the file are that it is greater than 8k bytes in length, it doesn't collide with the start of the flash filing system and that it fits in the first flash chip. The format of the file depends upon the target system but the file produced by the "mkflash" utility is generally suitable. All the data from the specified file are written starting at the beginning of the first flash chip.

WARNING

Support for this command is conditionally compiled; it is not intended that it be present in standard builds and is primarily intended for use during development. This command is potentially dangerous as writing the wrong image to the boot area could result in an unbootable system.

Example:

rewrite boot

See also:

rm

Syntax:

rm <file>

Scope:

ISFS only.

Description:

The `rm` command allows the user to remove a file from the ISFS file system. The memory used to store the file is freed . A subsequent FLASHFS update will write the new, shorter, ISFS files into FLASHFS, providing an implicit `rm` function for FLASHFS.

Note: If the file removed is the only file that would be stored in FLASHFS as type 'fixed', the file will remain in FLASHFS as the fixed file area will not be re-written during an update.

Example:

`rm foo`

See also:

`Ls`

trace

Syntax:

`trace [x]`

Scope:

ISFS and FLASHFS, although the maximum trace level is different for each process.

Description:

The `trace` command either displays or sets the current trace level. With no parameters, the `trace` command displays the current trace level. Trace levels vary between 0 and the maximum level, as displayed by the `help` command, for each process.

Example:

`trace 2`

See also:

`Help`

update

Syntax:

`update`

Scope:

FLASHFS only.

Description:

The 'update' command instructs FLASHFS to update the FLASH memory from the files contained in the ISFS file system.

Example:

`update`

See also:

version

Syntax:

`version`

Scope:

ISFS and FLASHFS.

Description:

The 'version' command displays software version information about the process. The version number, which is displayed in the form 'a.bc', is defined in the module file as an integer 'abc'.

Example:

`version`

See also:

`Help`

IP Console Commands

Summary

The table below shows the commands that can be issued to the IP process in TELL messages or on the console to its *stdin* stream (after typing "@ip", for example). It shows which are mentioned in the "ip help" output, and which set some configuration that is saved in flash memory.

Shown by

"help"

Saved in

configuration

abort

arp + -arprouting

```

-autoloop
•
config •
device ••
disable •
enable • -errors
etherfiles
files
flush
get -help
•
ipatm ••
iphostname •
noerrors
norelay + •
Shown by
"help"
Saved in
configuration
ping •
portname •
protocols
relay + •
restart •
rip + •
route ••
routeflush •
routes •
snmp + •
stats •
subnet ••
trace -
untrace -
uptime •
version •
? •
• = Yes

```

+ = Shown in "ip help" output only if HADRONSWITCH is not defined
- = May be inserted explicitly in //isfs/resolve (e.g. for debugging purposes), but not saved by "ip config save"

Three obsolescent commands are not shown in the table above nor in the fuller descriptions below; they are supported only for consistency with older versions of the software:
devices (Lists devices; equivalent to "device" with no parameters)
subnets (Lists subnets; equivalent to "subnet" with no parameters)
dynroutes (Outputs a message saying that the command is no longer supported.)

The "hidden" commands that are not shown in the "ip help" output are generally either commands that are useful for debugging rather than for use by the end-user or commands of limited utility that are supported mainly for consistency with earlier versions of the software; it may be unwise to rely on their working in the same way in later versions of the software. Those that are hidden only when HADRONSWITCH is defined are commands that are of little or no use on a VIRATAswitch, where IP will normally be configured with a single IP-over-ATM interface.

There are some obsolescent features that are supported only for commands presented to standard input (at the "ip>" prompt), not for commands in TELL messages (such as commands at the "mymachine ip>" prompt). These are the processing of multiple commands on a line, separated by ";" ; comments, starting with "#"; definition of macros with the syntax

“var=value”, used as “\$var” within commands; and the “env” command to list macros. These features are not discussed further in this document.

abort

Syntax:

```
abort <assoc>
```

Description:

Aborts an IP association; <assoc> is the number of the association as shown by the “files” command. Currently (ATMOS IP version 1.29) this seems to be unreliable on UDP associations and can cause a crash (possibly because of lax error-handling by the application that opened the file); it is reliable on TCP associations. The “abort” command is “hidden”, not shown by “ip help”; it is probably useful, if at all, for debugging and troubleshooting.

Example:

```
mymachine> ip abort 3
```

See also:

Files

arp

Syntax:

```
arp add <i/f> <IP address> <MAC address>
arp delete <i/f> <IP address>
arp flush
arp [list]
arp help [all|<cmd>]
```

Description:

Allows display and manipulation of the ARP table: the list of IP addresses and corresponding MAC addresses obtained by ARP (see 4.3.1) on Ethernet-like interfaces. Normally there is no need to add entries to the table with “arp add”, since they should be discovered by the ARP protocol. Displaying the table with “arp list” (or just “arp”) is sometimes useful, and deleting an entry with “arp delete”, or the whole table with “arp flush”, can sometimes speed up recovery from temporary problems if something unusual has happened. Entries added with “arp add” do not time out like those discovered by use of the ARP protocol, but they are deleted by “arp flush” and will not survive a restart (they are not saved by configuration saving). Note that the ARP table is used only for destinations on directly connected Ethernet-like networks, not for those reached through routers (although the ARP table may be used to discover the MAC address of the router).

Example:

```
mymachine> ip arp add ether 192.168.50.1 8:0:20:19:9A:D9
mymachine> ip arp
arp add flane 192.168.2.63 00:20:2b:e0:03:87 # 8m58s
arp add flane 192.168.2.108 00:20:2b:03:0a:72 # 7m02s
arp add flane 192.168.2.109 00:20:2b:03:08:b1 # 2m24s
arp add flane 192.168.2.156 00:20:2b:03:09:c4 # 1m01s
arp add ether 192.168.50.1 08:00:20:19:9a:d9 # forever
arp add ether 192.168.50.57 00:20:af:2e:fa:3c # 3m25s
mymachine> ip arp delete flane 192.168.2.109
mymachine> ip arp list
arp add flane 192.168.2.63 00:20:2b:e0:03:87 # 8m46s
arp add flane 192.168.2.108 00:20:2b:03:0a:72 # 6m50s
arp add flane 192.168.2.156 00:20:2b:03:09:c4 # 49s
arp add ether 192.168.50.1 08:00:20:19:9a:d9 # forever
arp add ether 192.168.50.57 00:20:af:2e:fa:3c # 3m13s
mymachine> ip arp flush
mymachine> ip arp
# flane ARP table is empty
# ether ARP table is empty
mymachine> ip arp
arp add flane 192.168.2.108 00:20:2b:03:0a:72 # 10m58s
# ether ARP table is empty
```

(The last example shows that the MAC address for 192.168.2.108 has been automatically added

again, having been discovered by means of the ARP protocol.)

arprouting

Syntax:

```
arprouting [on]|off [<i/f>]
```

Description:

The "arprouting" command was intended to control whether a router would perform proxy ARP on an Ethernet-like interface; that is, reply with its own MAC address to an ARP request for any IP address that it would route to. However, it is not supported and is believed currently (ATMOS IP version 1.29) not to work correctly; the command is "hidden", not shown by "ip help".

autoloop

Syntax:

```
autoloop [on|off]
```

Description:

Displays or sets the "autoloop" setting. This has effect only when a loopback device is configured (see 4.3.3): in that case, it controls whether datagrams addressed to the machine's own IP addresses (and not just the loopback addresses 127.*.*.*) will be looped back. Configuration saving saves this information. By default autoloop is disabled. The "autoloop" command is "hidden", not shown by "ip help".

Example:

```
mymachine> ip autoloop
autoloop off
mymachine> ip device
# type dev file IP address
device ether ether //nice mtu 1500 192.168.2.1
device loop loop - mtu 2048 127.0.0.1
mymachine> ip ping 127.0.0.1
ip: ping - reply received from 127.0.0.1
mymachine> ip ping 192.168.2.1
ip: ping - transmit error: Host is down (rc=62)
mymachine> ip autoloop on
mymachine> ip ping 192.168.2.1
ip: ping - reply received from 192.168.2.1
```

config

Syntax:

```
config [save]
```

Description:

Displays the IP configuration (not including the "snmp" configuration), or saves it in flash memory.

The functionality of the "config" command is also accessible in the standard way through the config process (e.g. "config print ip"), if that process is present. However, when accessed through the config process, the "snmp" configuration *is* included.

Example:

```
mymachine> ip config
device add ether ether //nice mtu 1500 192.168.2.1
device add vlane ether //lane mtu 1500 192.168.55.1
subnet add vlane.home . 192.168.55.0 ff:ff:ff:00
subnet add ether.home . 192.168.2.0 ff:ff:ff:00
rip send ether 2
rip send vlane 2
rip accept ether 1 2
rip accept vlane 1 2
autoloop on
route add default 0.0.0.0 192.168.2.7 00:00:00:00 2 # MAN
relay ether ether
relay ether vlane
relay vlane vlane
ipatm lifetime 60
# IP host table:
# Port table:
router 520/UDP
snmp 161/UDP
tftp 69/UDP
```

```
telnet 23/TCP
mymachine> ip config save
Updating flash filing system ...
done
ip: configuration saved
```

See also:

snmp

Commands used for setting configuration displayed and saved by "config"

autoloop, device, ipatm, iphostname, portname, relay, rip, route, subnet

device

Syntax:

```
device
device add <i/f> <type> [<file>] [mtu <size>] [<IP address>]
device delete <i/f>
device flush
```

Description:

Displays the interfaces that IP is configured to use, or adds an interface to the configuration, or

deletes an interface, or all interfaces, from the configuration. Currently (ATMOS IP version 1.29), however, the commands to change the configuration do not take effect immediately (except when the "device add" command is run at start-up from the initialisation file). It is necessary to save the configuration (e.g. with "ip config save") and restart the system (e.g. with "ip restart") before they take effect. "device" will display both the current interfaces and those that have been configured but are not yet in effect. (Other commands apply only to the devices in effect, rather than to those configured; when adding a device, for example, one may need to issue the "device add" command, then the "config save" and reboot, then issue any other configuration commands that depend on the existence of the device, and then "config save" again.)

"<i/f>" is an arbitrary label for the interface, which is used in referring to it in subsequent commands. (It is often chosen to be the same as "<type>", though this is perhaps slightly confusing.)

"<type>" specifies the class of interface: Ethernet-like, IP-over-ATM, or loopback, as described in section 4.3. For an Ethernet-like or IP-over-ATM interface, "<file>" specifies the file name that will be opened to access the underlying device (which must support the Emerald interface for an Ethernet-like interface, and the Blue interface, at least, for an IP-over-ATM interface). For a loopback interface, "<file>" is not used, and can just be specified as "-" or omitted altogether. Several different values of "<type>" specify the same class of interface; they differ in that each implies a different default value for "<file>". As a result, for the most common interface configurations, "<file>" can be omitted, and one need only specify the appropriate value of "<type>". The supported values for "<type>" are

Class <type> Default file

Ethernet ether //nice or

//ethernet or

//edd

vlane //lane

flane //lec1

bridge //bridge

IP-over-ATM atm //q93b

atmpvc //atm

Loopback loop -where

the default file in the first case is "//nice" if either of the C pre-processor symbols ETHERMOD and RACK_IMAGE is defined; otherwise "//ethernet" if ETHER8X10 is defined; otherwise "//edd" if HYDROGEN_ETHERMOD is defined; if none of these symbols is defined, the name "ether" is not supported. (These defaults reflect the history of devices existing in various standard Virata images.)

The class IP-over-ATM includes both SVC-based and PVC-based IP-over-ATM; the decision whether to use SVCs or PVCs is made at initialisation, by testing the interface colours of the file: if it supports the Indigo interface, then SVCs are used, and otherwise PVCs.

"<mtu>" specifies the MTU (maximum transmission unit); that is, the size of the largest datagram (excluding media-specific headers) that IP will attempt to send through the interface. The value specified will be ignored if it is larger than the maximum supported by the interface

class, which is currently (ATMOS IP version 1.29) 1500 except for the loopback interface, unless the IP-over-ATM MTU has been changed as described in section 5.2; normally there is no point in setting the MTU less than this, so the "<mtu>" option is of little use.

"<IP address>" is the IP address that this system uses on the interface ; if it is not specified, the interface will be disabled until an IP address is supplied with the "ip enable" command. For a loopback interface, the address should be set to 127.0.0.1. (All addresses of the form 127.*.* will then be recognised as loopback addresses, as is normal practice.) For non-loopback interfaces, the subnet mask for the local network will be assumed to be ff:ff:ff:00 (e.g. a class C network); if the correct subnet mask is other than this then it will need to be set with the "subnet" command (section 7.48).

If there is no initialisation file //isfs/resolve (or //isfs/arptable) at all, then default interfaces are configured as if by the "device" commands `device add ether ether //edd` (if the symbol 'hydrogen' is set in the ATMOS system file)

`device add ether ether //nice` (otherwise)

`device add atm atm //q93b`

but in each case only if the file concerned ("//edd", "//nice", or "//q93b") can be opened.

Furthermore, if the IP process is given a command line (a little-used feature of ATMOS!) then each argument will be treated as a possible Ethernet-like file to open, given names "ether1", "ether2", and so on. For example, if the IP process is defined in the system file as "Process ip is tcp_ip/ip //bridge //lec1 " (and "//bridge" and "//lec1" can be opened), then the equivalents of the commands

`device add ether1 ether //bridge`

`device add ether2 ether //lec1`

will be processed, in addition to the others above.

Configuration saving saves the interface configuration.

Example:

`mymachine> ip device`

`# type dev file IP address`

`device ether ether //nice mtu 1500 192.168.2.1`

`device vlane ether //lane mtu 1500 192.168.55.1`

`mymachine> ip device add loop loop 127.0.0.1`

Change will have no effect until after config save and restart.

`mymachine> ip device delete vlane`

Change will have no effect until after config save and restart.

`mymachine> ip device`

`# type dev file IP address`

`device ether ether //nice mtu 1500 192.168.2.1`

`device vlane ether //lane mtu 1500 192.168.55.1 # DELETED`

`device loop loop - mtu 2048 127.0.0.1 # ADDED`

Additions/deletions will have no effect until after config save and restart.

See also:

enable

subnet

disable

Syntax:

`disable [<i/f>]`

Description:

Disables all interfaces, or just a specified interface.

Example:

`mymachine> ip disable vlane`

`mymachine> ip device`

`# type dev file IP address`

`device ether ether //nice mtu 1500 192.168.2.1`

`device vlane ether //lane mtu 1500 192.168.55.1 # DISABLED`

See also:

device

enable

enable

Syntax:

```
enable [/f] [mtu <size>] [<IP address>]
```

Description:

Enables all interfaces, or just a specified interface. Can also be used to set the MTU and IP address on an interface when enabling it (or change them on an interface that is already enabled); see the “device” command for details on these.

Configuration saving saves the MTU and IP addresses, but not the disabled/enabled state.

Example:

```
mymachine> ip enable vlane 192.168.56.3
ip/vlane: IP address 192.168.56.3
mymachine> ip device
# type dev file IP address
device ether ether //nice mtu 1500 192.168.2.1
device vlane ether //lane mtu 1500 192.168.56.3
```

See also:

device
disable
subnet

errors

Syntax:

```
errors
```

Description:

Turns on tracing of various unusual events; equivalent to “trace errors”.

The “errors” command is “hidden”, not shown by “ip help”.

Example:

```
mymachine> ip errors
ip: currently tracing errors
```

See also:

noerrors
trace

etherfiles

Syntax:

```
etherfiles
```

Description:

Lists the file names for the underlying devices for all Ethernet-like interfaces, and displays the number of packets (usually none) that are queued for transmission awaiting an ARP reply. The “etherfiles” command is “hidden”, not shown by “ip help”.

Example:

```
mymachine> ip etherfiles
ether: //nice, 1 queued for tx
vlan: //lane
atm: (no ethernet device)
```

See also:

device

files

Syntax:

```
files [full]
files <assoc>
```

Description:

Lists the files (associations) that other applications (or, internally, RIP) have opened on “//ip”. More detailed information on an association can be displayed by specifying the association number, or on all associations by specifying “full”. The information for each association may include an interface name (“ether” or “vlan” in the example below). This can be either the interface last used to send a packet on the association or, for a new association, the interface

that is expected to be used for packets to the remote host. This interface can change over the lifetime of an association; in particular, for a UDP association not bound to a specific remote host it may change each time a packet is sent to a different destination. (In other cases it will normally change only as a result of routing changes.) The "files" command is "hidden", not shown by "ip help".

Example:

```
mymachine> ip files
1: rw+ ether 192.168.2.1 TCP port telnet (23) Established to 192.168.2.2
port 1071 1 rx requests
2: rw+ ether <noaddr> UDP port snmp (161) 3 rx requests
3: rw+ <unset> <noaddr> UDP port tftp (69) 4 rx requests
4: rw+ <unset> <noaddr> UDP port router (520) 2 rx requests
5: w vlane <noaddr> UDP port router (520)
6: rw+ <unset> <noaddr> UDP port 2050 4 rx requests
7: rw+ <unset> <noaddr> UDP port 2051 4 rx requests
8: rw+ <unset> <noaddr> UDP port 2052 4 rx requests
9: rw+ <unset> <noaddr> UDP port 2053 4 rx requests
mymachine> ip files 3
3: rw+ <unset> <noaddr> UDP port tftp (69) 4 rx requests
//ip/TYPE=UDP/LPORT=69/TIMEOUT_CONX=1000/TIMEOUT_LISTEN=0/TIMEOUT_IDLE=0/
RETRY_CONX=2/TOS=routine/DELAY=normal/THROUGHPUT=normal/RELIABILITY=normal/BUFFERRX=off/B
UFFER_TXSIZE=-1/BUFFER_RXSIZE=-
1/FRAGMENT=on/TTL=60/OPTIONS=off/CHECKSUM=on/TIMEOUT_USER=540000
```

flush

Syntax:

```
flush <assoc>
```

Description:

Given an association number (see "files" command) that corresponds to a TCP association, this does a TCP "push" (see RFC 793), which, roughly speaking, causes the data sent so far to be delivered as quickly as possible to the recipient, without waiting to be buffered with subsequent data. The "flush" command is "hidden", not shown by "ip help"; it is of little or no use.

See also:

```
files
```

get

Syntax:

```
get <file>
```

Description:

Reads and executes commands from a file. The commands in the file are in the same format as those documented in this chapter, with no "ip" prefix. They can contain comments, introduced by the "#" character. The "get" command is "hidden", not shown by "ip help".

Example:

```
mymachine> ip get //isfs/cmdfile
```

help

Syntax:

```
help
help <cmd>
help all
```

Description:

Displays a summary of available commands, more detailed information on a particular command, or more detailed information on all commands. (As described in section 7.1, some commands are "hidden" and are not displayed by "help" or "help all"; help is still available on these using the "help <cmd>" form if one knows the name of the command.)

Example:

```
mymachine> ip help
Commands are:
? arp config device
disable enable help ipatm
norelay ping relay restart
```

```
rip route routes snmp
stats subnet uptime version
Use "ip help all" or "ip help <command>" for syntax
mymachine> ip help arp
arp syntax:
arp <cmd> - execute arp subcommand
arp help - list subcommands available
```

ipatm abort

Syntax:

```
ipatm abort <n>
```

Description:

Closes an IP-over-ATM SVC; the number <n> is as displayed by "ipatm files". If there is still traffic being sent to the destination concerned, IP will soon open a new SVC to the destination.

Example:

```
mymachine> ip ipatm abort 14
```

See also:

ipatm files

ipatm arp

Syntax:

```
ipatm arp [list]
```

Description:

Lists the cached mappings from IP addresses to ATM addresses; only relevant when using IP-over-ATM with SVCs. (The "list" parameter is optional and makes no difference to the behaviour.)

Example:

```
mymachine> ip ipatm arp
192.168.5.72 47.00.83.10.a2.b1.00.00.00.00.00.00.00.00.20.2b.01.00.07.00
192.168.5.33 47.00.83.10.a4.00.00.00.00.00.00.00.00.00.20.2b.01.00.19.00
192.168.5.111 47.00.83.10.e2.00.00.00.00.20.2b.01.01.a8.00.20.2b.01.01.a8.00
```

ipatm arpserver

Syntax:

```
ipatm arpserver [i/f] [<ATM address>|here]
```

Description:

Displays or sets the ATMARP server used for an interface, which must be an IP-over-ATM interface using SVCs. The interface name is optional when displaying: if omitted, the ATMARP servers for all such interfaces are listed. (Since currently there can only be one such interface, this behaviour is present only for possible consistency with future versions.) The parameter "here" causes no ATMARP server to be used; only the local ATMARP cache will be consulted when setting up an SVC. This will normally be used when this machine is the ATMARP server for the local network.

Configuration saving saves this information.

Example:

```
mymachine> ip ipatm arpserver
ipatm arpserver atm here
mymachine> ip ipatm arpserver atm 47.0.83.10.a2.0.0.0.0.0.0.0.0.0.0.20.2b.4.3.8.0
mymachine> ip ipatm arpserver atm
ipatm arpserver atm 47.00.83.10.a2.00.00.00.00.00.00.00.00.00.00.00.20.2b.04.03.08.00
```

ipatm files

Syntax:

```
ipatm files
```

Description:

Lists the IP-over-ATM connections, listeners, and slots for available connections.

Example:

```
mymachine> ip ipatm files
i/f atm 0 transmissions queued, 6 free connections, 4 listeners
```

```
0: on atm Connected to 192.168.220.48, 2 rx buffers idle 0ms
1: on atm Listening, 1 rx buffers (in use)
2: on atm Listening, 1 rx buffers (in use)
3: on atm Listening, 1 rx buffers (in use)
4: on atm Listening, 1 rx buffers (in use)
5: on atm Idle, 0 rx buffers
6: on atm Idle, 0 rx buffers
7: on atm Idle, 0 rx buffers
8: on atm Idle, 0 rx buffers
9: on atm Idle, 0 rx buffers
10: on atm Idle, 0 rx buffers
```

ipatm help

Syntax:

```
ipatm help [<cmd>|all]
```

Description:

Displays help on "ipatm" subcommands.

Example:

```
mymachine> ip ipatm help
Commands are:
? abort arp arpserver
files help lifetime pvc
Use "ip ipatm help all" or "ip ipatm help <command>" for syntax
mymachine> ip ipatm help arp
arp syntax:
ipatm arp [list] - list ARP cache entries
```

ipatm lifetime

Syntax:

```
ipatm lifetime <secs>
```

Description:

Displays or sets idle time-out for IP-over-ATM SVCs: if there is no traffic on an SVC for this period, then it will be disconnected. (It might be disconnected before this period in order to make room for new connections.) There is no way to disable the time-out, but "ip ipatm lifetime 999999" will have much the same effect.

Configuration saving saves this information.

The default lifetime is 60 seconds.

Example:

```
mymachine> ip ipatm lifetime.ATMOS TCP/IP Functional Specification DO-007285-PS
©1998 Virata Limited 38
Idle lifetime for connections: 1m
mymachine> ip ipatm lifetime 90
Idle lifetime for connections: 1m30s
```

ipatm pvc

Syntax:

```
ipatm pvc
ipatm pvc add <i/f> <vci>/[<IP address>] [/<pcr>] [<port>]
ipatm pvc delete <vci> [<port>]
ipatm pvc flush
```

Description:

Lists configured PVCs for use by IP-over-ATM; configures another; deletes one; or deletes all.

"<i/f>" is the name of an interface configured for IP-over-ATM using PVCs.

"<vci>" is the VCI to use for the PVC. The range of possible VCIs depends on the system.

"<IP address>" is the IP address of the machine at the other end of the PVC. If it is not specified, ATMOS TCP/IP will use Inverse ATMARP (RFC 1577) to determine the IP address; if it is specified, then Inverse ATMARP will not be used.

"<pcr>" is the peak cell rate, in cells per second. The default is 60000. (If neither IP address nor

PCR is specified, the "/" after the VCI can be omitted.)

"<port>" is the port name: it must be specified if the machine is a switch, and not otherwise.

Configuration saving saves this information.

Example:

```
mymachine> ip ipatm pvc add atm 60 a3
mymachine> ip ipatm pvc add atm 61//50000 b1
mymachine> ip ipatm pvc add atm 62/192.168.4.32 b1
mymachine> ip ipatm pvc
ipatm pvc atm 60//60000 A3
ipatm pvc atm 61//50000 B1
ipatm pvc atm 62/192.168.4.32/60000 B1
```

iphostname

Syntax:

```
iphostname add <IP address> <name>
iphostname flush
iphostname list
iphostname help [all|<cmd>]
```

Description:

Sets up a mapping between an IP address and a symbolic name; deletes all such mappings; lists the mappings; or displays help on the "iphostname" command. The symbolic names can be used in most IP commands where an IP address is required, and as values of the attributes LHOST and RHOST (section 6.1). They are also displayed and returned as attribute values in place of numerical addresses, when a suitable mapping exists. The Damson interface (5.4.2) allows other processes to query the mapping. The "iphostname" command is "hidden", not shown by "ip help".

Configuration saving saves this information.

noerrors

Syntax:

```
noerrors
```

Description:

Undoes the effect of the "errors" command; equivalent to "untrace errors". The "noerrors" command is "hidden", not shown by "ip help".

Example:

```
mymachine> ip noerrors
ip: currently tracing nothing
```

See also:

```
errors
untrace
```

norelay

Syntax:

```
norelay [all | <i/f> [<i/f>] [forward]]
```

Description:

Turns off forwarding between interfaces; see the "relay" command for more details. The command "norelay" with no parameters is equivalent to "norelay all": it turns off all forwarding.

Configuration saving saves this information.

Example:

```
mymachine> ip relay
relay ether ether
relay ether vlane
relay vlane vlane
mymachine> ip norelay ether vlane forward
relay ether ether
relay vlane ether forward
relay vlane vlane
```

See also:

```
relay
```

ping

Syntax:

```
ping <IP address> [<ttl> [<size>]]
```

Description:

Sends an ICMP Echo message to the specified IP address. “<ttl>” (default 30) is the TTL (time-to-live) to use. A crude “traceroute” functionality can be obtained by repeating the “ping” command with increasing TTL values, starting with 1. “<size>” (default 56) is the data size of the Echo message. This does not include the IP header (20 bytes) and the ICMP header (8 bytes). ATMOS TCP/IP waits 10 seconds for a reply to the message; if none arrives, it reports the lack of a reply (and returns the TELL message, or redisplay the prompt). Any reply arriving after .ATMOS this time-out will be reported as a background message. (Whereas a reply arriving before the time-out expires is, of course, reported in the foreground.) A reply is an ICMP Echo Reply message, or an ICMP error message reporting destination unreachable, time exceeded, or (as should never happen) a parameter problem. ICMP redirect and source quench messages are reported, but ATMOS TCP/IP continues to wait for a final reply or time-out.

Example:

```
mymachine> ip ping 192.168.4.13 1
ip: ping - 192.168.1.9 reports pkt #5834 to 192.168.4.13: time-to-live
exceeded
mymachine> ip ping 192.168.4.13 2
ip: ping - reply received from 192.168.4.13
mymachine> ip ping 192.168.77.77
ip: ping - no reply received
```

portname

Syntax:

```
portname add <name> <number>[/<protocol>]
portname flush
portname list
portname read <file>
portname help [all|<cmd>]
```

Description:

Sets up a mapping between a UDP or TCP port and a symbolic name; deletes all such mappings; lists the mappings; reads the mappings from a file; or displays help on the “portname” command.

The symbolic names can be used as values of the attributes LPORT and RPORT (section 6.1) provided the protocol type (UDP or TCP) is appropriate. They are also displayed in place of port numbers, when a suitable mapping exists. The Damson interface (5.4.2) allows other processes to query the mapping.

“<protocol>” should be either “UDP” or “TCP”; it can be omitted, but that is not very useful.

For “portname read”, the file is in the same format as //isfs/services (section 5.3), which is the same as the output from “portname list”. The “portname” command is “hidden”, not shown by “ip help”.

Configuration saving saves this information.

Example:

```
mymachine> ip portname flush
mymachine> ip portname add someport 105/tcp
mymachine> ip portname list
someport 105/TCP
mymachine> ip portname read //isfs/services
mymachine> ip portname list
router 520/UDP
snmp 161/UDP
tftp 69/UDP
telnet 23/TCP
someport 105/TCP
```

protocols

Syntax:

```
protocols
```

Description:

Displays information on the protocols supported by ATMOS TCP/IP. The output will always be the same for a given version of ATMOS TCP/IP. The "protocols" command is "hidden", not shown by "ip help". (Currently, ATMOS IP version 1.29, there is a fault in the output: it claims falsely that UDP can reassemble fragments.)

Example:

```
mymachine> ip protocols
ICMP - IP ID 1, CL protocol, can't reassemble fragments
TCP - IP ID 6, CO protocol, can't reassemble fragments
UDP - IP ID 17, CL protocol, can reassemble fragments
```

relay**Syntax:**

```
relay
relay all | <i/f> [<i/f>] [forward]
```

Description:

Displays or sets what forwarding ATMOS TCP/IP will do between interfaces. The combinations of setting forwarding can be a bit confusing; they behave as follows:

Command: Enables forwarding:

```
relay all from every interface to every non-loopback interface
relay if1 from if1 to every non-loopback interface, and from
every interface to if1
relay if1 forward from if1 to every non-loopback interface
relay if1 if2 from if1 to if2 and from if2 to if1
relay if1 if2 forward from if1 to if2
```

(Don't confuse the "forward" keyword, which indicates one-way relaying, with the term "forwarding"!)

To disable forwarding, use the "norelay" command.

Configuration saving saves this information.

By default all forwarding is disabled.

Example:

```
mymachine> ip relay
No relaying is being performed
mymachine> ip relay ether vlane forward
relay ether vlane forward
mymachine> ip relay ether forward
relay ether ether
relay ether vlane forward
mymachine> ip relay ether vlane
relay ether ether
relay ether vlane
mymachine> ip relay all
relay ether ether
relay ether vlane
relay vlane vlane
```

See also:

norelay

restart**Syntax:**

```
restart
```

Description:

Reboots the system.

Example:

```
mymachine> ip restart
```

rip accept**Syntax:**

```
rip accept [all|<i/f>] [none|<version>*]
```

Description:

Controls for which version or versions of RIP (RIP version 1, RFC 1058, or RIP version 2, RFC1723) ATMOS TCP/IP will accept incoming information on each interface. Configuration saving saves this information. By default both RIP versions are accepted on all interfaces ("rip accept all 1 2 ").

Example:

```
mymachine> ip rip accept all 1 2
mymachine> ip rip accept ether 2
mymachine> ip rip allowed
rip send ether none
rip send vlane none
rip accept ether 2
rip accept vlane 1 2
```

See also:

```
rip allowed
rip send
```

rip allowed

Syntax:

```
rip allowed
```

Description:

Displays the RIP versions that will be accepted and sent on each interface.

Example:

```
mymachine> ip rip allowed
rip send ether 2
rip send vlane 2
rip accept ether 1 2
rip accept vlane 1 2
```

See also:

```
rip accept
rip send
```

rip boot

Syntax:

```
rip boot
```

Description:

Broadcasts a request for RIP information from other machines. ATMOS TCP/IP does this automatically when it first starts up, and the routing information should be kept up to date by regular broadcasts from the other machines, so this command is normally of little use.

Example:

```
mymachine> ip rip boot
```

rip help

Syntax:

```
rip help [<cmd>|all]
```

Description:

Displays help on "rip" subcommands.

Example:

```
mymachine> ip rip help
Commands are:
? accept allowed boot
help hostroutes killrelay poison
relay relays rxstatus send
trigger
Use "ip rip help all" or "ip rip help <command>" for syntax
mymachine> ip rip help boot
boot syntax:
rip boot - broadcast RIP request for routes
```

rip hostroutes

Syntax:

```
rip hostroutes [off]
```

Description:

Sets or clears the "hostroutes" flag; ATMOS TCP/IP will accept RIP routes to individual hosts only if this flag is on. If the flag is off, then RIP version 1 routes that appear to be to individual hosts will be treated as if they were to the network containing the host; RIP version 2 routes to individual hosts will be ignored. (The reason for this difference is that RIP version 1 does not allow specification of subnet masks; a RIP version 1 route that appears to be to an individual host might in fact be to a subnet, and treating it as a route to the whole network may be the best way to make use of the information.)

To see the state of the flag without changing it, the "config" command must be used.

Configuration saving saves this information.

By default the "hostroutes" flag is off.

Example:

```
mymachine> ip rip hostroutes off
```

See also:

```
config
```

rip killrelay

Syntax:

```
rip killrelay <relay>
```

Description:

Deletes a RIP relay. See "rip relay" for information on RIP relays.

See also:

```
rip relay
```

rip poison

Syntax:

```
rip poison [off]
```

Description:

Sets or clears the "poisoned reverse" flag. If this flag is on, ATMOS TCP/IP performs "poisoned reverse" as defined in RFC 1058; see that RFC for discussion of when this is a good thing. To see the state of the flag without changing it, the "config" command must be used.

Configuration saving saves this information.

By default the "poisoned reverse" flag is off.

Example:

```
mymachine> ip rip poison
```

rip relay

Syntax:

```
rip relay <RIP version> <name> [/f] [<timeout>]
```

Description:

Configures a RIP relay. RIP relays were designed as a means of using RIP on a non-broadcast medium (currently, only IP-over-ATM); on such an interface, ATMOS TCP/IP will send RIP information individually to each configured RIP relay, instead of broadcasting it. However, the RIP relay support has not been recently tested and is not believed to be reliable; furthermore, configuration saving does not save the RIP relay configuration. On a non-broadcast medium, therefore, it is preferable to use static (manually configured) routes.

See also:

```
rip killrelay  
rip relays
```

rip relays

Syntax:

```
rip relays
```

Description:

Displays the configured RIP relays. See "rip relay" for information on RIP relays.

See also:

rip relay

rip rxstatus**Syntax:**

```
rip rxstatus
```

Description:

Displays the status of the RIP packet reception mechanism. This command is of little or no use except for debugging.

Example:

```
mymachine> ip rip rxstatus
RIP has 2 reading threads and 1 worker
The worker is waiting for something to do
The readers have filled 0/6 buffers and have 4 available
Maximum work queue size was 0
Receiver 0 has 1 buffer and is not waiting for the worker
Receiver 1 has 1 buffer and is not waiting for the worker
```

rip send**Syntax:**

```
rip send [all|<i/f>] [none|<version>*]
```

Description:

Controls which version or versions of RIP (RIP version 1, RFC 1058, or RIP version 2, RFC1723) ATMOS TCP/IP will use to broadcast routing information on each interface. If both versions are specified, routing information is broadcast in duplicate, once using each version. Specifying "all" affects all interfaces except the loopback interface (if any). Configuration saving saves this information.

By default RIP version 2 only is used on all non-loopback interfaces ("rip send all 2").

Example:

```
mymachine> ip rip send all 2
mymachine> ip rip send ether 1
mymachine> ip rip allowed
rip send ether 1
rip send vlane 2
rip accept ether 1 2
rip accept vlane 1 2
```

See also:

```
rip accept
rip allowed
```

rip trigger**Syntax:**

```
rip trigger
```

Description:

Triggers broadcast of routing information. Normally the routing information is broadcast every 30 seconds; "rip trigger" causes it to be sent almost immediately rather than waiting for the next time it is due. This command is normally of little use.

Example:

```
mymachine> ip rip trigger
```

route**Syntax:**

```
route
route add <name> <dest> <relay> [<mask> [<cost> [<timeout>]]]
route delete <name>
route flush
```

Description:

Lists routes; adds or deletes a static route; or deletes all routes.

"<name>" is an arbitrary name specified to "route add" that can be used to delete the route using "route delete".

"<dest>" is the IP address of the network being routed to (only those bits of "<dest>" corresponding to bits set in "<mask>" are relevant).

"<relay>" is the IP address of the next-hop gateway for the route.

"<mask>" (default ff:ff:ff:00) is the subnet mask of the network being routed to, specified as four hexadecimal numbers separated by colons. For example, 0:0:0:0 is a default route (matches everything without a more specific route), ff:ff:ff:0 would match a Class C network, and ff:ff:ff:ff is a route to a single host. (Note: the default is not always sensible; in particular, if "<dest>" is 0.0.0.0 then it would be better for the mask to default to 0:0:0:0. This may change in future versions.)

"<cost>" (default 1) is the number of hops counted as the cost of the route, which may affect the choice of route when the route is competing with routes acquired from RIP. (But note that using a mixture of RIP and static routing is not advised.)

"<timeout>" (default 0, meaning that the route does not time out) is the number of seconds that

the route will remain in the routing table. Note that the routing table does not contain routes to the directly connected networks, without going through a gateway. ATMOS TCP/IP routes packets to such destinations by using the information in the device and subnet tables instead. The "route" command (with no parameters) displays the routing table. It adds a comment to each route with the following information:

- How the route was obtained; one of

MAN — configured by the "route" command

RIP — obtained from RIP

ICMP — obtained from an ICMP redirect message

SNMP — configured by SNMP network management;

- The time-out, if the route is not permanent;

- The original time-out, if the route is not permanent;

- The name of the interface (if known) that will be used for the route; .ATMOS TCP/IP Functional Specification

DO-007285-PS

©1998 Virata Limited 47

- An asterisk ("*") if the route was added recently and RIP has not yet processed the change (the asterisk should disappear within 30 seconds, when RIP next considers broadcasting routing information).

Configuration saving saves this information. (Only the routes configured by the "route" command are saved or displayed by "config".)

Example:

```
mymachine> ip route add default 0.0.0.0 192.168.2.3 0:0:0:0
mymachine> ip route add testnet1 192.168.101.0 192.168.2.34
mymachine> ip route add testnet2 192.168.102.0 192.168.2.34 ff:ff:ff:0 1 60
mymachine> ip route
route add testnet2 192.168.102.0 192.168.2.34 ff:ff:ff:00 1 # MAN 58s/1m via
ether *
route add testnet1 192.168.101.0 192.168.2.34 ff:ff:ff:00 1 # MAN via ether
route add default 0.0.0.0 192.168.2.3 00:00:00:00 1 # MAN via ether
```

See also:

device

subnet

routeflush

Syntax:

```
routeflush [/f] [all]
```

Description:

Removes routes from the route table. If "<i/f>" is specified, only routes through the named interface are removed. If "all" is not specified, only host routes (those with a mask of ff:ff:ff:ff) are removed.

The "routeflush" command is "hidden", not shown by "ip help".

Configuration saving saves this information.

Example:

```
mymachine> ip routeflush ether all
mymachine> ip routeflush
```

See also:
route

routes

Syntax:

```
routes
```

Description:

Lists routes. (The same as "route", with no parameters.)

Example:

```
mymachine> ip routes
route add testnet1 192.168.101.0 192.168.2.34 ff:ff:ff:00 1 # MAN via ether
route add default 0.0.0.0 192.168.2.3 00:00:00:00 1 # MAN via ether
```

See also:

```
route
```

snmp

Syntax:

```
snmp access [read|write|delete|flush] <parameters>
snmp config [save]
snmp help [<cmd>|all]
snmp trap [add|delete|flush|list] <parameters>
```

Description:

Manages the list of SNMP community names (also used as passwords by other applications, such as telnet) and the list of SNMP trap destinations. See chapter 9 for the interface to this information.

The syntax of the commands is documented in the ATMOS SNMP Functional Specification [2]. Note that in standard ATMOS systems the console is configured to allow the commands to be accessed by typing just "snmp ..." instead of "ip snmp ..." at the command line.

stats

Syntax:

```
stats arp|icmp|ip|tcp|udp [reset]
stats help [<cmd>|all]
```

Description:

Displays or clears a subset of IP statistics.

Example:

```
mymachine> ip stats udp
ip: UDP receptions delivered to users: 0
ip: UDP receptions with no users: 170
ip: Otherwise discarded UDP receptions: 0
ip: Transmitted UDP packets: 35
mymachine> ip stats udp reset
mymachine> ip stats udp
ip: UDP receptions delivered to users: 0
ip: UDP receptions with no users: 0
ip: Otherwise discarded UDP receptions: 0
ip: Transmitted UDP packets: 0
```

subnet

Syntax:

```
subnet
subnet add <name> <i/f> <IP address> <mask>
subnet delete <name>
subnet flush
```

Description:

Lists defined subnets; defines a subnet; deletes a subnet definition; or deletes all subnet definitions. “<name>” is a label, that can be specified by “subnet add” and later used by “subnet delete” to delete the subnet. “<i/f>” is not used, but is present for historical reasons and must be specified as either “.” or a valid interface name. “<IP address>” is the IP address of the subnet being defined (only those bits of “<dest>” corresponding to bits set in “<mask>” are relevant). “<mask>” is the subnet mask of the subnet being defined, specified as four hexadecimal numbers separated by colons.

A subnet is defined automatically for each interface, with a name formed by appending “.home”

to the device name. The only significant use for the “subnet” command is to change the masks for these automatic subnets, if the default masks (see “device” command) are not correct. (Subnet definitions for other subnets *can* also be useful in conjunction with RIP version 1, which

does not communicate subnet masks, but this is not very common.)

Configuration saving saves this information.

Example:

```
mymachine> ip device
# type dev file IP address
device ether ether //nice mtu 1500 192.168.2.1
device vlane ether //lane mtu 1500 192.168.55.1
mymachine> ip subnet
subnet vlane.home . 192.168.55.0 ff:ff:ff:00 vlane
subnet ether.home . 192.168.2.0 ff:ff:ff:00 ether
mymachine> ip subnet add vlane.home . 192.168.55.1 ff:ff:fc:0
mymachine> ip subnet
subnet vlane.home . 192.168.52.0 ff:ff:fc:00 vlane
subnet ether.home . 192.168.2.0 ff:ff:ff:00 ether
```

See also:

device
route

trace

Syntax:

```
trace [<option>]
```

Description:

Turns on an IP tracing option, or lists the available options. Note that tracing messages are written to background output, so with the standard console one must use the “event” commands to see them.

An option can be:

- One of various keywords. The details of just what tracing messages are enabled by each keyword are not documented here; one must examine the source if one really wants to know.
- An association number (see the “files” command). For a TCP association this turns on detailed tracing of events (including all packet transmission and reception) on that association; for a UDP association it has no effect. The “files” command shows (by appending “TRACE”) whether each association has tracing enabled.
- An interface name (see the “device” command). This turns on tracing of every packet sent or received through the interface (one line per packet). The “device” command shows (by appending “TRACE”) whether each interface has tracing enabled.
- “ip”. This turns on tracing for all interfaces.
- “all”. This turns on all tracing.

Note that “trace” does *not* display which associations and interfaces are being traced; one must

use the “files” and “device” commands for that. The “trace” command is “hidden”, not shown by “ip help”. It is useful mainly for debugging and troubleshooting.

Example:

```
mymachine> ip trace
ip: try trace - <assoc no> <i/f name> all ip errors resolve ipatm atmarp
iploop arp ipether icmp udp tcp tcphdr tcpstate routes riptx riprx names
ip: currently tracing nothing
mymachine> ip trace tcp
ip: currently tracing tcp
```

See also:

errors
untrace

untrace

Syntax:

untrace [<option>]

Description:

Turns off IP tracing options. The syntax is the same as for "trace"; in particular, "untrace all" turns off all tracing. The "trace" command is "hidden", not shown by "ip help".

See also:

noerrors
trace

uptime

Syntax:

uptime

Description:

Displays the time for which the ATMOS system has been running.

Example:

```
mymachine> ip uptime
up 8 hours 33 minutes
```

version

Syntax:

version

Description:

Displays the system version, ATM address, and MAC address. (An obsolescent option "ip" still exists, but "version ip" now displays misleading information and should not be used.)

Example:

```
mymachine> ip version
Modem version 5.00.0.4 (August 27 1998) (DEBUG)
ATM address: 47.00.83.10.a2.b2.c2.00.00.00.00.00.00.00.20.2b.00.00.38.00
MAC address: 0:20:2b:0:0:38 .ATMOS TCP/IP Functional Specification DO-007285-PS
```

7.53 ?

Syntax:

?
? <cmd>
? all

Description:

The "?" command is simply a synonym for the "help" command, and behaves in the same way.

See also:

help

NAT Console Commands

ip nat

Syntax:

ip nat add|delete <i/f name>

Description:

This command adds or removes NAT functionality from the named interface. The interface name is the name as listed by the **ip device** command. NAT should always be enabled only on the interface connecting to the public network, not the interface connecting to the private network.

Example:

```
bd2000> ip nat add ethernet
```

See also:

nat event

Syntax:

```
nat event [n]
```

Description:

This command displays or sets the current level of event tracing in the NAT process. Larger values of **n** result in more verbose trace output, for example:

Event level Output

- 1 Only show fatal errors, e.g. lack of system resources
- 2 Only show important information and problems
- 3 Show the creation of new sessions
- 4 Show trace output for discarded packets
- 5 Show trace output for all packets

All trace messages are printed as background output, and therefore will not be displayed asynchronously on the console unless the **event show** command has been issued.

Example:

```
bd2000> nat event
Event level: 1
bd2000> nat event 2
```

See also:

```
event show
```

nat help

Syntax:

```
nat help [command]
```

Description:

Lists the commands provided by the NAT console interface. If an optional command name is supplied, help on that command's usage is displayed.

See also:

nat interfaces

Syntax:

```
nat interfaces
```

Description:

The **nat interfaces** command displays the IP router ports on which NAT is currently enabled. For each of these, a status and IP address is listed. The IP address is discovered automatically from the IP stack.

The status shows the user whether NAT is currently operational on that interface ("enabled"), or whether NAT is still waiting to find out the interface's IP address ("not ready").

Example:

```
bd2000> nat interfaces
Name Status IP address
ethernet enabled 194.129.40.2
ppp not ready
```

-See also:

nat inbound

Syntax:

```
nat inbound list
nat inbound add <i/f> <port>/<proto> <new IP> [quiet]
nat inbound delete <#>
nat inbound flush
```

Description:

This command enables the user to list or to set up a series of rules, to determine what happens to incoming traffic. By default all incoming packets, other than packets arriving in response to outgoing traffic, will be rejected. The **nat inbound add** command allows packets arriving on a specific port and IP protocol to be forwarded to a machine on the private network. **<i/f>** is an interface name as shown by the **nat interface list** command; **<port>** is the destination UDP or TCP port number to match in the incoming traffic; **<proto>** is the IP protocol, either "udp" or "tcp"; **<new IP>** is the new IP address on the private network which

the packet's destination IP address should be translated to. If a rule is added for an interface on which NAT is not enabled, the rule is added anyway but a warning is printed to alert the user to this fact. **quiet** is a special option which should not normally be issued at the console, and causes this warning to be suppressed. The **quiet** option is automatically added by NAT to when writing its configuration to flash; this is because when a system boots, the NAT process reads in these rules before IP has registered any interfaces. **nat inbound list** shows the current rules for inbound traffic, including all the arguments passed to the **nat inbound add** command. **nat inbound delete** removes a rule, where <#> is the rule number as shown by the **nat inbound list** command. **nat inbound flush** removes all the rules.

Example:

```
bd2000> nat inbound add ethernet 80/TCP 192.168.219.38
bd2000> nat inbound list
# Interface Port/Proto New IP address
1 ethernet 80/tcp 192.168.219.38
2 r1483 21/tcp 192.168.219.40
bd2000> nat inbound delete 2
```

See also:

nat info

Syntax:

```
nat info
```

Description:

This command displays the values of various parameters which are defined in the module file, for example the session table size and the session timeouts. NAT's current memory usage is also displayed.

Example:

```
bd2000> nat info
Interface table size 1 (116 bytes)
Session table size per interface: 128 (6656 bytes)
Total: 6656 bytes
Hash table size per interface: 128 (512 bytes)
Total: 512 bytes
Fragment table size per interface: 32 (640 bytes)
Total: 640 bytes
Max queued buffers: 16
Fragment timeout: 30
Support for incoming fragments: enabled
Support for outgoing fragments: enabled
Session timeouts:
ICMP query: 10
UDP: 30
TCP (established): 300
TCP (other): 15
Initial port number: 10000
```

See also:

```
nat version
```

nat protocol

Syntax:

```
nat protocols
```

Description:

The **nat protocols** command lists the application level gateways (ALGs) provided in the current image in order to support particular higher-level protocols, and the port or ports which each ALG monitors.

Example

```
bd2000> nat protocols
Name Port/IP protocol
ftp 21/tcp
```

See also:

nat sessions

Syntax:

```
nat sessions <i/f> [all | summary]
```

Description:

The **nat sessions** command displays a list of currently active NAT sessions on the interface **<i/f>**. In this context, a session is a pair of source IP addresses and port numbers (and corresponding new port number) that NAT regards as one side of an active connection. For each TCP or UDP session active, the source and destination IP address and port number, and the local port number and the age of the session, are printed. The **all** option causes the **sessions** command to print out information on every session, including sessions which have timed out. Normally the **sessions** command only shows active sessions (those which have not timed out).

The **summary** command does not show detailed information on each session, but only prints out

the total number of active, timed out and available sessions.

Example:

```
bd2000> nat sessions ppp
Proto Age NAT port Private address/port Public address/port
TCP 34 1024 192.168.219.38/3562 194.129.50.6/21
TCP 10 1025 192.168.219.64/2135 185.45.30.30/80
Total:
2 sessions active
101 sessions timed out
126 sessions available
```

See also:

nat stats

Syntax:

```
nat stats <i/f> [reset]
```

Description:

This command displays various statistics gathered by NAT on the interface **<i/f>**. These are cumulative totals since power on, or since the **reset** keyword was given. The **nat stats** command does not provide the total number of packets or bytes transferred, as this information is normally

available from the device driver on the interface which NAT is filtering.

Example:

```
bd2000> nat stats ethernet
Outgoing TCP sessions created: 456
Outgoing UDP sessions created: 123
Outgoing ICMP query sessions: 12
Outgoing ICMP errors: 0
Incoming ICMP errors: 6
Incoming connections refused: 2
Sessions deleted early: 0
Fragments currently queued:
```

See also:

nat version

Syntax:

```
nat version
```

Description:

This command displays NAT's internal version number.

Example:

```
bd2000> nat version
NAT Version 1.00
```

See also:

```
nat info
```

nat dump

Syntax:

```
nat dump on|off
```

Description:

This command is only available in debug builds.

nat dump causes a detailed dump of the information in each packet's header to be printed both

before and after translation. This command is provided for debug purposes.

Example:

```
bd2000> nat dump on
```

See also:

nat fragments

Syntax:

```
nat fragments <i/f name>
```

Description:

This command is only available in debug builds.

nat fragments prints information on the queues in which NAT holds fragmented IP datagrams, displaying the IP datagram identifier, the number of fragments queued and a NAT session

pointer for each queue. This command is provided for debug purposes only.

Example:

```
bd2000> nat fragments ether
```

See also:

nat hashtable

Syntax:

```
nat hashtable <i/f name>
```

Description:

This command is only available in debug builds.

nat hashtable prints the number of sessions linked to each entry in the hashtable used to look up outgoing packet on the given interface. This command is provided for debug purposes only.

Example:

```
bd2000> nat hashtable ethernet
```

```
# Linked sessions
```

```
0 1
```

```
1 0
```

```
2 1
```

```
3 2
```

See also:

OAM Console Commands

For all commands, these arguments (where applicable) take the following meanings:

port Name of an ATM port, or numerical port number.

vpi Numerical Virtual Path Identifier, or 'any'.

vci Numerical Virtual Circuit Identifier, or 'any'.

flags One or both of **f4** **f5** and one or both of **etoe** **segment** in any order. Some commands have restrictions on the flags imposed by the OAM module – these are documented in [1].

For F4 streams, the **vci** argument must be 'any'.

Most argument checking is done by the OAM module, not OAMCLI – the arguments are passed straight through to OAMLIB.

ccactivate

Syntax:

```
ccactivate <port> <vpi> <vci> <flags> <tx|rx>
```

Description:

Send a continuity checking activate request on the specified OAM flow. ne or both of **tx** and **rx** should be specified.

Example:

```
0:20:2b:0:52:c0 oamcli> ccactivate a1 0 any f4 etoe tx
```

See also:

```
ccdeactivate
```

ccclear

Syntax:

ccclear <port> <vpi> <vci> <flags>

Description:

Disable reception of activate requests on the specified OAM flow.

Example:

```
0:20:2b:0:52:c0 oamcli> ccclear a1 0 22 f5 etoe
```

See also:

ccset.

ccdeactivate

Syntax:

ccdeactivate <port> <vpi> <vci> <flags> <tx|rx>

Description:

Send a continuity checking deactivate request on the specified OAM flow. One or both of tx and rx should be specified.

Example:

```
0:20:2b:0:52:c0 oamcli> ccdeactivate a1 0 any f4 etoe tx rx
```

See also:

ccactivate

5.4 ccglobalaccept

Syntax:

ccglobalaccept

Description:

Toggles behaviour on reception of OAM_CC_ACTIVATE_REQ messages. By default, OAMCLI will call oamlib_CCAccept on the message and reply to it. This command will toggle the behaviour to calling oamlib_CCRreject before replying to it.

Example:

```
0:20:2b:0:52:c0 oamcli> ccglobalaccept
```

5.5 ccset

Syntax:

ccset <port> <vpi> <vci> <flags>

Description:

Enable reception of activate requests on the specified OAM flow.

Example:

```
0:20:2b:0:52:c0 oamcli> ccset a1 0 22 f5 etoe
```

See also:

ccclear . Specification for OAMCLI Module DO-007640-PS
©1999 Virata Limited 11

5.6 ccsetauto

Syntax:

ccsetauto <port> <vpi> <vci> <flags>

Description:

Enable automatic acceptance of received activate requests on the specified OAM flow.

Example:

```
0:20:2b:0:52:c0 oamcli> ccsetauto a1 0 any f4 segment
```

See also:

ccclearauto

5.7 ccstart

Syntax:

ccstart <port> <vpi> <vci> <flags> <tx|rx>

Description:

Start transmission and/or reception of continuity checking cells on the specified OAM flow. One or both of tx and rx should be specified.

Example:

```
0:20:2b:0:52:c0 oamcli> ccstart a1 0 any f4 segment rx
```

See also:

ccstop

5.8 ccstop

Syntax:

ccstop <port> <vpi> <vci> <flags> <tx|rx>

Description:

Stop transmission and/or reception of continuity checking cells on the specified OAM flow. One or both of tx and rx should be specified.

Example:

0:20:2b:0:52:c0 oamcli> **ccstop a1 0 any f4 segment rx**

See also:

ccstart . Specification for OAMCLI Module DO-007640-PS
©1999 Virata Limited 12

5.9 config

Syntax:

config

Description:

Display a message saying there is no storable configuration. This command is present only to comply with the console guidelines.

Example:

0:20:2b:0:52:c0 oamcli> **config**
This module has no storable configuration

5.10 faultbegin

Syntax:

faultbegin <port> <vpi> <vci> <flags>

Description:

Declare a fault state on the specified OAM flow.

Example:

0:20:2b:0:52:c0 oamcli> **faultbegin a1 0 19 f5 segment**

See also:

faultend

5.11 faultclear

Syntax:

faultclear <port> <vpi> <vci> <flags>

Description:

Turn fault management off for the specified OAM flow.

Example:

0:20:2b:0:52:c0 oamcli> **faultclear a1 0 19 f5 segment**

See also:

faultset

5.12 faultend

Syntax:

faultend <port> <vpi> <vci> <flags>

Description:

Clear a fault state on the specified OAM flow.

Example:

0:20:2b:0:52:c0 oamcli> **faultend a1 0 19 f5 segment**

See also:

faultbegin . Specification for OAMCLI Module DO-007640-PS
©1999 Virata Limited 13

5.13 faultset

Syntax:

faultset <port> <vpi> <vci> <flags>

Description:

Turn fault management on for the specified OAM flow.

Example:

0:20:2b:0:52:c0 oamcli> **faultset a1 0 19 f5 segment**

See also:

faultclear

5.14 faultwatch

Syntax:

faultwatch <port> <vpi> <vci> <flags>

Description:

Turn fault monitoring on for the specified OAM flow.

Example:

0:20:2b:0:52:c0 oamcli> **faultwatch a1 0 any f4 etoe**

See also:

faultunwatch

5.15 faultunwatch

Syntax:

faultunwatch <port> <vpi> <vci> <flags>

Description:

Turn fault monitoring off for the specified OAM flow.

Example:

```
0:20:2b:0:52:c0 oamcli> faultunwatch a1 0 any f4 etoe
```

See also:

faultwatch

5.16 getversion

Syntax:

```
getversion
```

Description:

Display the current version of the OAM module by calling into OAMLIB.

Example:

```
0:20:2b:0:52:c0 oamcli> getversion
oam Version 1.01
```

See also:

version . Specification for OAMCLI Module DO-007640-PS
©1999 Virata Limited 14

5.17 help

Syntax:

```
help [all | <command>]
```

Description:

Displays help about available console commands.

Example:

```
0:20:2b:0:52:c0 oamcli> help
commands are:
config version getversion inf
list segmentset segmentclear faultset
faultclear faultwatch faultunwatch faultbegin
faultend ccset cclear ccsetauto
ccclearauto ccactivate ccdeactivate ccstart
ccstop loopset loopclear loop
loopstats
'.' repeats the last command
Type 'help all' or 'help <command>' for more details
```

5.18 inf

Syntax:

```
inf <port> <vpi> <vci>
```

Description:

Display information about the specified VC/VP. Note that no flags are necessary – information about all flows matching the specified port, vpi and vci are displayed.

Example:

```
0:20:2b:0:52:c0 oamcli> inf a1 0 any
port 0 vpi 0 vci any settings 0x4000041
F4-ETOE F4-SEGM F5-ETOE F5-SEGM
loopback y
cc y
cc auto
fault
fault non-intrusive
segment sinkpoint y
F4 etoe llid: ff.ff.ff.ff.ff.ff.ff.ff.ff.ff.ff.ff.ff.ff.ff.ff.
```

See also:

list . Specification for OAMCLI Module DO-007640-PS
©1999 Virata Limited 15

5.19 list

Syntax:

```
list
```

Description:

List information about all flows that have any OAM settings.

Example:

```
0:20:2b:0:52:c0 oamcli> list
port 0 vpi 0 vci -1 settings 0x4000041 F4_ETOE F4_SEGM
port 0 vpi 0 vci 22 settings 0x2 F5_ETOE
```

See also:

inf

5.20 loopclear

Syntax:

loopclear <port> <vpi> <vci> <flags>

Description:

Disable loopback for the specified OAM flow.

Example:

0:20:2b:0:52:c0 oamcli> loopclear a1 0 any f4 etoe

See also:

loopset loopstats

5.21 loop

Syntax:

loop <port> <vpi> <vci> <flags> [<llid>]

Description:

Send out a loopback cell for the specified OAM flow, with an optional loopback location identifier. If no LLID is specified, it defaults to all 1's. Any llid that is specified is padded with zeros to the full 16 octets.

The command returns immediately – the response (or lack of response) is indicated by the arrival

of a message from the OAM process, which appears in the event log.

Example:

0:20:2b:0:52:c0 oamcli> loop a1 0 any f4 etoe

0:20:2b:0:52:c0 oamcli> loop a1 0 22 f5 segment 11.12.13. Specification for OAMCLI Module DO-007640-PS

©1999 Virata Limited 16

5.22 loopset

Syntax:

loopset <port> <vpi> <vci> <flags> [<llid>]

Description:

Enable loopback for the specified OAM flow, with an optional loopback location identifier. If no LLID is specified, it defaults to all 1's. Any llid that is specified is padded with zeros to the full 16 octets.

Example:

0:20:2b:0:52:c0 oamcli> loopset a1 0 any f4 etoe

0:20:2b:0:52:c0 oamcli> loopset a1 0 22 f5 segment 11.12.13

See also:

loopclear loopstats

5.23 loopstats

Syntax:

loopstats <port> <vpi> <vci> <flags>

Description:

Display loopback statistics for a specified flow. This is only valid if loopback is enabled for that flow.

Example:

0:20:2b:0:52:c0 oamcli> loopstats a1 0 any f4 etoe

Statistics for port 0 vpi 0 vci -1 flags = f4 etoe

Loopback requests received = 5

Loopback requests dropped = 1

See also:

loopset loopclear

5.24 pmactivate

Syntax:

pmactivate <port> <vpi> <vci> <flags> <txblk> <rxblk> <tx|rx>

Description:

Send a performance monitoring activate request on the specified OAM flow.

One or both of tx and rx should be specified.

txblk and rxblk are the transmit and receive block sizes respectively.

Example:

0:20:2b:0:52:c0 oamcli> pmactivate a1 0 any f4 etoe 128 128 tx

See also:

pmdeactivate. Specification for OAMCLI Module DO-007640-PS

©1999 Virata Limited 17

5.25 pmclear

Syntax:

pmclear <port> <vpi> <vci> <flags>

Description:

Disable reception of activate requests on the specified OAM flow.

Example:

```
0:20:2b:0:52:c0 oamcli> pmclear a1 0 22 f5 etoe
```

See also:

pmset

5.26 pmdeactivate

Syntax:

```
pmdeactivate <port> <vpi> <vci> <flags> <tx|rx>
```

Description:

Send a performance monitoring deactivate request on the specified OAM flow. One or both of `tx` and `rx` should be specified.

Example:

```
0:20:2b:0:52:c0 oamcli> pmdeactivate a1 0 any f4 etoe tx rx
```

See also:

pmactivate

5.27 pmglobalaccept

Syntax:

```
pmglobalaccept
```

Description:

Toggles behaviour on reception of OAM_PM_ACTIVATE_REQ messages. By default, OAMCLI will call `oamlib_PMAccept` on the message and reply to it. This command will toggle the behaviour to calling `oamlib_PMReject` before replying to it.

Example:

```
0:20:2b:0:52:c0 oamcli> pmglobalaccept
```

5.28 pmset

Syntax:

```
pmset <port> <vpi> <vci> <flags>
```

Description:

Enable reception of activate requests on the specified OAM flow.

Example:

```
0:20:2b:0:52:c0 oamcli> pmset a1 0 22 f5 etoe
```

See also:

pmclear . Specification for OAMCLI Module DO-007640-PS
©1999 Virata Limited 18

5.29 pmsetauto

Syntax:

```
pmsetauto <port> <vpi> <vci> <flags>
```

Description:

Enable automatic acceptance of received activate requests on the specified OAM flow.

Example:

```
0:20:2b:0:52:c0 oamcli> pmsetauto a1 0 any f4 segment
```

See also:

pmclearauto

5.30 pmstart

Syntax:

```
pmstart <port> <vpi> <vci> <flags> <txblk> <rxblk> <tx|rx>
```

Description:

Start transmission and/or reception of performance monitoring cells on the specified OAM flow. One or both of `tx` and `rx` should be specified.

`txblk` and `rxblk` are the transmit and receive block sizes respectively.

Example:

```
0:20:2b:0:52:c0 oamcli> pmstart a1 0 any f4 segment 256 256 rx
```

See also:

pmstop

5.31 pmstop

Syntax:

```
ccstop <port> <vpi> <vci> <flags> <tx|rx>
```

Description:

Stop transmission and/or reception of performance monitoring cells on the specified OAM flow. One or both of `tx` and `rx` should be specified.

Example:

```
0:20:2b:0:52:c0 oamcli> pmstop a1 0 any f4 segment rx
```

See also:

pmstart

5.32 segmentclear

Syntax:

segmentclear <port> <vpi> <vci> <flags>. Specification for OAMCLI Module DO-007640-PS
©1999 Virata Limited 19

Description:

Remove a segment sinkpoint from this location.

Example:

```
0:20:2b:0:52:c0 oamcli> segmentclear a1 0 any f4 segment
```

See also:

segmentset. Specification for OAMCLI Module DO-007640-PS

5.33 segmentset

Syntax:

segmentset <port> <vpi> <vci> <flags>

Description:

Set this location as a segment sinkpoint.

Example:

```
0:20:2b:0:52:c0 oamcli> segmentset a1 0 any f4 segment
```

See also:

segmentclear

5.34 version

Syntax:

version

Description:

Display the current version of the OAMCLI module.

Example:

```
0:20:2b:0:52:c0 oamcli> version
oamcli Version 1.00
```

See also:

getversion

PPP Console commands

Console commands should be prefixed with `ppp` in order to direct them to the **ppp** process.

Console object types

The **ppp** process presents its setup in terms of a number of distinct object types:

The upper limit on the number of each of these objects permitted in a system is configured using

the `'config resource'` console command.

The current state of each object is saved by `'config save'`.

5.1.1 Channels

The **ppp** process provides a number of PPP connection *channels*. A channel is a single PPP connection. Channels are numbered from 1. Many **ppp** console commands affect only a single channel. The command is prefixed with the channel number.

5.1.2 Users

A *user* is a user name and password. All users must have distinct names. The `user` console command controls these.

5.1.3 Tunnels

A *tunnel* is a PPTP or L2TP connection. Tunnels are numbered from 1. PPP channels must be associated with a tunnel to be involved in PPP tunnelling. The `tunnel` console command provides control of these.

5.1.4 Interfaces

An interface is an internal MAC (Ethernet) device. PPP channels must be associated with an interface to be involved with bridging or routing.

5.1.5 Interface 1 and Channel 1

Interface 1 has some special functions associated with it, allowing dynamic IP address assignment to be performed. Channel 1 is by default associated with Interface 1. These two should be used only for IP dial-out functions, and for this function should be attached to the router interface named `'ppp_device'`. The dial-out example in the following section makes this

clearer. These specialisations have been made to make the configuration of an IP dial-out simpler.

Console examples

5.2.1 Simple test

The simplest thing you can do to test ATMOS PPP, between two PPP channels in a single ATMOS system, is to create a PVC in the switch to which the test box is connected, between two VCIs (say 32 and 33 here) on the connected switch port. Type the following:

```
pvccreate a1 32 a1 33 (at the switch console, if it is a Virata Switch)
```

```
ppp event 5 (at the console of the PPP ATMOS system)
```

```
ppp 1 pvc 32
```

```
ppp 2 pvc 33
```

```
ppp 1 enable
```

```
ppp 2 enable (they should now swap packets and synchronise)
```

```
ppp 1 status
```

This should show that the two ends are connected. No data will be exchanged.

5.2.2 IP dial-out over PPP

To perform a dial-out over a PVC, operate as follows:

First set up a router device for PPP to use. No IP address should be specified, so that the device is created but not enabled. The device name 'ppp_device' should be used.

```
ip device add ppp_device ether //ppp/DEVICE=1
```

```
ip config save
```

```
ip restart (the system will reset)
```

```
ppp 1 pvc <whatever>
```

```
ppp 1 welogin <name> <password>
```

```
ppp 1 enable
```

If the configuration is saved at this point then the dial-in will be attempted automatically when the system is reset.

5.2.3 IP dial-in server setup

To create a system which can accept dial-in connections over PVCs, type the following. For a complex setup such as this it may be more convenient to create it on another system using a text editor, then TFTP the setup into the ATMOS system. Assume that 8 dial-in PVCs are being created, numbered 32..39. These will be created as channels 2..9. A single IP subnet will be created, attached to a port of the router via interface 3. The IP subnet 192.168.200.0 will be used, with channel n assigning address 192.168.200.n to the far end. The server interface will take address 192.168.200.99.

```
ip device add ppp_device3 ether //ppp/DEVICE=3 192.168.200.99
```

```
ip config save
```

```
ip restart
```

(The system restarts. Further IP setup may be needed, for instance to route the result to some other device such as the Ethernet port.)

```
ppp interface 3 localip 192.168.220.99
```

```
ppp 2 pvc 32 listen
```

```
ppp 2 interface 3
```

```
ppp 2 remoteip 192.168.200.2
```

```
ppp 2 enable
```

(and the corresponding setup for each of the channels 3..8 as well)

Clients can now dial in to this server, be allocated IP addresses and traffic will be sent to and from the router.

5.2.4 Remote Bridging

To create a system where two bridges are connected by a PVC, do the following at each end: In this example we attach interface 2 to the bridge in ATMOS (interface 1 is reserved for routed traffic).

```
bridge device add ppp/DEVICE=2 (attach interface 2 to the bridge)
```

```
config save (this only takes effect after a restart)
```

```
ip restart
```

(the system restarts)

```
ppp 1 pvc 32 mac
```

```
ppp 1 interface 2
```

```
ppp 1 enable
```

If required, multiple interfaces can be attached to the bridge of a single ATMOS system so that a single 'master' site is bridged to several satellites. Each incoming bridging PPP channel

should be attached to a distinct interface. Each interface must be independently attached to the bridge.

5.2.5 Tunneling

To create PPTP tunnel number 1 and connect it to PPP channel 2:

```
pptp bind 192.168.220.100
pptp 1 create listen
ppp 2 pvc 902
ppp 2 interface 0
ppp 2 tunnel 1 pptp out
ppp 2 enable
```

Console commands

The rest of this section details the individual console commands provided.

<channel> clear

Syntax:

```
<channel> clear
```

Description:

Clear all aspects of this channel back to their default settings. If there is an active connection it is torn down.

<channel> disable

Syntax:

```
<channel> disable
```

Description:

Clear the enable flag for a PPP channel. This is the default setting. Disabling does not remove other configured information about this channel. In the PPP state machine, this sets the PPP link to 'closed'. If it is already closed, there is no effect.

Configuration saving saves this information. By default all channels are disabled.

See also:

```
<channel> enable
```

<channel> discard

Syntax:

```
<channel> discard [<size>].
```

Description:

Discard is a PPP LCP packet type, which is like the Echo packet type but does not generate a return. This can be used for more careful tests of data transfer on the link, for instance at sizes near the negotiated MRU. This command sends an LCP Discard packet, of the specified size. If no size is given, a minimal sized packet is sent. Arrival of a Discard packet is logged locally as a level 2 event.

The link must be up and operational in order to do the discard test.

See also:

```
<channel> echo
```

<channel> echo

Syntax:

```
<channel> echo [<size>]
```

Description:

Echo is an LCP packet, which is used to test an established PPP link. It solicits a ping-like reply from the far end. This command sends an LCP Echo packet, of the specified size. If no size is

given, a minimal sized packet is sent. If a size greater than the remote Maximum Receive Unit size is specified, the value is reduced to the remote MRU before sending. The command waits for 1 second for a reply packet to arrive, and prints whether the reply arrived. If a reply arrives subsequent to this, it is logged as a level 2 event. The link must be up and operational in order to do the echo test.
See also the `discard` test.

<channel> echo every

Syntax:

```
<channel> echo every <seconds>
```

Description:

Echo is an LCP packet, which is used to test an established PPP link. It solicits a ping-like reply from the far end. This command sets a channel to confirm the continued presence of an open PPP connection by sending an LCP echo every few seconds, and requiring an echo reply. The number of seconds between echo requests is specified as a parameter. If 0 is specified, the function is disabled. Use the `info all` command to read the current state on a channel. Configuration saving saves this information. By default the function is disabled.

See also:

`echo` (manually initiated LCP echo)
`info all` (show current state).

<channel> enable

Syntax:

```
<channel> enable
```

Description:

Set the enable flag for a PPP channel. By default this is disabled. In the PPP state machine, this flag sets the PPP link to 'open'. If it is already open, there is no effect. Configuration saving saves this information. By default all channels are disabled.

See also:

`disable` (reverse the effect)
`status`

<channel> event

Syntax:

```
<channel> event [<n>]
```

Description:

Read or set the overall trace output level.
Configuration saving does not save this value. The default event level is 1.
Event levels are:
1 only very serious errors reported (default)
2 definite protocol errors or very significant events reported
3 links going up/down are reported
4 every packet and significant state change is reported
5 every packet sent/received is disassembled, and hex dumped

<channel> hdlc

Syntax:

```
<channel> hdlc [1|0]
```

Description:

If 1, use an HDLC header on the front of transmitted packets and require one on received ones. This consists of two bytes, FF-03, and assists in interoperability with some other (non-

standard) implementations. If 0, disable this. Call with no argument to find the current setting. The default value is 0 (disabled).

Configuration saving saves this information.

If not set, and a packet is received with an HDLC header, the channel goes into a 'learned HDLC' mode and sends packets with the HDLC header. Thus, interoperation with HDLC-using equipment should not normally require any configuration. Learning occurs in this direction only. Setting `hdlc` to 0 clears this learned state. Configuration saving does not save the learned state.

<channel> info

Syntax:

```
<channel> info [all]
```

Description:

Provide information about the current settings of this channel. This includes all configured state, and also current protocol information. Specifying 'all' prints out more information.

`info` and `status` are synonyms.

<channel> interface

Syntax:

```
<channel> interface <n>
```

Description:

Logically associate the specified channel with the specified interface. Interface 1 is always the router port. It should be used for any PPP channel over which IPCP communication with the local system's IP router is desired. Other interfaces can be created for bridging. A single PPP channel can only be associated with a single interface, or a single tunnel. Use `info` to find the current setting.

Calling with `n=0` removes any association. This is the default state.

Configuration saving saves this information.

See also:

```
<channel> info  
<channel> tunnel <n>
```

<channel> lcpmaxconfigure

Syntax:

```
<channel> lcpmaxconfigure [<n>]
```

Description:

Set the Max-Configure parameter for LCP, as described in section 4.6 of RFC1661. This is the maximum number of Configure Requests that will be sent without reply, before assuming that the peer is unable to respond. Call with no argument to find the current setting.

The default value is 10.

Configuration saving saves this information.

<channel> lcpmaxfailure

Syntax:

```
<channel> lcpmaxfailure [<n>]
```

Description:

Set the Max-Failure parameter for LCP, as described in section 4.6 of RFC1661. This is the maximum number of consecutive Configure Naks that will be sent before assuming that parameter negotiation is not converging. Call with no argument to find the current setting.

The default value is 5.

Configuration saving saves this information.

<channel> lcpmaxterminate

Syntax:

```
<channel> lcpmaxterminate [<n>]
```

Description:

Set the Max-Terminate parameter for LCP, as described in section 4.6 of RFC1661. This is the maximum number of Terminate Requests that will be sent without reply, before assuming that the peer is unable to respond. Call with no argument to find the current setting.

The default value is 2.

Configuration saving saves this information.

<channel> llc

Syntax:

```
<channel> llc [1|0]
```

Description:

If 1, use an LLC header on the front of transmitted packets and require one on received ones. This consists of four bytes, FE-FE-03-CF, and is required for PPP Over AAL5 (RFC 2364 p4) when using LLC encapsulated PPP. If 0, disable this. Call with no argument to find the current setting.

The default value is 0 (disabled). Configuration saving saves this information. If not set, and a packet is received with an LLC header, the channel goes into a 'learned LLC' mode and sends packets with the LLC header. Thus, interoperation with LLC-using equipment should not normally require any configuration. Learning occurs in this direction only. Setting `hdlc` to 0 clears this learned state. Configuration saving does not save the learned state.

<channel> pvc

Syntax:

```
<channel> pvc [[<port>] <vpi>] <vci> [ip|mac] [listen]
```

```
<channel> pvc none
```

Description:

Attach an ATM PVC to the given PPP channel. The port can be specified (only for a multi-port device), and the VPI (default is 0), and the VCI. The allowable range of port, VPI, VCI depends on the atm driver. Normal limits are 0 only for port, 0 only for VPI, 1..1023 for VCI. If a single argument `none` is supplied, any current connection is torn down. This is equivalent to `svc none` on the channel. In the PPP state machine, providing a link of this form causes the link to be 'up'. Note that `enable` must also be used, to allow the link to become operational. The `ip` or `mac` indicates which form of data is transported over the connection: one of IP data (controlled by the IPCP protocol), or MAC data (for BCP). If neither is provided, `ip` is assumed. If the channel is not linked to an interface, and the channel is for IP data, the channel is linked to interface 1. If the channel is not linked to an interface, and the channel is for MAC data, the channel is linked to interface 2. Providing a PVC setting unsets any SVC setting. See the `svc` command. It is possible for a PVC to become 'down' in the PPP state machine even though the PVC is still there, for instance due to an authentication failure. If in this state, an incoming packet will cause the PPP state machine to go 'up'. If `listen` is specified then this is the server end of a PVC. It will not send out PPP Configure Requests until it first receives a packet over the PVC. When a connection is torn down it goes returns to this state. Use the `info` command to read this information.

Configuration saving saves this information. By default a channel has no connection information.

Example:

```
ppp 3 pvc 3 32 set channel 3 to be (VPI=3, VCI=32)
```

```
ppp 4 pvc read PVC settings for channel 4
```

```
ppp 5 pvc 0 remove any PVC settings from channel 5
```

<channel> qos

Syntax:

```
<channel> qos [cbr|ubr] [pcr <pcr-tx> [<pcr-rx>]]
```

Description:

Specify that the VC for a PPP channel should be Constant Bit Rate or Unspecified Bit Rate, and (optionally for UBR) give a Peak Cell Rate for the connection. If two values are specified then they are the transmit and receive PCRs respectively. If called while not attached to a VC then the settings are saved for use when a VC is created. If the channel is already attached to a VC then it is closed, and re-opened with the new values. If it cannot be reopened, it remains closed.

Configuration saving saves this information. By default channels are established UBR.

Example:

```
ppp 3 qos cbr pcr 10000 set channel 3 to be CBR limited at  
10000 cells/sec
```

<channel> remoteip

Syntax:

```
<channel> remoteip [<ipaddress>]
```

Description:

If a PPP link is established using IPCP, this call causes the channel to provide the given IP address to the remote end of the connection. PPP will refuse to complete the connection if the other end will not accept this. This is normally used for channels on which the remote party dials in, to allocate the IP address to that remote party. Call with no argument to find the current setting. Call with 0.0.0.0 to remove any setting. This is the default state.

Configuration saving saves this information.

See also:

```
interface <n> localip
```

<channel> svc

Syntax:

```
<channel> svc listen [ip|mac]  
<channel> svc addr <addr> [ip|mac]  
<channel> svc none
```

Description:

Specify that the VC for a PPP channel should be an SVC (i.e. created by signalling). This can either be by listening for an incoming call, or by making an outgoing call to a specified ATM address. The outgoing call or listen occurs immediately. If the call fails it will be retried after a few seconds. In the PPP state machine, providing a connection of this form causes the channel to be 'up' or 'down'. Note that `enable` must also be used, to allow the link to become operational. Outgoing and incoming UNI signalling calls are identified by a BLLI value that identifies PPP. (Aside: A BLLI of length 3 bytes is used, hex values 6B, 78, C0.) If the channel is already attached to an SVC or PVC then it is closed, and re-opened with the new settings. If it cannot, it remains closed. If a single argument `none` is supplied, any current connection is torn down. This is equivalent to `pvc none` on the channel. The `ip` or `mac` indicates which form of data is transported over the connection: one of IP data (controlled by the IPCP protocol), or MAC data (for BCP). If neither is provided, `ip` is assumed. Providing an SVC setting unsets any PVC setting.

Configuration saving saves this information. By default a channel has no connection information.

Example:

```
ppp 3 svc 47.00.83.01.03.00.00.00.00.00.00.00.00.00.20.2b.00.03.0b.00  
ppp 4 svc listen (listen for incoming call)  
ppp 7 svc none (tear down connection, remove setting)
```

<channel> theylogin

Syntax:

```
<channel> theylogin pap|chap|none
```

Description:

This command describes how we require the far end to log in on this channel. Requiring the other end to log in most frequently happens when they dial us (rather than the other way round), so this is likely to be one of several channels which are set using `svc listen`. Because of this, exact names and passwords are not attached to individual channels but are matched to particular users, as defined using the `user` command. This command specifies that when using this channel, the user must log on using the specified protocol, and that they must provide any name/password combination which has been defined for that protocol, using the `user` command. To remove this information on a channel, call `theylogin` with a single argument of `none`.

Configuration saving saves this information. By default no login is required.

<channel> tunnel <n> <tunnel protocol> <dial direction>

Syntax:

```
<channel> tunnel <n> <tunnel protocol> <dial direction>
```

Description:

Logically associate the specified channel with the specified tunnel. A single PPP channel can only be associated with a single interface, or a single tunnel. Use `info` to find the current setting.

Calling with `n=0` removes any association. This is the default state. Configuration saving saves this information. The possible tunnel protocols are: `pptp` and `l2tp`. The dial directions may be: `in` or `out` for dial-in or dial-out respectively.

Example:

```
ppp 3 tunnel 1 pptp out
```

See also:

```
<channel> info  
<channel> interface <n>.
```

<channel> welogin

Syntax:

```
<channel> welogin <name> <password> [pap|chap]  
<channel> welogin none
```

Description:

This command describes how we should log in to the far end when a connection is established. A name and password are supplied, and whether these should be used with the PAP or CHAP authentication protocol. CHAP is the default. To remove this information on a channel, call `welogin` with a single argument of `none`. If `chap` is specified, we will also log in using `pap` if the other end prefers this. If `pap` is specified we will only log in using `pap`.

Configuration saving saves this information. By default no login is performed.

bcp

Syntax:

```
bcp stp|nostp
```

Description:

This command describes parameters for BCP, the Bridge Control Protocol, which is used to transport MAC (Ethernet) packets over the PPP link. See the protocol conformance section of this spec for BCP option settings which are not controllable. If `stp` is specified, the Spanning Tree Protocol is in use by the Bridges, to control bridge loops. In this case STP frames should be carried over any links using BCP. If `nostp` is specified, STP frames should not be carried. Configuration saving saves this information. By default STP is not supported.

interface <n> localip

Syntax:

```
interface <n> localip <address>
```

Description:

This command describes parameters for IPCP, the IP Control Protocol, when providing the server end of an IPCP connection. The server knows its own IP address (and may allocate an IP address to the remote end). This command tells the PPP process, for a particular interface, the local IP address to be associated with the local end. For interface 1, this should be the same IP address as possessed by the device `ppp_device` in the IP stack. See the IP dial-in server console example, at the start of this section. If PPP channels are now associated with this interface, remote users can dial in to those channels and will be connected to the IP stack. They can be allocated IP addresses, see the command

`<channel> remoteip`. Call with 0.0.0.0 to remove any IP address setting. This is the default state.

Configuration saving saves this information.

See also:

`<channel> remoteip`

interface <n> stats

Syntax:

```
interface <n> stats
```

Description:

The interface is regarded by the operating system as an Ethernet-like device which can be attached to the bridge or router, like other Ethernet devices in ATMOS. It also provides an ifEntry to SNMP providing basic information about traffic through the interface. This command shows the basic information about byte and packet traffic through the interface, in SNMP terms.

user

Syntax:

```
user add <name> [pwd <passwd> [pap|chap]]
user [<name>]
user delete <name>|all
```

Description:

This command stores information about a particular login name/password combination. This is referred to as a 'user', regardless of whether it represents an individual. When `user` is called on its own, information about all existing users is listed. When `user <name>` is called with no further arguments, details of that user alone are printed. Passwords are not shown. Use `user delete` to delete an individual user by name, or to delete all users. Use `user add <name>` to create a new user or update an existing one. The password is stored, and the authentication protocol which must be used for this user. If a user is deleted or changed, existing sessions are not affected.

Configuration saving saves this information.

version

Syntax:

```
version
```

Description:

Provide the version number for the source of the **ppp** process.

Spanning Tree Console Commands

Commands are directed to the spanning tree “process” by sending commands to the **bridge** process and preceding such commands with the keyword `spanning`. Depending on the console interface provided, it may be necessary to precede `spanning` with `bridge`, for example:

`bridge spanning status`: Display the status of the spanning tree.

disable

Syntax:

`disable`

Description:

Disables the spanning tree process. When spanning tree operation is disabled, the bridge operates in transparent mode and all bridge ports are set to the forwarding state. The `status` command reports the enabled state of the spanning tree process.

Configuration saving saves this information. By default, spanning tree operation is enabled.

Example:

`spanning disable` Disables spanning tree operation

See also:

`enable`
`status`

enable

Syntax:

`enable`

Description:

Enables the spanning tree process. When spanning tree operation is enabled, the state of the bridge’s ports is controlled by the spanning tree process. The `status` command reports the enabled state of the spanning tree process.

Configuration saving saves this information. By default, spanning tree operation is enabled.

Example:

`spanning enable` Enables spanning tree operation

See also:

`disable`
`status`

forwarddelay

Syntax:

`forwarddelay [<time>]`

Description:

Reads or sets the time in seconds, in which the bridge remains in the listening or learning states, and is used when the bridge is or is attempting to become the root bridge. The forward delay time may be any value between 4 and 30 but it is also constrained by the maximum age and hello times. The forward delay time may also be changed by SNMP command. The maxage, hellotime and forwarddelay times are constrained as follows:

$2 \times (\text{forwarddelay} - 1) \geq \text{maxage}$.

$\text{maxage} \geq 2 \times (\text{hellotime} + 1)$

Configuration saving saves this information. By default the forward delay time is set to 15 seconds.

Example:

`forwarddelay 10` Sets the forwarding delay to 10 seconds.

`forwarddelay` Reports the current forward delay time.

See also:

`hellotime`
`maxage`

hellotime

Syntax:

```
hellotime [<time>]
```

Description:

Reads or sets the time in seconds, after which the spanning tree process sends notification of topology changes to the root bridge, and is used when the bridge is or is attempting to become the root bridge. The hello time may be any value between 1 and 10 and is also constrained by the forwarddelay and maxage times. The hello time may also be changed by SNMP command. Configuration saving saves this information. By default the hello time is set to 2 seconds.

Example:

```
hellotime 5 Sets the hello time to 5 seconds
```

See also:

```
forwarddelay  
maxage
```

info

Syntax:

```
info
```

Description:

Displays the version number of the spanning tree implementation.

Example:

```
info
```

See also:

```
version
```

maxage

Syntax:

```
maxage [<time>]
```

Description:

Reads or sets the maximum age of received spanning tree protocol information before it is discarded, and is used when the bridge is or is attempting to become the root bridge. The maxage time may be any value between 6 and 40 and is also constrained by the forwarddelay and hellotime times. The maxage time may also be changed by SNMP command. Configuration saving saves this information. By default the maxage time is set to 20 seconds.

Example:

```
maxage 6 Sets the maxage time to 6 seconds.
```

See also:

```
forwarddelay  
hellotime
```

port <number>

The port commands, described in subsequent sections, control the configuration of the bridge's ports so far as the operation of the spanning tree protocol is concerned. Ports are numbered from 1. Every port on the bridge may be specified by typing `all` instead of a port number.

port <number> disable

Syntax:

```
port <number> disable
```

Description:

Allows a port to be disabled. The state of a port may also be changed by SNMP command. A port which is enabled will not take part in the operation of the spanning tree protocol. Configuration saving saves this information. By default ports are enabled.

Example:

```
port 2 disable Disables port 2 on the bridge.
```

See also:

```
port <number> enable
status
```

port <number> enable

Syntax:

```
port <number> enable
```

Description:

Allows a port to be enabled. The state of a port may also be changed by SNMP command. A port which is enabled will take part in the operation of the spanning tree protocol. If enabled, the physical port may be "enabled" or "disabled" as demanded by the operation of the protocol.

Configuration saving saves this information. By default ports are enabled.

Example:

```
port 1 enable Enables port 1 on the bridge.
```

See also:

```
port <number> enable
status
```

port <number> pathcost

Syntax:

```
port <number> pathcost [<cost>]
```

Description:

Reads or sets the cost of using this port. The cost may be any number between 1 and 65535. The cost of the port is used when deciding which is the best path to the root bridge. The cost of a port may also be changed by SNMP command.

Configuration saving saves this information. By default a cost of 10 is assigned to a port.

Example:

```
port 2 pathcost Displays the path cost for port 2 on the bridge
```

See also:

```
port <number> priority
```

port <number> priority

Syntax:

```
port <number> priority [<portpriority>]
```

Description:

Reads or sets the priority of the port. The priority may be any value between 0 and 255. The priority is used in conjunction with the pathcost to determine the best root to the root bridge. The higher the priority number, the less significant, in protocol terms, the port. The port priority may also be changed by SNMP command.

Configuration saving saves this information. By default a port has a priority of 128.

Example:

```
port 1 priority Displays the priority for port 1 on the bridge
```

See also:

```
priority
port <number> pathcost
```

priority

Syntax:

```
priority [<bridgepriority>]
```

Description:

Reads or sets the priority of the bridge. The priority may be any value in the range 0 to 65535. The higher the priority number, the less significant, in protocol terms, the bridge. Where two bridges have the same priority, their MAC address is compared and the smaller MAC address is treated as more significant. The priority of the bridge may be changed by SNMP command. Configuration saving saves this information. By default the bridge is assigned a priority of 32768.

Example:

```
priority 4000 Sets the bridge priority to 4000.
```

See also:

```
port <number> priority
```

status**Syntax:**

```
status
```

Description:

Reports the status of the spanning tree. If spanning tree operation is disabled, a message is printed to that effect and no other information is displayed. When spanning tree operation is enabled, the following information is displayed:

- The identifier of the bridge.
- The identifier of the root bridge.
- The root port for this bridge.
- The root path cost: how far the bridge is from the root.
- The various spanning tree time values as defined by the current root bridge:
- The maximum age of spanning tree information before it is discarded: max age time.
- The amount of time between configuration protocol packets: hello time.
- The amount of time delay when ports are changing state: forward delay time.
- For each port:
 - The identifier of the designated bridge
 - The identifier of the designated port for the designated bridge
 - The identifier of the designated root bridge

Example:

```
status
```

See also:

version**Syntax:**

```
version
```

Description:

Displays the version number of the spanning tree implementation.

Example:

```
version
```

See also:

```
Info
```

©1998 Virata Limited 17

SNMP Console Commands

The ATMOS SNMP agent does not itself provide any console commands. However, the **ip** process manages the list of community names (which are also used for other forms of access control by other ATMOS processes, such as TFTP and telnet passwords) and the list of trap destinations. The standard ATMOS console allows the commands to manage these to be issued starting with "snmp", as if they were issued to the SNMP process; it automatically translates this to "ip snmp".

access**Syntax:**

```
access [read | write] <community> [<IP addr>]
access delete <community> [<IP addr>]
access flush
access list
```

Description:

The "read" and "write" options configure a community name that can be used for read-only or read-write access, respectively. If an IP address is specified, then the community name is valid only for SNMP requests issued from that IP address. (It should be noted that this can be rather weak security, since it is possible for the source address of IP packets to be forged.) The same

community name can be configured several times with different IP addresses, to allow access with the same community name from a number of different machines. The number of access records (community names paired with optional IP addresses) that can be configured is limited only by available memory.

The "delete" option deletes an access record. The IP address must match exactly; if it is not specified, only a matching access record that has no IP address will be deleted. The "flush" option deletes all access records. The "list" option lists the access records. Configuration saving saves the access records.

By default, if there are no access records in the snmpinit file, no SNMP management is allowed. (However, ATMOS systems created by Virata often have a default snmpinit file that contains the permissions in the first example below: read-only access with community "public" and read-writeaccess with community "password".)

Example:

```
mymachine> snmp access list
access read public
access write password
mymachine> snmp access write xyzzy 192.168.4.73
mymachine> snmp access delete password
mymachine> snmp access list
access read public
access write xyzzy 192.168.4.73.
```

config

Syntax:

```
config [save]
```

Description:

Displays the configuration (as from "access list" and "trap list" together), or saves it to flash memory.

Example:

```
mymachine> snmp config
access read public
access write xyzzy 192.168.4.73
trap add public 192.168.4.73 162
```

help

Syntax:

```
help
help all
help <cmd>
```

Description:

Displays a brief list of "snmp" console commands, a detailed list, or detailed help on just one command.

Example:

```
mymachine> snmp help config
config syntax:
config [save] - display config or save to flash
```

trap

Syntax:

```
trap add <community> <IP addr> [<port>]
trap delete <community> <IP addr> [<port>]
trap flush
trap list
```

Description:

Manipulates the list of destinations to which SNMP traps will be sent. The default UDP port to send traps to is 162, but it may be overridden by specifying <port>.

Configuration saving saves the list of trap destinations.

Example:

```
mymachine> snmp trap flush
mymachine> snmp trap add public 192.168.4.73
mymachine> snmp trap add public 192.168.4.74 999
mymachine> snmp trap list
trap add public 192.168.4.73 162
trap add public 192.168.4.74 999
```

TFTP Console Commands

The TFTP process provides a number of console commands. Some commands are available whatever the build configuration, whilst some commands are available only if client mode operation is enabled.

To aid understanding, an indication is made where a command is client mode only.

connect

Syntax:

```
connect <node_name> || <ipaddr>
```

Scope:

Client mode only.

Description:

The 'connect' command is used to specify the remote host name or IP address that will be used in subsequent client mode transfers. Either a host name may be entered, searched for in the 'ipaddresses' configuration file, or an IP address in the form 'abc.def.ghi.jkl'. If the host name is not recognised or the IP address does not convert correctly an error is signalled.

The non-appearance of an error message after the command *does not* signify that the remote host is accessible, only that the syntax of the command was appropriate. This command is required before a client mode user first attempts to 'put' or 'get' a file, but need not be issued again unless its desired to change the remote machine name or address.

Example:

```
connect 192.168.200.10
```

See also:

put, get

get

Syntax:

```
get <remote_file> [local_file]
```

Scope:

Client mode only.

Description:

The 'get' command requests TFTP to retrieve a file from the remote host previously specified using the 'connect' command. Only files that fit within the file storage area within the session data (currently 8K) can be retrieved. This means that it not possible to initiate a software update from the client.

By default the file is named locally as the remote filename but by specifying a second filename an implicit rename is performed.

Example:

```
get ipaddresses.
```

See also:

connect, put

help

Syntax:

```
help
```

Description:

The 'help' command displays the help text which lists the (commonly used) TFTP commands. The 'init' command is not listed in the help text. The trace command has a large number of

optional parameters and detail on this command may be displayed by typing 'trace help'. If the software build supports client mode operation, these commands will be displayed in the help text.

Example:

help

See also:

version, trace help

init

Syntax:

init

Description:

The 'init' command causes all sessions to be initialised to an idle state. This command can be used during testing but is not required in normal operation. The command does not appear in the help text.

Example:

init

list

Syntax:

list

Description:

The 'list' command displays the status of any active sessions. This command is primarily intended for use during debug.

Example:

List

put

Syntax:

put [local_file] <remote_file>.

Scope:

Client mode only.

Description:

The 'put' command requests TFTP to transmit a file to the remote host previously specified using the 'connect' command. By default the file is named remotely as the local filename but by specifying a second filename an implicit rename is performed.

Example:

put ipaddresses

See also:

connect, get

trace

Syntax:

trace <help> || <-*> || <+event1> <-event2>

Description:

The 'trace' command allows the user to examine the currently set trace types or add /subtract trace types. Trace help lists all the available tracing types. If the trace command is used with no parameters the currently set trace types are displayed.

Example:

trace +tmr_exp

version

Syntax:

version

Description:

The `version` command displays software version information about the process. The version number, which is displayed in the form `a.bc`, is defined in the module file as an integer `abc`.

Example:

version

See also:

Help