corelatus

Corelatus GTH API (November 2023)

Part number 10-0003

The latest version is available at https://www.corelatus.com/gth/api/

Contents

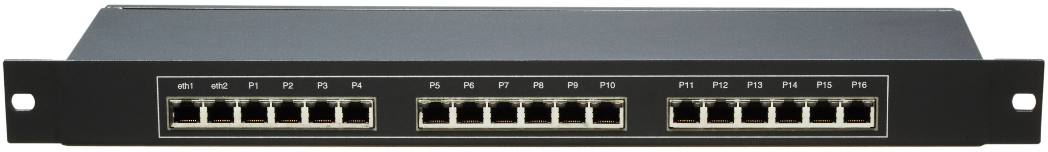
1 Introduction

Corelatus GTH is a family of rack-mounted systems for interfacing to the telecommunications network via E1/T1/J1 and SDH/SONET.

This note describes the commands used to control GTH hardware over ethernet. The commands can be used to connect to signalling and to work with audio on timeslots.

Impatient? Skip to the command reference starting on page 16. Looking at the commands and examples is the quickest way to see what's possible.

1.1 Hardware Models



E1/T1 Monitor 3.0, shown above, has two 10/100 Mbit/s ethernet interfaces (the two leftmost RJ45 ports) and 64 E1/T1/J1 receivers (in the other 16 RJ45 ports). It is mainly used for extracting signalling information from GSM and UMTS/3G radio networks.

E1/T1 Messenger 3.0 uses the same hardware as *E1/T1 Monitor 3.0*, but different firmware. This provides 16 E1/T1 transceivers, which is useful in applications where the hardware needs to transmit information *to* the telephone network as well as receiving it.

SDH Monitor 3.0 (the chassis shown above has three sub-modules in one chassis) has two 10/100 Mbit/s ethernet interfaces and two SFP slots per sub-module. The SFP slots accommodate SFP modules for 155 Mbit/s STM-1/OC-3 fiber. This probe is used for extracting signalling information carried on optical fiber.

1.2 Typical Application: A Voicemail System

Using the command interface on a GTH, you can build a scalable voicemail system suitable for connecting to E1 or T1 lines. The GTH handles:

- Recording timeslots for later playback, for instance as the message left in a mailbox.

- Switching timeslots among the GTH's E1/T1 interfaces, for instance when a call needs to be forwarded to an operator.
- Playing pre-recorded messages on timeslots.
- Detecting and generating DTMF signalling.
- Terminating ISDN LAPD signalling, for setting up calls.

1.3 Typical Application: Signalling Analysis

GTH is well suited for building permanently installed signalling analysis systems for a networks consisting of SDH/SONET optical links (which carry many E1, T1 or J1 lines) and electrical E1/T1/J1 links. An *E1/T1 Monitor 3.0* can connect to a G.772 monitor point to sniff and decode SS7-MTP2, ATM, LAPD, Frame Relay and CAS signalling. An *SDH Monitor 3.0* can be connected to an optical splitter to perform the same function on a fiber. The decoded signalling is then sent over Ethernet to an external system for analysis or logging.

1.4 Typical Application: Voice Quality Measurement

The GTH can forward a copy of selected conversations, for instance test conversations, to an external system for storage and analysis. The GTH handles:

- Sniffing MTP-2 or LAPD signalling. Your software can analyse the signalling to determine A and B numbers of interest and which timeslots are carrying the call.
- Recording selected timeslots.

1.5 Sample Code and Blog Examples

The Corelatus website has complete, working examples for decoding signalling, recording timeslot audio, setting up E1/T1/J1 and SDH/SONET interfaces and more. The examples are written in five languages (C, Java, Python, Erlang and Perl), at https://www.corelatus.com/gth/api/.

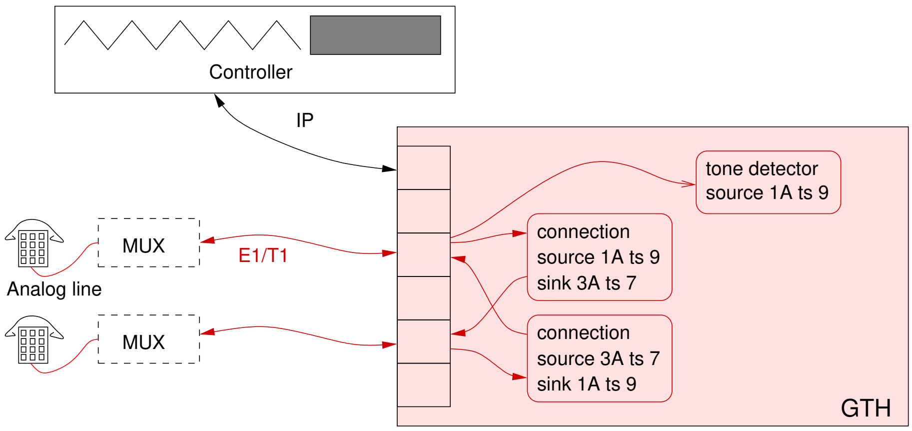
A comfortable way to get started with GTH is to hack some of those examples. Corelatus' blog, at https://www.corelatus.com/blog, includes explanations of many of the examples and screenshots showing them in action.

2 Principles and Terms

A *controller* is the system controlling, i.e. issuing commands to, the Corelatus hardware. It could be a Java system running on a 1U unix server in the same rack as the GTH, or it could be a laptop running a .net application on windows. GTH is language and OS agnostic.

Resources are parts of the GTH which always exist: the CPU, the logs and ethernet interfaces are all resources.

Jobs are transient parts of the GTH which are created in response to commands. DTMF detectors, message players, MTP-2 protocol handlers and connections between timeslots are jobs.



The figure above shows the API components used to set up a normal telephone call between two subscribers while detecting DTMF on one subscriber's line.

2.1 Sending Commands to the GTH

To send commands to the GTH, open a socket to the GTH's API on TCP port 2089. The factory default IP address is 172.16.1.10, netmask 255.255.0.0.

Once the socket is open, you can start sending commands. Every command starts with these three lines:

```
Content-type: text/xml
Content-length: 6
(a blank line)
```

The content-length indicates the number of octets (bytes) which follow the three header-lines. Each header line is case-sensitive and is always terminated with the two octets 0x0d (CR) and 0x0a (LF). Then, send the command itself.

Here's a complete command:

```
Content-type: text/xml
Content-length: 6
```

```
<nop/>
```

and here's the response from the GTH:

```
Content-type: text/xml
Content-length: 5

<ok/>
```

After the response, the GTH waits for the next command. I.e. the same socket is kept open indefinitely. Up to 100 API sockets can be open at the same time, the GTH will serve the API sockets concurrently.

The GTH guarantees three things for commands issued in *one* socket:

1. One command is executed at a time
2. Commands are executed in reception order
3. GTH sends a response for every command, in execution order.

2.2 Commands and Responses

The text protocol used on the port 2089 socket is a small XML language. If you know nothing about XML, don't worry: you can figure all you need to know from the examples in this manual and from the library you use to parse the XML. Here's a complete list of commands:

custom Modifies the firmware on the GTH module. This lets you put your own logo on the in-built webserver, for instance.

delete Remove a previously started job.

disable Disable a telecom interface (E1/T1/J1 or SDH/SONET).

enable Enable a telecom interface (E1/T1/J1 or SDH/SONET).

install Install firmware on the GTH module's flash memory.

map Extract an E1/T1/J1 carried by SDH/SONET.

new Create a job. There are jobs which perform signalling, for instance `atm_aal5_monitor` and `lapd_layer`, and jobs which manipulate audio, such as `connection` and `player`. Normally, jobs run until the `delete` command kills them, or the port 2089 API socket is terminated with a `bye` command.

nop The no-operation command.

query Check the status of jobs and resources in the GTH.

reset Restart the GTH.

set Change a resource's attributes.

takeover Transfer control of a job from one API socket to another.

unmap Remove an E1/T1/J1 resource created by the `map` command.

update Change parameters in a job.

zero Set counters and timers to zero.

Section 3 (p. 16) explains each command in detail. The GTH responds to each command. The possible responses are:

error There is a problem with the command.

job The successful reply to `new`, it carries the new job's ID.

ok The successful reply to commands such as `set` and `delete`.

resource The successful reply to `map`, it carries the mapped E1/T1/J1 resource name.

state The successful reply to `query` commands.

2.3 Data on the API Socket

In addition to the `text/xml` content-type, the port 2089 API socket can carry blocks of data of other content-types. Such data blocks always come immediately after the command and use the `content-length` header in exactly the same way as blocks carrying commands.

Commands which use data blocks include `install`, `query` and `custom`; the command reference for those commands notes which `content-type` to use.

2.4 Events

The GTH informs the controller of asynchronous events, i.e. events which didn't occur in direct response to a command. A controller **must** be prepared to receive an event even when there is no command pending. An event may even arrive between the moment a command is issued and when its response is sent.

By default, events concerning resources are broadcast to all controllers:

`alarm`	A resource has exceeded a limit, e.g. the temperature is outside the range $10 - 60^o$ Celsius; $10 - 70^o$ on SDH Monitor 3.0.
`alert`	An external system has failed in such a way that may cause problems for the GTH, for instance one of the two power inputs is no longer supplying power.
`info`	An informative message, for instance an indication of progress during firmware upgrade.
`l1_message`	The Layer 1 state of an E1/T1/J1 resource has changed.
`l2_socket_alert`	A TCP socket carrying L2 signalling to a controller has encountered a problem
`sdh_message`	The state of an SDH/SONET resource has changed.
`sfp_message`	The state of an SFP resource has changed.
`slip`	An E1/T1/J1 resource has 'slipped'.
`sync_message`	The E1/T1/J1 sync subsystem has changed state.

Events concerning jobs are only sent to the command socket which started the job, the *owner*:

`atm_message` An ATM channel changed state.

`backup` One or more jobs have been automatically transferred to this command socket.

`fatality` A job died.

`f_relay_message` A frame relay channel changed state.

`l2_alarm` A layer 2 signalling job has altered its alarm state.

`lapd_message` An LAPD timeslot changed state.

`level` A voice level detector has tripped.

`message_ended` A voice prompt has completed.

`mtp2_message` An MTP-2 timeslot changed state.

`tone` A DTMF tone was detected.

A simple policy to deal with events is to handle the expected ones and log the unexpected ones for manual attention.

If your system architecture uses more than a few concurrent API connections, use the `update` command to set the *broadcast_events* parameter of most of the controllers to `no` so that broadcast events are only sent on the API sockets which need them.

2.5 Voice

The GTH's connection to the telephone network is via either E1/T1 interfaces or SDH/SONET. A timeslot leaving the GTH, i.e. going to a subscriber, is called a *sink*. A timeslot entering the GTH is called a *source*.

GTH uses the `new connection` command (section 3.15) to switch timeslots; to connect a source to a sink. This is the building block used to let subscribers talk to each other.

GTH can play pre-recorded messages on a timeslot, using `new player` (section 3.22) and record a timeslot using `new recorder` (section 3.24). You can make IVR and voicemail systems using these commands. In both recording and playback, the audio data is streamed to or from the controller over a separate TCP socket.

GTH can detect DTMF tones from a subscriber's handset using `new tone_detector` (section 3.27). The DTMF tones could be a dialled number, or they could be the user navigating in a menu system.

2.6 Signalling

GTH can monitor (passively sniff) all common layer 2 signalling protocols used on E1/T1/J1 and SDH/SONET links: SS7 MTP-2, ATM AAL5 (used in SS7-HSSL), ATM AAL2, ATM AAL0, frame relay (used on the GSM Gb interface), ISDN LAPD and CAS. Monitoring is useful if you want to keep track of what is happening in a network: to create billing data, to detect when mobiles enter and leave a network, to detect fraud in real-time, to trace calls and to log signalling for later analysis.

GTH can terminate (actively participate in) ISDN LAPD, allowing an application with ISDN Layer 3 support to set up and tear down calls.

GTH has support for terminating Frame Relay, SS7 MTP-2 and ATM.

Signalling Setup

The steps for monitoring and terminating signalling are the same:

1. Establish a *listening* TCP socket on the controller.
2. Issue the XML command to start the signalling job, specifying the IP address and port number of the socket created in the previous step.
3. On the controller, *accept* the inbound connection.
4. The GTH sends the signalling packets to the newly established socket, completely separate from the socket carrying XML commands.

Signalling Socket Protocol

Each signalling packet arrives with a header which starts with the same fields, all big-endian, for all protocols:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
octet 0x00	Length															
octet 0x02	Tag															

The *Length* field indicates how many octets of data the rest of the packet contains. To find the next packet, read exactly `Length` octets.

The rest of the header is protocol-specific and defined separately for each protocol, for instance section 3.17 shows the complete header for frame relay.

Mixing Signalling Protocols

The GTH can decode different signalling protocols at the same time on different timeslots. For instance, the GTH can decode MTP-2 on one E1/T1/J1 while simultaneously decoding ATM AAL5 on another E1/T1/J1.

In some special cases, the GTH can decode multiple signalling protocols on the same input timeslots at the same time. ATM AAL0, AAL2 and AAL5 can be run concurrently on the same timeslots. The remaining protocols are mutually exclusive.

Signalling Socket Multiplexing

Normally, one signalling TCP socket carries packets from many signalling jobs. Multiple signalling jobs share one socket to the controller by all indicating the same IP address and port number.

Sharing one socket between multiple jobs is recommended because it reduces the per-channel TCP overhead and also because the number of outgoing sockets is limited.

Different channels on the same socket are distinguished by the tag value supplied by the controller in the `new` command.

Sending Signalling to Multiple Destinations

The signalling captured from one channel can be sent to multiple sockets. This is done by starting multiple jobs where all the parameters apart from the IP address and IP port are identical.

Adding destinations to a signalling channel does not consume additional signalling decoding capacity, however it does consume TCP output capacity.

Signalling Socket Auto-reconnect

If the controller closes the signalling socket, the GTH will first send an XML `l2_socket_alert` event and then, after a few seconds, attempt to reconnect the socket. In most cases, such a remote close will result in dropped signal units. The reconnect attempts continue periodically.

l2_socket_alert events

Whenever there is a problem with a signalling socket, the GTH sends an `l2_socket_alert` event. The possible reason codes are:

buffer_limit The output buffer for the socket just passed the half-full mark. This is an early warning that the receiving machine or IP network can't keep up with the rate of monitored packets.

buffer_overrun The output buffer for the socket filled completely. This means the receiving machine or IP network can't keep up and the GTH now has no choice but to discard packets.

remote_close The TCP socket closed unexpectedly.

Signalling Counters

Each signalling monitor has three load meters: *current_load, average_load* and *maximum_load*. The current load is computed over a period of 1s:

$$current\_load(t) = 100 \times \frac{M_t - M_{t-1s}}{B}$$

where M is the number of octets contained in correct signalling packets and B is the nominal channel bandwidth in octets-per-second. Typical values for B are 8000 for a 64 kbit/s MTP-2 link, 7000 for a 56 kbit/s ANSI MTP-2 link, 2000 for a 16 kbit/s subrate LAPD link and 240000 (for ATM-over-E1 links).

The average load is updated once every 30 seconds by default:

$$average\_load(t) = 100 \times \frac{M_t - M_{t-30s}}{30 \times B}$$

The *maximum_load* is the highest observed *current_load* value.

All counters and timers are 32 bits wide, so the octet counters on 2 Mbit/s frame relay link can wrap after a few hours.

2.7 Handling Errors

By being well separated from the controller, a system using a GTH is relatively easy to debug. There is no shared bus, so you don't have to worry about the GTH corrupting the controller's memory or vice versa. The control protocol runs over plain TCP, so no proprietary device drivers intrude on your system. The next sections look at the remaining problems.

Attempting the Impossible

The first class of errors occurs when an application sends a command the GTH can't carry out. For instance, an application might try enabling E1/T1/J1 interface which doesn't exist:

```
⇒ <enable name="pcm972"/>
⇐ <error reason="bad argument">invalid PCM</error>
```

These sorts of errors are fairly common during development. They're easy to deal with as long your application prints or logs the error message. The GTH also helps by logging incoming commands—you can browse the log on the in-built webserver on port 8888, commands appear in the "application log", which is under "OS". The logs can also be read automatically with a query command (section 3.31).

Returning to a Known State

After some error conditions, you want to return a GTH to a known starting state. The most complete way to do that is issue a `reset` command, which is equivalent to cycling power. The GTH completely resets on boot-up, there are only three things it remembers from before:

1. IP address settings (addresses, masks and default gateway).
2. Log files.
3. The firmware images. (see section 4.12).

A less brutal and less complete way to return to a known state is to close the port 2089 API socket from the controller by issuing a `bye`. The GTH will automatically `delete` all jobs started using *that* socket. Resources, such as E1/T1/J1 interfaces, are *not* returned to their default state—i.e. everything done via `set`, `enable` and `map` is remembered. Jobs started by other API sockets are not terminated either.

GTH Crashes

If all of the GTH's power inputs lose power, the GTH will use a small on-board power store to write a log entry before shutting down. When power is restored, the GTH knows the reason for the most recent shutdown:

```
⇒ <query><resource name="os"/></query>
⇐ <state>
     <resource name="os">
       <attribute name="last restart" value="Mon Feb 25 08:21:31 2008"/>
       <attribute name="restart cause" value="power failure"/>
     </resource>
   </state>
```

If the GTH crashes in response to some combination of XML commands, that is a problem Corelatus will track down and fix for you by studying log files and Ethernet traces.

Semantic Problems

The final class of errors is the most subtle: nothing crashes, but the results aren't as expected. The tools for solving that are the API manual, the log files, Ethernet traces (e.g. from tcpdump or wireshark) and Corelatus support at https://www.corelatus.com/contact.html

3 Commands

This chapter describes each GTH command in detail. Each section starts with a brief description of the syntax. A right arrow starts each possible variant of a command, for instance:

⇒ <custom **name**='inventory'|'board'|'http_server'|'os'>
<attribute **name**=string **value**=string/>
...
</custom>

The `custom` command in the example above has one parameter, *name*, and that parameter is shown in bold because it is mandatory. *Name* has four possible values, which is indicated by listing the values separated by a vertical bar.

The `custom` command always contains at least one `attribute`; the dots indicate that you can have more than one `attribute`. The `attribute` has two string parameters, both bold and therefore both mandatory. Parameter values, even integers, must always be enclosed in quotes.

A few commands, such as `update` (section 3.36) have more than one variant. Each variant starts with a new arrow ⇒.

The GTH's possible successful responses to each command are shown in the same format, but start with a left arrow:

⇐ <ok/>

Abnormal responses

When the GTH cannot execute a command, the GTH responds with `error`:

⇐ <error **reason**='bad argument'|'busy'|'conflict'|'failure'|'no such job'
|'not yet implemented'|'parse'|'refused'|'transport'>
human readable text
</error>

The *reason* is intended to be machine readable. The *human readable text* is not documented, it is intended to help humans debug problems, for instance by appearing in an application's log file.

Reason	Description
bad argument	One of the attributes in the command had an invalid value.
busy	The command attempted to start an action which was already in progress.
conflict	The command attempted to configure the GTH in a way which conflicts with the GTH's current configuration.
failure	The GTH had an internal problem. If you see this, report it to Corelatus, preferably with log files.
no such job	The command included a *job_id*, but the job_id does not correspond to any currently running job.
not yet implemented	The command attempted to use a feature which is not implemented in the currently installed firmware release.
parse	The command could not be parsed (a syntax error).
refused	The command could not be executed due to a constraint on the GTH, for example a capacity limit.
transport	The `content-type/content-length` headers preceding the command were invalid.

The `bad argument, parse` and `transport` errors can occur for all commands. The remaining errors are specifically noted in each command's description.

Machine Readable Protocol Definition

The XML protocol's syntax is defined in a machine-readable format by a pair of document type definitions (DTDs), one for commands *to* the GTH, another for responses *from* the GTH. The DTDs are available at:

https://www.corelatus.com/gth/api/

A validating XML parser such as *xmllint* can use the DTDs to find syntax errors in your XML commands. Section 6 shows how to do that.

3.1 bye

⇒ <bye/>

⇐ <ok/>

`bye` terminates an API connection gracefully. When the GTH receives a `bye`, the GTH deletes all jobs started by (owned by) the the API socket, responds with `ok` and closes the socket.

If the controller closes the API socket without a `bye`, the GTH performs the same steps as above, but also logs an error.

See also

`takeover` (3.34) can be used to prevent the GTH from automatically deleting jobs when an API connection is closed.

Example: Terminating an API connection gracefully

⇒ `<bye/>`

⇐ `<ok/>`

3.2 custom

⇒ <custom **name**='inventory'|'board'|'http_server'|'os'>
<attribute **name**=string **value**=string/>
...
</custom>

⇐ <ok/>

At boot time, the GTH runs a *start script* consisting of a sequence of `custom` commands separated by blank lines.

The *start script* is installed using the `install` command. To test the `custom` commands, they can also be issued, one at a time, in normal operation. The GTH takes the same action as it would if the commands were in a start script, though the effects only persist until the next reboot.

Each `custom` command affects a resource in exactly the same way as a `set` command, except that resources and attributes are limited to:

Resource Attribute	**Possible Values**	**Description**
inventory pcm_numbering	physical, sequential	`physical` numbering uses the E1/T1/J1 interface names pcm1A, pcm1B, pcm1C, pcm1D, ... pcm4D, where the number corresponds to the RJ45 port and the letter to the position within that port. `sequential` numbering uses the scheme pcm1, pcm2, pcm3, pcm4, pcm5 ... pcm16.
board PCM LED assignment	universal, sequential	E1/T1/J1 probe modules used in monitoring applications have four E1/T1/J1 receivers in each RJ45 connector, but only two LEDs. The default `universal` scheme maps PCM1A and PCM1C to the left LED and PCM1B and PCM1D to the right LED. The `sequential` scheme maps 1A and 1B to the left and 1C and 1D to the right.
http_server		All the attributes in the http_server resource may be customised, see section 4.15.
os remote login	enabled, disabled	
os API whitelist	IP addresses	A space-delimited list of IP addresses from which the API can be accessed.

See also

`install` (3.6) loads a start script onto a GTH.

Example: A start script

```
⇒     <custom name="inventory">
        <attribute name="pcm_numbering" value="sequential"/>
      </custom>

⇒     <custom name="http_server">
        <attribute name="pcm_overview" value="https://www.corelatus.com"/>
        <attribute name="pcm_zero_buttons" value="true"/>
      </custom>

⇒     <custom name="os">
        <attribute name="API whitelist" value="172.16.2.1 128.250.22.3"/>
        <attribute name="remote login" value="disabled"/>
      </custom>
```

This configuration script sets up the PCM numbering to sequential mode, customises several attributes in the on-board WWW server, disables remote SSH logins and restricts API control to two IP addresses.

Example: Altering PCM numbering with a custom command at runtime

```
⇒     <custom name="inventory">
        <attribute name="pcm_numbering" value="sequential"/>
      </custom>

⇐     <ok/>
```

Sending `custom` commands at runtime only has a temporary effect. If you want them to be permanent, put them in the start script.

3.3 delete

⇒ <delete **id**=string/>

⇐ <ok/>

`delete` removes the given job. The ID argument is the ID returned by `new`.

See also

`new` creates jobs which `delete` deletes. `query` can show a list of all the jobs currently running. `bye` terminates the API socket, which in turn terminates all the jobs started on (*owned* by) this API socket.

Error reasons

`no such job`: the given job_id does not correspond to any currently running job. The job either never existed, or it terminated before the `delete` was executed.

`refused`: the given job_id is this API connection. (Use `bye` to make an API connection terminate itself.)

Example: Deleting a job

Normally, the application running on the controller keeps track of which jobs it started with `new` and deletes them when they're no longer needed:

```
⇒    <delete id="cnxn182"/>
⇐    <ok/>
```

Example: Querying the schedule to find unwanted jobs

In some special cases, for instance recovering from a software error in the controller, it may be necessary to ask the GTH for a list of jobs and then to kill unwanted ones:

```
⇒    <query><resource name="schedule"/></query>
⇐    <state>
        <job id="cnxn182" owner="apic16"/>
        <job id="m2mo4" owner="apic15"/>
        <job id="m2mo5" owner="apic15"/>
        <job id="m2mo6" owner="apic15"/>
        <job id="at5m16" owner="apic15"/>
     </state>
⇒    <delete id="cnxn182"/>
⇐    <ok/>
```

3.4 disable

⇒ <disable **name**=string/>

⇐ <ok/>

Disable E1/T1/J1 and SDH/SONET interfaces.

When an SDH/SONET interface is disabled, any E1/T1/J1 interfaces mapped from that interface are automatically removed.

The LED next to each telecom interface is dark once the interface is disabled. All interfaces are disabled at startup.

Example: Disable an E1/T1/J1 interface

```
⇒     <disable name="pcm1A"/>
⇐     <ok/>
```

Example: Disable an SDH/SONET interface

```
⇒     <disable name="sdh2"/>
⇐     <ok/>
```

3.5 enable

⇒ <enable **name**=string>
<attribute **name**=string **value**=string/>
...
</enable>

⇐ <ok/>

Activates a telecom interface, either an E1/T1/J1 or an SDH/SONET interface, allowing other commands on the GTH to process the data carried on the interface.

E1/T1/J1 interfaces are described in 4.3.

SDH interfaces are described in 4.5.

SONET interfaces are described in 4.6.

The LED next to each telecom interface lights up when the interface is enabled.

See also

The `disable` command.

History

`enable` is available in release 37a and later. It replaces functionality previously offered by the `set` command.

Example: Enable an E1 interface

```
⇒    <enable name="pcm1A">
       <attribute name="framing" value="multiframe"/>
       <attribute name="monitoring" value="true"/>
     </enable>
⇐    <ok/>
```

In this example, we enabled 'monitoring', i.e. we're telling the hardware that the incoming signal is attenuated by a -20dB protected monitor point.

Example: Enable a T1 interface

```
⇒    <enable name="pcm1A">
       <attribute name="mode" value="T1"/>
       <attribute name="framing" value="extended superframe"/>
     </enable>
⇐    <ok/>
```

Example: Enable a J1 interface

```
⇒   <enable name="pcm1A">
      <attribute name="mode" value="J1"/>
      <attribute name="framing" value="extended superframe"/>
      <attribute name="monitoring" value="true"/>
    </enable>
⇐   <ok/>
```

Using J1 interfaces for transmit is not supported.

Example: Enable an SDH interface

```
⇒   <enable name="sdh1">
      <attribute name="STM" value="1"/>
      <attribute name="AU" value="4"/>
      <attribute name="TU" value="12"/>
    </enable>
⇐   <ok/>
```

All three attributes must be specified; they don't have default values.

3.6 install

⇒ <install **name**=string/>

⇐ <ok/>

`install` loads new software into the GTH's flash memory. New releases are released several times per year with new features and bug fixes. Releases are available to customers at https://www.corelatus.com/gth/releases/

An `install` command is always followed by a block with the content to be installed. The block's *content-type* is:

Name	Description	Content-type
failsafe_image	the failsafe firmware	binary/filesystem
system_image	the normal firmware	binary/filesystem
logo	the webserver logo	binary/file
start_script	a script run at boot time	binary/file

The failsafe image can only be upgraded if the system is currently running the system image, and vice versa.

`install` reports `ok` once the complete content has been received. When installing firmware images, the GTH also sends an event when all the actions associated with the firmware installation are complete:

⇐ <event>
<info **reason**=string>
<pcdata/>
</info>
</event>

The order of the `ok` and the `event` is not defined. An installation script should wait until it has seen *both*.

`install` can also be used to customise a software image *at install time*. The webserver logo, password file and a startup script can be customised:

Name	Description
logo	The logo displayed in the left margin on most of the web pages. It is a PNG image file.
start_script	A customisation file executed when the GTH starts. Section 3.2 describes the syntax.

See also

The `custom` command (section 3.2) and section 4.12.

Error reasons

`refused`: The name was not one of the names in the table above.

`refused`: The firmware image was the one currently being executed (the system image can only be upgraded while the system is running the failsafe image.)

`busy`: Another `install` command is currently executing.

`failure`: The release could not be installed. The on-board log files provide more information about why. One reason is that the release file is corrupt. Another is that the release file is intended for a different hardware generation.

Example: Checking which image is running

```
⇒    <query>
        <resource name="system_image"/>
     </query>

⇐    <state>
        <resource name="system_image">
           <attribute name="version" value="gth2_system_37a"/>
           <attribute name="locked" value="false"/>
           <attribute name="busy" value="true"/>
        </resource>
     </state>
```

Example: Booting the failsafe image

```
⇒    <set name="os">
        <attribute name="boot mode" value="failsafe"/>
     </set>

⇐    <ok/>

⇒    <reset><resource name="cpu"/></reset>

⇐    <ok/>
```

Example: Installing new system firmware

Once the GTH has booted in failsafe, unlock the firmware we want to change and then send the new firmware:

```
⇒    <set name="system_image">
        <attribute name="locked" value="false"/>
     </set>

⇐    <ok/>

⇒    <install name="system_image"/>

⇐    <ok/>
```

The firmware image is sent immediately following the `install` command, in a block with content type 'binary/filesystem'. When the GTH has completed all background tasks started by the install process, it sends an event:

⇐ `<event><info reason="install_done"/></event>`

Wait for that event before rebooting. Do not rely on the `event` coming after the `ok`—if there isn't any background work to do, the `event` can arrive first.

3.7 map

⇒ <map **target_type**='pcm_source'>
<sdh_source **name**=string/>
</map>

⇐ <resource **name**=string/>

Make an E1/T1/J1 carried on an SDH/SONET link available to commands which use a `pcm_source`. The command returns the name of the mapped E1/T1/J1 resource.

There are two ways to remove a mapping:

1. Send an `unmap` command.
2. Disable the SDH/SONET interface; this removes all mappings for that SDH/SONET interface.

The resource name chosen by the GTH for a particular E1/T1/J1 is always the same, as long as the SDH/SONET interface is configured with the same options. Or: sending two `map` commands with the same interface options results in the same response, even across a reboot.

Example: Mapping a T1 line carried on SONET

VT1.5 containers can carry T1 and J1 lines. Assuming the SDH interface has been enabled with SONET=true, OC=3 and VT=1.5, this command would map one of the T1/J1 lines:

```
⇒    <map target_type="pcm_source">
       <sdh_source name="sdh1:hop2:lop3_2"/>
     </map>
⇐    <resource name="pcm42"/>
```

Once a T1/J1 is mapped, `enable` starts layer 1 processing on the T1/J1, and `new` starts layer 2 processing on the data carried by the T1/J1.

Example: Mapping an E1/T1/J1 line carried on an STM-1

C-12 containers can carry E1, T1 and J1 lines. Assuming the SDH interface has been enabled with AU=4 and TU=12, this command would map one of the E1/T1/J1 lines:

```
⇒    <map target_type="pcm_source">
       <sdh_source name="sdh1:hop1_1:lop1_5_2"/>
     </map>
⇐    <resource name="pcm18"/>
```

3.8 new atm_aal0_monitor

⇒ <new>
<atm_aal0_monitor
cell='yes'
oam_cell='no'
corrupt_cell='no'
idle_cell='no'
scrambling='yes'
header_only='no'
load_limit='50'
buffer_limit='256000'
average_period='30'
tag='0'
ip_addr=int.int.int.int
ip_port=int>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
...
</atm_aal0_monitor>
</new>

⇒ <new>
<atm_aal0_monitor
cell='yes'
oam_cell='no'
corrupt_cell='no'
idle_cell='no'
scrambling='yes'
header_only='no'
load_limit='50'
buffer_limit='256000'
average_period='30'
tag='0'
ip_addr=int.int.int.int
ip_port=int>
<sdh_source **name**=string/>
</atm_aal0_monitor>
</new>

⇐ <job **id**=string/>

An `atm_aal0_monitor` extracts packets from the interface above the ATM layer.

On hardware with E1/T1/J1 interfaces, the ATM layer is always ATM-over-E1/T1/J1. E1 installations usually use timeslots 1–15 and 17–31 to make a 1920 kbit/s channel. T1 installations normally use 1–24 to make a 1536 kbit/s channel. The GTH supports both standard configurations, as well as arbitrary sets of timeslots.

Hardware with SDH/SONET interfaces supports additional modes: ATM can also be carried in a VC-4/STS-3c at 150 Mbit/s and at 48 Mbit/s ATM in a VC-3/STS-1.

cell, oam_cell, corrupt_cell: select whether or not normal, OAM and corrupt cells are delivered.

scrambling: ATM payload scrambling is on by default. ITU I.432.3 requires scrambling (x^{43} + 1) to be enabled for E1 links and leaves it optional for T1.

header_only: On 50 Mbit/s and 150 Mbit/s ATM links carried on SDH/SONET, this option prevents packet bodies from being sent. This is useful for gather statistics about which VPI/VCI pairs are in use on a link.

`buffer_limit`: the limit at which an alarm is generated if more than N bytes are queued for transmission on a socket. The default corresponds to half the buffer size. The buffer limit is shared for all monitoring jobs, i.e. it is always set to either the default or whatever was specified in the most recently created monitoring job, regardless of protocol.

`average_period`: the length of time, in seconds, over which average load is computed. The load limit alarm uses the average load as its trigger source. The maximum allowed value is 900 seconds.

`tag`: A user-supplied value which is then sent in the header of each packet generated by this job.

Counters and Indicators

Use `query` to read counters and indicators.

Category	Members	Description
Cell counters	n_cells	The total number of cells, including idle and O&M.
	n_idle	The number of idle cells which have arrived on the interface.
	n_oam	The number of O&M cells.
Load meters	current, average and maximum load	
State	current state	either `sync` or `hunt`
	n_sync, t_sync, t_hunt	The number of times the `sync` state was entered and the number of milliseconds spent in the `sync` and `hunt` states, respectively.

Monitoring Socket Protocol

The signalling information extracted from a link using an `atm_aal0_monitor` is forwarded to a socket on a remote machine defined by *ip_addr* and *ip_port*. This socket is independent of the controller's API socket. The GTH expects the external host to be listening on this socket before the `new` command is issued.

Each ATM cell is delivered with a 16 octet big-endian header:

<table>
<tr><th>Bit</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>octet 0x00</td><td colspan="16">Length</td></tr>
<tr><td>octet 0x02</td><td colspan="16">Tag</td></tr>
<tr><td>octet 0x04</td><td colspan="3">protocol = 4</td><td colspan="4">reserved</td><td>CR</td><td colspan="6">reserved</td><td colspan="2">Ver</td></tr>
<tr><td>octet 0x06</td><td colspan="16" rowspan="3">Timestamp</td></tr>
<tr><td>octet 0x08</td></tr>
<tr><td>octet 0x0a</td></tr>
<tr><td>octet 0x0c</td><td colspan="4">GFC</td><td colspan="8">VPI</td><td colspan="4">VCI...</td></tr>
<tr><td>octet 0x0e</td><td colspan="12">...VCI</td><td colspan="4">PTC</td></tr>
</table>

Length: the number of octets following the length field, including the rest of the header.

Tag: the tag value sent by the controller in the XML command

CR: set if the HEC (CRC) was incorrect

Ver: The header version, currently 0x0.

Timestamp: A 48-bit wide field indicating the instant the packet arrived, in number of milliseconds since the 1970 Unix epoch. This value is nondecreasing.

The *GFC, VPI, VCI and PTC* fields make up the ATM cell header. ITU-T I.361 describes the purpose of these fields in ATM networks.

See also

Section 2.6 describes general principles concerning layer 2 signalling sockets.

Link status information

An ATM link running normally is in link state `sync`. The other possible state, `hunt` indicates that the link is not working. Whenever the link switches from one to the other, GTH sends an `event`:

⇐ <event>
<atm_message **id**=string **value**='hunt'|'sync'/>
</event>

The GTH also issues an `event` whenever the *average_load* exceeds the value supplied in *load_limit*:

⇐ <event>
<l2_alarm **id**=string **attribute**=string **state**=string **value**=string/>
</event>

The GTH also issues an `event` whenever the TCP socket carrying the signalling data is congested (buffer more than half full), overruns or is closed remotely:

⇐ <event>

<l2_socket_alert
reason='buffer_limit'|'buffer_overrun'|'remote_close'
ip_addr=int.int.int.int
ip_port=int/>
</event>

Error reasons

`refused`: The GTH is already decoding signalling at maximum capacity.

`refused`: The GTH already has the maximum number (12) of signalling sockets open.

`conflict`: The GTH is already decoding a different, incompatible protocol on the given input timeslots.

Standards: ITU-T I.432.3, I.432.1, ANSI T1.111.1 section 2.2.3A.

Example: Monitoring a standard 1920kbit/s ATM AAL0 channel on an E1

```
⇒    <new>
       <atm_aal0_monitor ip_addr="172.16.2.1" ip_port="1234" tag="98">
         <pcm_source span="2A" timeslot="1"/>
         ...
         <pcm_source span="2A" timeslot="15"/>
         <pcm_source span="2A" timeslot="17"/>
         ...
         <pcm_source span="2A" timeslot="31"/>
       </atm_aal0_monitor>
     </new>
⇐    <job id="at0m72"/>
```

Example: Monitoring 150 Mbit/s ATM AAL0 channel on SDH

```
⇒    <new>
       <atm_aal0_monitor ip_addr="172.16.2.1" ip_port="1234" tag="98">
         <sdh_source name="sdh1:hop1_1"/>
       </atm_aal0_monitor>
     </new>
⇐    <job id="at0m73"/>
```

Example: Examining the AAL0 counters

```
⇒    <query><job id="at0m72"/></query>
```

⇐

```
<state>
  <atm_aal0_monitor id="at0m72" owner="apic18">
    <attribute name="span" value="1A"/>
    <attribute name="timeslot" value="3"/>
    <attribute name="n_cell" value="41631"/>
    <attribute name="n_idle" value="134690"/>
    <attribute name="n_oam" value="134"/>
    <attribute name="n_sync" value="1"/>
    <attribute name="t_sync" value="346113"/>
    <attribute name="t_hunt" value="24708"/>
    <attribute name="current state" value="sync"/>
    <attribute name="current load" value="8"/>
    <attribute name="average load" value="4"/>
    <attribute name="maximum load" value="32"/>
  </atm_aal0_monitor>
</state>
```

3.9 new atm_aal0_layer

⇒ <new>
    <atm_aal0_layer **ip_addr**=int.int.int.int **ip_port**=int scrambling='yes'>
        <pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
        ...
        <pcm_sink **span**=string **timeslot**=int bandwidth='64'/>
        ...
    </atm_aal0_layer>
</new>

⇐ <job **id**=string/>

The `atm_aal0_layer` is **unsupported**.

Use it to transmit ATM cells on one or more timeslots of an E1

After the initial setup through the XML interface, the specified socket is used to transfer signal units. All messages on the socket have a six octet header:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
octet 0	Length															
octet 2	GFC				VPI								VCI...			
octet 4	...VCI												PTC			

The *length* indicates the number of octets following the length field, including the rest of the header. This is always 52 octets.

The *GFC, VPI, VCI* and *PTC* fields make up the ATM cell header. ITU-T I.361 describes the purpose of these fields in ATM networks.

The GTH automatically calculates and inserts the header checksum (HEC) in the cell header.

3.10 new atm_aal2_monitor

```
⇒ <new>
      <atm_aal2_monitor
        vpi=int
        vci=int
        sdu='yes'
        corrupt_sdu='no'
        scrambling='yes'
        link_load_alarm='no'
        load_limit='50'
        buffer_limit='256000'
        average_period='30'
        tag='0'
        ip_addr=int.int.int.int
        ip_port=int>
          <pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
          ...
      </atm_aal2_monitor>
  </new>
```

```
⇒ <new>
      <atm_aal2_monitor
        vpi=int
        vci=int
        sdu='yes'
        corrupt_sdu='no'
        scrambling='yes'
        link_load_alarm='no'
        load_limit='50'
        buffer_limit='256000'
        average_period='30'
        tag='0'
        ip_addr=int.int.int.int
        ip_port=int>
          <sdh_source name=string/>
      </atm_aal2_monitor>
  </new>
```

```
⇐ <job id=string/>
```

An `atm_aal2_monitor` extracts packets from the interface above the ATM AAL2 layer on ATM links and forwards them to an external server over TCP.

On hardware with E1/T1/J1 interfaces, the ATM layer is always ATM-over-E1/T1/J1. E1 installations usually use timeslots 1–15 and 17–31 to make a 1920 kbit/s channel. T1 installations normally use 1–24 to make a 1536 kbit/s channel. The GTH supports both standard configurations, as well as arbitrary sets of timeslots.

Hardware with SDH/SONET interfaces supports additional modes: ATM can also be carried in a VC-4/STS-3c at 150 Mbit/s and at 48 Mbit/s ATM in a VC-3/STS-1.

The GTH supports starting multiple AAL2 monitors (with different VPI/VCI) on the same set of timeslots. AAL2 is used on some interfaces in UMTS (3G) systems.

vpi, vci: These parameters specify which AAL2 channel is monitored.

sdu, corrupt_sdu: select whether normal packets (SDUs) and corrupt SDUs should be delivered.

scrambling: ATM payload scrambling is on by default. ITU I.432.3 requires scrambling (x^{43} + 1) to be enabled for E1 links and leaves it optional for T1.

`buffer_limit`: the limit at which an alarm is generated if more than N bytes are queued for transmission on a socket. The default corresponds to half the buffer size. The buffer limit is shared for all monitoring jobs, i.e. it is always set to either the default or whatever was specified in the most recently created monitoring job, regardless of protocol.

`average_period`: the length of time, in seconds, over which average load is computed. The load limit alarm uses the average load as its trigger source. The maximum allowed value is 900 seconds.

`tag`: A user-supplied value which is then sent in the header of each packet generated by this job.

Counters and Indicators

Use `query` to read counters and indicators.

Category	Members	Description
Packet counters	n_sdu n_cdu n_oam	The number of valid SDUs. The number of corrupt SDUs. The number of O&M cells.
Octet counters	sdu_o cdu_o	
Load meters	current, average and maximum load	The percentage of the nominal link capacity used by this VPI/VCI combination.
	current, average and maximum *link* load	The percentage of the nominal link capacity used by *all* VPI/VCI channels on this ATM link.
State	current state `n_sync`, `t_sync`, `t_hunt`	either `sync` or `hunt` The number of times the `sync` state was entered and the number of milliseconds spent in the `sync` and `hunt` states, respectively.

Monitoring Socket Protocol

The signalling information extracted from a link using an `atm_aal2_monitor` is forwarded to a socket on a remote machine defined by *ip_addr* and *ip_port*. This socket is independent of the controller's API socket. The GTH expects the external host to be listening on this socket before the `new` command is issued.

Each AAL2 PDU is delivered with a 19 octet big-endian header:

<table>
<tr><td>Bit</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>octet 0x00</td><td colspan="16">Length</td></tr>
<tr><td>octet 0x02</td><td colspan="16">Tag</td></tr>
<tr><td>octet 0x04</td><td colspan="3">protocol = 6</td><td colspan="4">reserved</td><td>CR</td><td colspan="6">reserved</td><td colspan="2">Ver</td></tr>
<tr><td>octet 0x06
octet 0x08
octet 0x0a</td><td colspan="16">Timestamp</td></tr>
<tr><td>octet 0x0c</td><td colspan="4">GFC</td><td colspan="8">VPI</td><td colspan="4">VCI...</td></tr>
<tr><td>octet 0x0e</td><td colspan="12">...VCI</td><td colspan="4">PTC</td></tr>
<tr><td>octet 0x10</td><td colspan="8">CID</td><td colspan="6">LI</td><td colspan="2">UUI</td></tr>
<tr><td>octet 0x12</td><td colspan="3">UUI</td><td colspan="5">HEC</td><td colspan="8">(Payload)</td></tr>
</table>

Length: the number of octets following the length field, including the rest of the header.

Tag: the *tag* given in the `new` command

CR: CRC error in the AAL2 PDU

Ver: The header version, currently 0.

GFC, VPI, VCI, PTC: are all members of the AAL0 packet header

CID, LI, UUI, HEC: are all members of the CPS packet header, as described in ITU-T I.363.2. These fields are presented in the same format as I.363.2 specifies.

Link status information

An ATM link running normally is in link state `sync`. The other possible state, `hunt` indicates that the link is not working. Whenever the link switches from one to the other, GTH sends an `event`:

⇐ <event>
<atm_message **id**=string **value**='hunt'|'sync'/>
</event>

The GTH issues an `event` whenever the *average_load* exceeds the limit:

⇐ <event>
<l2_alarm **id**=string **attribute**=string **state**=string **value**=string/>
</event>

If the `link_load_alarm` attribute is `yes`, then GTH *also* issues load alarms for the *average link load*.

The GTH also issues an `event` whenever the TCP socket carrying the signalling data is congested (buffer more than half full), overruns or is closed remotely:

⇐ <event>
<l2_socket_alert
reason='buffer_limit'|'buffer_overrun'|'remote_close'
ip_addr=int.int.int.int
ip_port=int/>
</event>

Error reasons

`refused`: The GTH is already decoding signalling at maximum capacity.

`refused`: The GTH already has the maximum number (12) of signalling sockets open.

`conflict`: The GTH is already decoding a different, incompatible protocol on the given input timeslots.

Standards: ITU-T I.363.2

3.11 new atm_aal5_monitor

⇒ <new>
<atm_aal5_monitor
vpi=int
vci=int
sdu='yes'
corrupt_sdu='no'
scrambling='yes'
link_load_alarm='no'
load_limit='50'
buffer_limit='256000'
average_period='30'
timeout='0'
tag='0'
ip_addr=int.int.int.int
ip_port=int>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
...
</atm_aal5_monitor>
</new>

⇒ <new>
<atm_aal5_monitor
vpi=int
vci=int
sdu='yes'
corrupt_sdu='no'
scrambling='yes'
link_load_alarm='no'
load_limit='50'
buffer_limit='256000'
average_period='30'
timeout='0'
tag='0'
ip_addr=int.int.int.int
ip_port=int>
<sdh_source **name**=string/>
</atm_aal5_monitor>
</new>

⇐ <job **id**=string/>

An `atm_aal5_monitor` extracts packets from the interface above the CPCS sublayer of ATM AAL5 on ATM links and forwards them to an external server over TCP.

On hardware with E1/T1/J1 interfaces, the ATM layer is always ATM-over-E1/T1/J1. E1 installations usually use timeslots 1–15 and 17–31 to make a 1920 kbit/s channel. T1 installations normally use 1–24 to make a 1536 kbit/s channel. The GTH supports both standard configurations, as well as arbitrary sets of timeslots.

Hardware with SDH/SONET interfaces supports additional modes: ATM can also be carried in a VC-4/STS-3c at 150 Mbit/s and at 48 Mbit/s ATM in a VC-3/STS-1.

vpi, vci: These parameters specify which AAL5 channel is monitored.

sdu, corrupt_sdu: select whether normal packets (SDUs) and corrupt SDUs should be delivered.

scrambling: ATM payload scrambling is on by default. ITU I.432.3 requires scrambling (x^{43} + 1) to be enabled for E1 links and leaves it optional for T1.

timeout: The AAL5 re-assembly timeout, in seconds. 0 means disabled.

`buffer_limit`: the limit at which an alarm is generated if more than N bytes are queued for transmission on a socket. The default corresponds to half the buffer size. The buffer limit is shared for all monitoring jobs, i.e. it is always set to either the default or whatever was specified in the most recently created monitoring job, regardless of protocol.

`average_period`: the length of time, in seconds, over which average load is computed. The load limit alarm uses the average load as its trigger source. The maximum allowed value is 900 seconds.

`tag`: A user-supplied value which is then sent in the header of each packet generated by this job.

Counters and Indicators

Use `query` to read counters and indicators.

Category	Members	Description
Packet counters	n_sdu n_cdu	The number of valid SDUs The number of corrupt SDUs
Octet counters	sdu_o cdu_o	
Load meters	current, average and maximum load	The percentage of the nominal link capacity used by this particular VPI/VCI combination
	current, average and maximum *link* load	The percentage of the nominal link capacity used by *all* VPI/VCI channels on this ATM link.
State	current state n_sync, t_sync, t_hunt	either `sync` or `hunt` The number of times the `sync` state was entered and the number of milliseconds spent in the `sync` and `hunt` states, respectively.

Monitoring Socket Protocol

The signalling information extracted from a link using an `atm_aal5_monitor` is forwarded to a socket on a remote machine defined by *ip_addr* and *ip_port*. This socket is independent of the controller's API socket. The GTH expects the external host to be listening on this socket before the `new` command is issued.

Each AAL PDU is delivered with a 24 octet big-endian header:

<table>
<tr><th>Bit</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>octet 0x00</td><td colspan="16">Length</td></tr>
<tr><td>octet 0x02</td><td colspan="16">Tag</td></tr>
<tr><td>octet 0x04</td><td colspan="3">protocol = 5</td><td>res.</td><td>IL</td><td>CG</td><td>AB</td><td>CR</td><td colspan="6">reserved</td><td colspan="2">Ver</td></tr>
<tr><td>octet 0x06
octet 0x08
octet 0x0a</td><td colspan="16">Timestamp</td></tr>
<tr><td>octet 0x0c</td><td colspan="4">GFC</td><td colspan="8">VPI</td><td colspan="4">VCI...</td></tr>
<tr><td>octet 0x0e</td><td colspan="12">...VCI</td><td colspan="4">PTC</td></tr>
<tr><td>octet 0x10</td><td colspan="8">CPCS-UU</td><td colspan="8">CPI</td></tr>
<tr><td>octet 0x12</td><td colspan="16">CPCS-Length</td></tr>
<tr><td>octet 0x14</td><td colspan="16">CPCS-CRC...</td></tr>
<tr><td>octet 0x16</td><td colspan="16">...CPCS-CRC</td></tr>
</table>

Length: the number of octets following the length field, including the rest of the header.

Tag: the tag value sent by the controller in the XML command

IL: CPCS Invalid Length. Set if the length in the CPCS trailer is not consistent with the number of received octets.

CG: CPCS congestion indicator. Set if CPCS re-assembly failed after packets were dropped due to congestion.

AB: CPCS abort indicator. Set if the CPCS re-assembly failed due to a remote abort.

CR: CPCS CRC error indicator. Set if the CPCS PDU is corrupt.

Ver: The header version, currently 0x0.

Timestamp: A 48-bit wide field indicating the instant the packet arrived, in number of milliseconds since the 1970 Unix epoch. This value is nondecreasing.

GFC, VPI, VCI, PTC: are all members of the ATM cell header

CPCS-UU, CPI, CPCS-Length and *CPCS-CRC* are the CPCS trailer field, as described in ITU-T I.363.5.

Link status information

An ATM link running normally is in link state `sync`. The other possible state, `hunt` indicates that the link is not working. Whenever the link switches from one to the other, GTH sends an `event`:

⇐ <event>
<atm_message **id**=string **value**='hunt'|'sync'/>
</event>

The GTH issues an `event` whenever the *average_load* exceeds the limit:

⇐ <event>
<l2_alarm **id**=string **attribute**=string **state**=string **value**=string/>
</event>

If *link_load_alarm* is `yes`, then GTH *also* issues load alarms for the *average_link_load*.

The GTH also issues an `event` whenever the TCP socket carrying the signalling data is congested (buffer more than half full), overruns or is closed remotely:

⇐ <event>
<l2_socket_alert
reason='buffer_limit'|'buffer_overrun'|'remote_close'
ip_addr=int.int.int.int
ip_port=int/>
</event>

Error reasons

`refused`: The GTH is already decoding signalling at maximum capacity.

`refused`: The GTH already has the maximum number (12) of signalling sockets open.

`conflict`: The GTH is already decoding a different, incompatible protocol on the given input timeslots.

Standards: ITU-T I.363.5 and ANSI T1.111.1 section 2.2.3A, T1.635

Example: Monitoring a standard 1920kbit/s ATM AAL5 channel on an E1

This example uses VPI/VCI 0/5, which is the channel used in SS7 HSSL (high speed signalling link).

⇒
```
<new>
  <atm_aal5_monitor vpi="0" vci="5" ip_addr="172.16.2.1"
ip_port="1234" tag="98">
    <pcm_source span="2A" timeslot="1"/>
    ...
    <pcm_source span="2A" timeslot="15"/>
    <pcm_source span="2A" timeslot="17"/>
    ...
    <pcm_source span="2A" timeslot="31"/>
  </atm_aal5_monitor>
</new>
```

```
⇐    <job id="at5m199"/>
```

Example: Monitoring a 150 Mbit/s ATM AAL5 channel in an SDH VC-4

```
⇒    <new>
       <atm_aal5_monitor vpi="0" vci="36" ip_addr="172.16.2.1"
    ip_port="1234" tag="98">
         <sdh_source name="sdh2:hop1_1"/>
       </atm_aal5_monitor>
     </new>
⇐    <job id="at5m203"/>
```

Example: Examining the AAL5 counters

```
⇒    <query>
       <job id="at5m199"/>
     </query>
⇐    <state>
       <atm_aal5_monitor id="at5m199" owner="apic18">
         <attribute name="span" value="3A"/>
         <attribute name="timeslot" value="1"/>
         <attribute name="n_sdu" value="136134"/>
         <attribute name="sdu_o" value="13461346"/>
         <attribute name="n_cdu" value="5"/>
         ...
         <attribute name="maximum load" value="33"/>
         <attribute name="current link load" value="23"/>
         <attribute name="average link load" value="41"/>
         <attribute name="maximum link load" value="48"/>
       </atm_aal5_monitor>
     </state>
```

3.12 new cas_r2_linesig_monitor

⇒ <new>
<cas_r2_linesig_monitor tag='0' **ip_addr**=int.int.int.int **ip_port**=int>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
</cas_r2_linesig_monitor>
</new>

⇐ <job **id**=string/>

A CAS Line signalling monitor detects changes in the *a* and *b* bits in CAS signalling and forwards this information to a server over a TCP socket. The GTH reports each time the bits change and stay stable for at least 10ms.

CAS line signalling normally occupies timeslot 16 on an E1. The GTH can decode CAS signalling on any timeslot.

tag: A user-supplied value which is then sent in the header of each packet generated by this job.

Monitoring Socket Protocol

The signalling information extracted from a link using a `cas_r2_linesig_monitor` is forwarded to a socket on a remote machine defined by *ip_addr* and *ip_port*. This socket is independent of the controller's API socket. The GTH expects the external host to be listening on this socket before the `new` command is issued.

Each bit change is delivered with a 16 octet big-endian header:

<table>
<tr><th>Bit</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>octet 0x00</td><td colspan="16">Length (always 0x0e)</td></tr>
<tr><td>octet 0x02</td><td colspan="16">Tag</td></tr>
<tr><td>octet 0x04</td><td colspan="3">protocol$_1$ = 7</td><td colspan="5">reserved</td><td colspan="3">protocol$_2$ = 2</td><td colspan="5">reserved</td></tr>
<tr><td>octet 0x06
octet 0x08
octet 0x0a</td><td colspan="16">Timestamp</td></tr>
<tr><td>octet 0x0c</td><td colspan="16">reserved</td></tr>
<tr><td>octet 0x0e</td><td colspan="8">Channel</td><td colspan="8">Bits</td></tr>
</table>

Length: the number of octets following the length field, including the rest of the header.

Tag: the tag value sent by the controller in the XML command

Channel: The "telephone channel number" as numbered in G.704 (i.e. timeslots 1–15, 17–31 are numbered 1–30).

Bits: The current *a,b,c,d* bits. CAS only uses the first two out of the four bits. The *c* and *d* are normally fixed at 0 and 1, respectively.

Error reasons

`refused`: The GTH is already decoding signalling at maximum capacity.

`refused`: The GTH already has the maximum number (12) of signalling sockets open.

`conflict`: The GTH is already decoding a different, incompatible protocol on the given input timeslots.

Standards: ITU-T Q.421, ITU-T G.704 (Table 9)

Example: Extracting CAS line signalling

⇒
```
<new>
  <cas_r2_linesig_monitor tag="1234" ip_addr="172.16.2.1"
ip_port="1234">
    <pcm_source span="2A" timeslot="16"/>
  </cas_r2_linesig_monitor>
</new>
```

⇐
```
<job id="clsm19"/>
```

3.13 new cas_r2_mfc_detector

⇒ <new>
<cas_r2_mfc_detector tag='0' **direction**='forward'|'backward'
ip_addr=int.int.int.int **ip_port**=int>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
</cas_r2_mfc_detector>
</new>

⇐ <job **id**=string/>

A CAS R2 MFC register signalling detector detects the CAS tones 1–15. CAS MFC is normally used in conjunction with CAS line signalling.

tag: A user-supplied value which is then sent in the header of each packet generated by this job.

direction: CAS MFC uses one set of frequencies for each direction of signalling. This parameter selects which of the two.

Monitoring Socket Protocol

The signalling information extracted from a link using a `cas_r2_mfc_detector` is forwarded to a socket on a remote machine defined by *ip_addr* and *ip_port*. This socket is independent of the controller's API socket. The GTH expects the external host to be listening on this socket before the `new` command is issued.

Each MFC tone is delivered with a 16 octet big-endian header:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
octet 0x00	Length (always 0x0e)															
octet 0x02	Tag															
octet 0x04	$protocol_1$ = 7			reserved					$protocol_2$ = 1			reserved				
octet 0x06 octet 0x08 octet 0x0a	Timestamp															
octet 0x0c	reserved															
octet 0x0e	Type								Digit							

Length: the number of octets following the length field, including the rest of the header.

Tag: the tag value sent by the controller in the XML command

Type: 1=start-of-tone, 2=end-of-tone

Digit: The Q.400 digit number (1–15).

Error reasons

`refused`: The GTH is already decoding signalling at maximum capacity.

`refused`: The GTH already has the maximum number (12) of signalling sockets open.

`conflict`: The GTH is already decoding a different, incompatible protocol on the given input timeslots.

Standards: ITU-T Q.440 and ITU-T Q.441 (Table 5)

Example: Monitoring CAS R2 MFC register signalling

⇒
```
<new>
  <cas_r2_mfc_detector direction="forward" tag="1234"
ip_addr="172.16.2.1" ip_port="1234">
    <pcm_source span="2A" timeslot="3"/>
  </cas_r2_mfc_detector>
</new>
```

⇐
```
<job id="cmfm17"/>
```

3.14 new clip

⇒ <new>
<clip **id**=string/>
</new>

⇐ <job **id**=string/>

A clip represents a short sequence of sampled audio stored on the GTH. It could be a tone, e.g. a dial tone, or some voice, for instance a greeting for a mailbox.

id: a string of the controller's choosing. The job ID returned by the GTH will be that string, with a "clip " prefix.

The GTH expects a block of audio immediately following the end of the XML command. The block must have content type `binary/audio` and it may be up to 500000 octets (60 seconds of audio) long.

In a system which uses large clips, sending a clip to the GTH may tie up the Ethernet interface for a few hundred milliseconds. Applications can avoid delaying other commands by using a second API socket for the timing-critical operations.

Error reasons

`refused`: The GTH does not have enough memory to store this clip, either because there are too many clips on the GTH or because the total length of all the clips on the GTH is too large.

See also

The `player` command (see 3.22) plays a clip on a timeslot.

Example: Defining a clip

```
⇒<new><clip id="dtmf9"/></new>

Content-type: binary/audio
Content-length: 120
(the audio data)

⇐<job id="clip dtmf9"/>
```

A `player` is used to play the audio on a timeslot. `player` copies the clip byte-by-byte without any modifying it and without removing any header (e.g. .wav header) which might be present. So, the audio data you provide to a GTH should not have a header and should be 8000 samples per second, mono, A-law or mu-law.

3.15 new connection

⇒ <new>

<connection>

<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>

<pcm_sink **span**=string **timeslot**=int bandwidth='64'/>

</connection>

</new>

⇐ <job **id**=string/>

A connection is used for switching. A connection establishes a simplex stream of data from *one* source to *one* sink.

An ordinary telephone call is made by creating two simplex connections.

An N-part conference call is made by setting up multiple connections to the same sink: one from each source which that sink should hear. The subscriber will then hear the linear sum of all the sources. If necessary, the sum will be clipped.

Calls can also be intercepted without interruption by creating one or more additional connections from the source or sources to an intercepting sink.

An implicit conference is also created if a recorded message, i.e. a `player`, is currently playing to the same sink as a connection: the subscriber will hear the linear sum of the player and the connection.

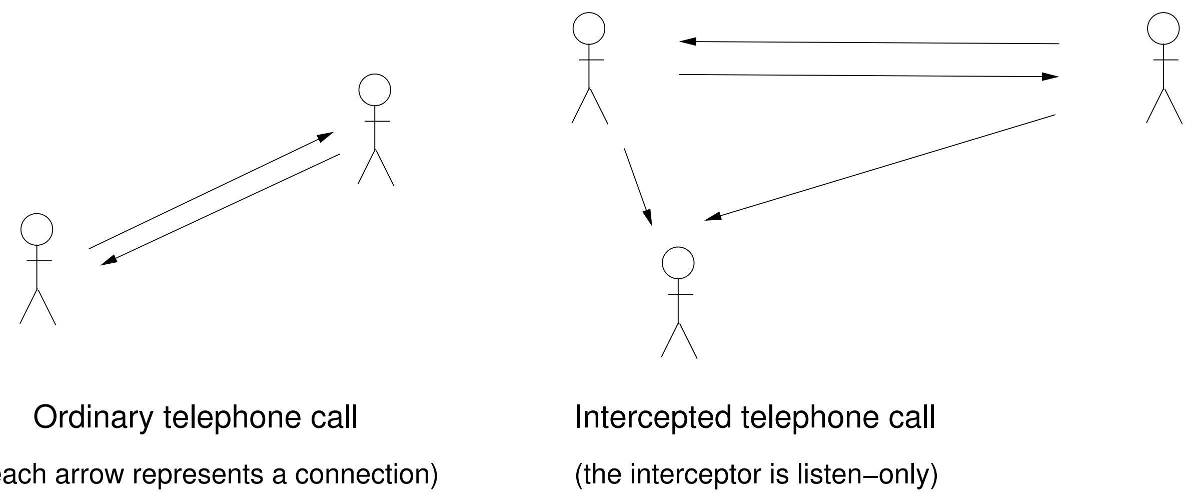


Figure 1: Two examples of call switching

Error reasons

`refused`: The GTH has used up all available switching capacity.

See also

The *voice coding* attribute of the `board` resource (see 4.10) can be set to either alaw or mulaw voice coding. Conferences will have very bad audio quality if the GTH's coding is different to the coding used in the telephone network.

The *auto conferences* attribute of the `board` resource (see 4.10) can be set to disable conferences.

Example: Creating a "half call"

```
⇒     <new>
        <connection>
          <pcm_source span="2A" timeslot="16"/>
          <pcm_sink span="3A" timeslot="1"/>
        </connection>
      </new>
⇐     <job id="cnxn15"/>
```

It's called a "half call" because the connection is unidirectional, i.e. the subscriber at span 3, timeslot 1 can hear the subscriber at span 2, timeslot 16, but not vice versa. A normal telephone call consists of two half calls.

Example: Breaking a half call

```
⇒     <delete id="cnxn15"/>
⇐     <ok/>
```

Example: Creating a 3-party conference call

```
⇒     <new><connection><pcm_source span="1A" timeslot="1"/>
          <pcm_sink span="2A" timeslot="1"/></connection></new>
⇐     <job id="cnxn16"/>
⇒     <new><connection><pcm_source span="1A" timeslot="1"/>
          <pcm_sink span="3A" timeslot="1"/></connection></new>
⇐     <job id="cnxn17"/>
⇒     <new><connection><pcm_source span="2A" timeslot="1"/>
          <pcm_sink span="1A" timeslot="1"/></connection></new>
⇐     <job id="cnxn18"/>
⇒     <new><connection><pcm_source span="2A" timeslot="1"/>
          <pcm_sink span="3A" timeslot="1"/></connection></new>
⇐     <job id="cnxn19"/>
⇒     <new><connection><pcm_source span="3A" timeslot="1"/>
          <pcm_sink span="2A" timeslot="1"/></connection></new>
```

```
⇐    <job id="cnxn20"/>
⇒    <new><connection><pcm_source span="3A" timeslot="1"/>
         <pcm_sink span="1A" timeslot="1"/></connection></new>
⇐    <job id="cnxn21"/>
```

3.16 new fr_layer

⇒ <new>
<fr_layer **ip_addr**=int.int.int.int **ip_port**=int>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
...
<pcm_sink **span**=string **timeslot**=int bandwidth='64'/>
...
</fr_layer>
</new>

⇐ <job **id**=string/>

The `fr_layer` is used to transmit Frame Relay and SS7 MTP-2 packets on E1/T1 Messenger 3.0.

After the initial setup through the XML interface, the specified socket is used to transfer signal units. All messages on the socket have a six octet header:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
octet 0	Length															
octet 2	Opcode															
octet 4	reserved = 0															
octet 6	Payload															

The *length* indicates the number of octets following the length field, including the rest of the header.

opcode=2 means *transmit packet*.

opcode=3 means *transmit MTP-2 packet*. The packet is transmitted as for opcode 2. In addition, the hardware automatically transmits FISUs, using the FSN and BSN from the transmitted packet.

The GTH automatically calculates and appends the frame check sequence to the payload.

3.17 new fr_monitor

```
⇒ <new>
      <fr_monitor
        su='yes'
        esu='no'
        load_limit='50'
        buffer_limit='256000'
        average_period='30'
        tag='0'
        timeout='0'
        ip_addr=int.int.int.int
        ip_port=int>
          <pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
          ...
      </fr_monitor>
   </new>
```

```
⇐ <job id=string/>
```

A frame relay monitor extracts frame relay signal units from a frame relay link and forwards them to a server over TCP. Common input bandwidths are 256 kbit/s and 1980 kbit/s. The GTH supports all multiples of 64 kbit/s.

All timeslots in a channel must be from the same E1/T1/J1 span. Timeslots need not be consecutive, but they must be in ascending order.

su, esu: Select whether or not correct signal units (packets) and errored signal units are delivered.

`buffer_limit`: the limit at which an alarm is generated if more than N bytes are queued for transmission on a socket. The default corresponds to half the buffer size. The buffer limit is shared for all monitoring jobs, i.e. it is always set to either the default or whatever was specified in the most recently created monitoring job, regardless of protocol.

`average_period`: the length of time, in seconds, over which average load is computed. The load limit alarm uses the average load as its trigger source. The maximum allowed value is 900 seconds.

`tag`: A user-supplied value which is then sent in the header of each packet generated by this job.

timeout: If nonzero, it indicates the maximum time, in seconds, the channel can be without packets before it is considered to be `down`.

Counters and Indicators

Use `query` to read counters and indicators.

Category	Members	Description
Packet counters	n_su	The number of signal-units which have arrived on the interface.
	n_esu	The number of errored signal units, which is defined as all packets which are too short, too long, non octet-aligned or have an incorrect CRC.
Octet counters	su_o, esu_o	The total number of octets in the packet types above.
Load meters	current, average and maximum load	
State	current state	either `up` or `down`
	n_up, t_up, n_down, t_down	The number of times the `up` and `down` states were entered, and the number of milliseconds spent in each state.

Link status information

Frame relay links have two possible link states: `up` and `down`. A link moves to the `up` state when a correct SU is received. A link moves to the `down` state for two possible reasons:

1. No correct SUs are received for *timeout* seconds.
2. The space between SUs is filled with something other than flags, for instance abort (0xff).

Whenever the link switches from one to the other, GTH sends an `event`:

⇐ <event>
<f_relay_message **id**=string **value**=string/>
</event>

The GTH issues an `event` whenever the *average_load* exceeds the value specified in the *load_limit*:

⇐ <event>
<l2_alarm **id**=string **attribute**=string **state**=string **value**=string/>
</event>

The GTH also issues an `event` whenever the TCP socket carrying the signalling data is congested (buffer more than half full), overruns or is closed remotely:

⇐ <event>

<l2_socket_alert
reason='buffer_limit'|'buffer_overrun'|'remote_close'
ip_addr=int.int.int.int
ip_port=int/>
</event>

Monitoring Socket Protocol

The signalling information extracted from a link using a `fr_monitor` is forwarded to a socket on a remote machine defined by *ip_addr* and *ip_port*. This socket is independent of the controller's API socket. The GTH expects the external host to be listening on this socket before the `new` command is issued.

Each signal unit is delivered with a 12 octet big-endian header:

<table>
<tr><th>Bit</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>octet 0x00</td><td colspan="16">Length</td></tr>
<tr><td>octet 0x02</td><td colspan="16">Tag</td></tr>
<tr><td>octet 0x04</td><td colspan="3">protocol=2</td><td>FS</td><td>FL</td><td>NA</td><td>AF</td><td>CR</td><td colspan="8">reserved</td></tr>
<tr><td>octet 0x06
octet 0x08
octet 0x0a</td><td colspan="16">Timestamp</td></tr>
</table>

length: indicates the number of octets following the length field, including the rest of the header and the CRC.

tag: specified by the user. It identifies the channel.

FS: 1 = Frame too short.

FL: 1 = Frame too long.

NA: 1 = Non octet-aligned frame (not a multiple of 8 bits).

AF: 1 = Aborted frame (frame was terminated by an abort signal).

CR: 1 = Invalid CRC.

timestamp: Timestamp marks the instant of time the packet ended, measured in milliseconds since the Unix epoch. It is a monotonically increasing integer.

Error reasons

`refused`: The GTH is already decoding signalling at maximum capacity.

`refused`: The GTH already has the maximum number (12) of signalling sockets open.

`conflict`: The GTH is already decoding a different, incompatible protocol on the given input timeslots.

Standards: ITU-T Q.922

Example: Monitoring a quad-timeslot (256 kbit/s) frame relay channel

⇒
```
<new>
  <fr_monitor ip_addr="172.16.2.1" ip_port="1234" tag="98">
    <pcm_source span="2A" timeslot="13"/>
    <pcm_source span="2A" timeslot="14"/>
    <pcm_source span="2A" timeslot="15"/>
    <pcm_source span="2A" timeslot="17"/>
  </fr_monitor>
</new>
```

⇐
```
<job id="frmo146"/>
```

3.18 new lapd_layer

⇒ <new>
<lapd_layer
side='network'
sapi='0'
tei='0'
tag=int
ip_addr=int.int.int.int
ip_port=int>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
<pcm_sink **span**=string **timeslot**=int bandwidth='64'/>
</lapd_layer>
</new>

⇐ <job **id**=string/>

A `lapd_layer` terminates a LAPD (ISDN Layer 2) signalling link. The GTH can act as either the `network` or the `user` side of the link, selectable on a per-job basis.

A `lapd_layer` requires both a sink and source. Normally these will be the same E1/T1/J1 and timeslot, usually timeslot 16.

Side: either `network` or `user`

SAPI and *TEI*: identifiers the GTH uses on the signalling link.

tag: a user-specified integer in the range [0, 65535]. Each signal unit is marked with this tag.

ip_addr and *ip_port:* these parameters define the TCP port the GTH will connect to in order to transfer the signal units.

After the initial setup through the XML interface, the specified socket is used to transfer signal units. All messages on the socket have an eight octet header:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
octet 0	Length															
octet 2	Tag															
octet 4	protocol = 1			Version = 1					Opcode							
octet 6	reserved															

The *length* indicates the number of octets following the length field, including the rest of the header.

The *tag* is a user-specified channel identifier. The tag must have the same value in the initial XML command and in all subsequent communication with the LAPD data transfer socket.

The messages transmitted on the socket by the GTH are:

Opcode	Signal name	Data following the header
0x02	DL Data Indication	Up to N201 octets of payload
0x06	DL Establish Confirm	none (length is always 6)
0x07	DL Establish Indication	none
0x09	DL Release Confirm	none
0x0A	DL Release Indication	none
0x10	MDL Error Indication	one octet: the ASCII value of the error code (A–O)

The messages transmitted on the socket *to* the GTH are:

Opcode	Signal name	Data following the header
0x01	DL Data Request	Up to N201 octets of payload
0x05	DL Establish Request	none
0x08	DL Release Request	none

The implementation-defined timer and counter values are:

Parameter	Default value	Unit
T200	1000	ms
T203	10000	ms
N200	3	attempts
N201	260	octets
k	7	outstanding I frames

Error reasons

`refused`: The GTH is already decoding signalling at maximum capacity.

`refused`: The GTH already has the maximum number (12) of signalling sockets open.

`conflict`: The GTH is already decoding a different, incompatible protocol on the given input timeslots.

Standards: ITU-T Q.920 and ITU-T Q.921.

Example: Setting up one duplex LAPD channel

⇒
```
<new>
  <lapd_layer ip_addr="172.16.2.1" ip_port="1234" tag="548"
side="network" sapi="3" tei="9">
    <pcm_source span="2A" timeslot="16"/>
    <pcm_sink   span="2A" timeslot="16"/>
  </lapd_layer>
</new>
```

⇐
```
<job id="lapd123"/>
```

3.19 new lapd_monitor

```
⇒ <new>
      <lapd_monitor
        id=string
        su='yes'
        esu='no'
        load_limit='50'
        buffer_limit='256000'
        average_period='30'
        tag='0'
        timeout='15'
        detect_abort='yes'
        ip_addr=int.int.int.int
        ip_port='0'>
          <pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
          ...
      </lapd_monitor>
  </new>

⇐ <job id=string/>
```

A LAPD monitor extracts ISDN LAPD signal units from a timeslot and forwards the signal units to the TCP port specified by the IP address and port number.

su, esu: Whether or not correct signal units and error signal units should be delivered, respectively.

`buffer_limit`: the limit at which an alarm is generated if more than N bytes are queued for transmission on a socket. The default corresponds to half the buffer size. The buffer limit is shared for all monitoring jobs, i.e. it is always set to either the default or whatever was specified in the most recently created monitoring job, regardless of protocol.

`average_period`: the length of time, in seconds, over which average load is computed. The load limit alarm uses the average load as its trigger source. The maximum allowed value is 900 seconds.

`tag`: A user-supplied value which is then sent in the header of each packet generated by this job.

timeout: If there are no correct SUs for this many seconds, the link state moves to `down`.

detect_abort: If the link switches to sending *abort* signals, change link states to `down`. A standards-conforming LAPD link in normal operation never contains an abort signal (seven or more consecutive 1-bits).

Monitoring Socket Protocol

The signalling information extracted from a link using a `lapd_monitor` is forwarded to a socket on a remote machine defined by *ip_addr* and *ip_port*. This socket is independent of the controller's API socket. The GTH expects the external host to be listening on this socket before the `new` command is issued.

Each signal unit is delivered with a 12 octet big-endian header:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
octet 0x00	Length															
octet 0x02	Tag															
octet 0x04	protocol=1			FS	FL	NA	AF	CR	reserved							
octet 0x06	Timestamp															
octet 0x08																
octet 0x0a																

length: indicates the number of octets following the length field, including the rest of the header and the CRC.

tag: specified by the user. It identifies the channel.

FS: 1 = Frame too short.

FL: 1 = Frame too long.

NA: 1 = Non octet-aligned frame (not a multiple of 8 bits).

AF: 1 = Aborted frame (frame was terminated by an abort signal).

CR: 1 = Invalid CRC.

timestamp: Timestamp marks the instant of time the packet ended, measured in milliseconds since the Unix epoch. It is a monotonically increasing integer.

Counters and Indicators

Category	Members	Description
Packet counters	n_su	The number of correct signal-units which have arrived on the interface.
	i_frames, s_frames, u_frames	The number of correct signal-units, broken down into types.
	n_esu	The number of errored signal units, which is defined as all packets which are too short, too long, non octet-aligned or have an incorrect CRC.
Octet counters	su_o, esu_o	The total number of octets in the packet types above.
Load meters	current, average and maximum load	
State	current state	either *up* or *down*.
	n_up, t_up, n_down, t_down	The number of times the *up* and *down* states were entered, and the number of milliseconds spent in each state.

Link status information

LAPD links have two possible link states: `up` and `down`. A link moves to the `up` state when a correct SU is received. A link moves to the `down` state for two possible reasons:

1. No correct SUs are received for *timeout* seconds.
2. The space between SUs is filled with something other than flags, for instance abort (0xff).

Whenever the link switches from one to the other, GTH sends an `event`:

⇐ <event>
<lapd_message **id**=string **value**='up'|'down'/>
</event>

The GTH issues an `event` whenever the *average_load* exceeds the *load_limit*:

⇐ <event>
<l2_alarm **id**=string **attribute**=string **state**=string **value**=string/>
</event>

The GTH also issues an `event` whenever the TCP socket carrying the signalling data is congested (buffer more than half full), overruns or is closed remotely:

⇐ <event>
<l2_socket_alert
reason='buffer_limit'|'buffer_overrun'|'remote_close'
ip_addr=int.int.int.int
ip_port=int/>
</event>

Error reasons

`refused`: The GTH is already decoding signalling at maximum capacity.

`refused`: The GTH already has the maximum number (12) of signalling sockets open.

`conflict`: The GTH is already decoding a different, incompatible protocol on the given input timeslots.

Standards: ITU-T Q. 921

Example: Monitoring a 64 kbit/s LAPD timeslot

```
⇒   <new>
      <lapd_monitor ip_addr="172.16.2.1" ip_port="1234" tag="77">
        <pcm_source span="2A" timeslot="16"/>
      </lapd_monitor>
    </new>
⇐   <job id="ldmo81"/>
```

Example: Monitoring subrate LAPD timeslots

Some GSM interfaces use 16 kbit/s or 32 kbit/s LAPD. This pair of examples, shows a 16 kbit/s channel on bits 4 and 5 in each octet and then a 32 kbit/s channel on bits 4,5,6 and 7.

```
⇒   <new>
      <lapd_monitor ip_addr="172.16.2.1" ip_port="1234" tag="77">
        <pcm_source span="2A" timeslot="16" first_bit="4" bandwidth="16"/>
      </lapd_monitor>
    </new>
⇐   <job id="ldmo82"/>
⇒   <new>
      <lapd_monitor ip_addr="172.16.2.1" ip_port="1234" tag="78">
        <pcm_source span="2A" timeslot="17" first_bit="4" bandwidth="32"/>
      </lapd_monitor>
    </new>
⇐   <job id="ldmo83"/>
```

3.20 new level_detector

⇒ <new>
<level_detector
threshold='-10'
type='both'|'low_to_high'|'high_to_low'
period='100'>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
</level_detector>
</new>

⇐ <job **id**=string/>

A `level_detector` requires an audio source. Whenever it detects that the audio level (the power on that timeslot) crosses the specified threshold, an `event` appears on the API socket:

⇐ <event>
<level **detector**=string **state**=string/>
</event>

threshold: -60 .. +6. The power threshold is in dBm0, as defined in ITU-T G.711. +6 is the loudest possible signal on a timeslot and -60 is the quietest possible. The practical range is from about -50 to about +0, with an accuracy of about 3dB.

type: Determines which transitions are reported. By default, both transitions from low power (silence) to high power (speech) and vice versa are reported.

period: 100 ... 10000. The time period the power is measured for, in milliseconds.

In an IVR application, a level detector could be used to detect a subscriber repeatedly attempting to speak to the application (instead of pressing keys on their handset) so that the subscriber can be forwarded to a human operator. In a telephone conference system (party line), a level detector can help identify which party is currently speaking.

Error reasons

`refused`: The GTH does not have enough capacity left to start another level detector.

Example: Level Detection

⇒
```
<new>
  <level_detector threshold="-10" type="both" period="200">
    <pcm_source span="2A" timeslot="19"/>
  </level_detector>
</new>
```

⇐
```
<job id="lede1"/>
```

3.21 new mtp2_monitor

⇒ <new>
<mtp2_monitor
id=string
fisu='yes'
dup_fisu='no'
lssu='yes'
dup_lssu='no'
msu='yes'
esu='no'
mark_likely_retrans='no'
load_limit='50'
buffer_limit='256000'
average_period='30'
tag='0'
esnf='no'
ip_addr=int.int.int.int
ip_port='0'>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
...
</mtp2_monitor>
</new>

⇐ <job **id**=string/>

An MTP-2 monitor passively extracts MTP-2 signal units from an MTP-2 link and forwards them to a server over TCP. Both single timeslot (48, 56 or 64 kbit/s) and a list of timeslots (ITU-T Q.703 Annex A, or ANSI T1.111.1 section 2.2.2) are supported.

`id`: only required when issuing an `update`. The job ID of the item being updated. It is not allowed as part of a `new`.

`fisu`: whether or not FISUs should be sent to the external system.

`dup_fisu`: whether or not duplicate FISUs should be delivered.

`lssu, dup_lssu, msu`: analogous to `fisu/dup_fisu`.

`mark_likely_retrans`: undocumented and unsupported option.

`ip_addr`: the destination IP address for the packets. A dotted quad, e.g. "172.16.2.1".

`load_limit`: a percentage of maximum MSU load; if the load exceeds this an `event` is generated.

`buffer_limit`: the limit at which an alarm is generated if more than N bytes are queued for transmission on a socket. The default corresponds to half the buffer size. The buffer limit is shared for all monitoring jobs, i.e. it is always set to either the default or whatever was specified in the most recently created monitoring job, regardless of protocol.

`average_period`: the length of time, in seconds, over which average load is computed. The load limit alarm uses the average load as its trigger source. The maximum allowed value is 900 seconds.

`tag`: A user-supplied value which is then sent in the header of each packet generated by this job.

`esnf`: Enables the "extended sequence number format" often used on 2 Mbit/s ITU-T Q.703 annex A links.

Monitoring Socket Protocol

The signalling information extracted from a link using an `mtp2_monitor` is forwarded to a socket on a remote machine defined by *ip_addr* and *ip_port*. This socket is independent of the controller's API socket. The GTH expects the external host to be listening on this socket before the `new` command is issued.

Each signal unit is delivered with a 12 octet big-endian header:

<table>
<tr><th>Bit</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>octet 0x00</td><td colspan="16">Length</td></tr>
<tr><td>octet 0x02</td><td colspan="16">Tag</td></tr>
<tr><td>octet 0x04</td><td colspan="3">protocol=0</td><td>FS</td><td>FL</td><td>NA</td><td>AF</td><td>CR</td><td colspan="8">reserved</td></tr>
<tr><td>octet 0x06
octet 0x08
octet 0x0a</td><td colspan="16">Timestamp</td></tr>
</table>

length: indicates the number of octets following the length field, including the rest of the header and the CRC.

tag: specified by the user. It identifies the channel.

FS: 1 = Frame too short.

FL: 1 = Frame too long.

NA: 1 = Non octet-aligned frame (not a multiple of 8 bits).

AF: 1 = Aborted frame (frame was terminated by an abort signal).

CR: 1 = Invalid CRC.

timestamp: Timestamp marks the instant of time the packet ended, measured in milliseconds since the Unix epoch. It is a monotonically increasing integer.

Counters and Indicators

Use `query` to read counters and indicators.

Category	Members	Description
Packet counters	n_fisu, n_lssu, n_msu, n_esu, n_rsu	The number of packets of different MTP-2 types which have arrived on the interface. *n_esu* is the number of errored packets, which is defined as all packets which are too short, too long, non octet-aligned or incorrect CRC. *n_rsu* is the number of retransmitted MSUs. When the FIB inverts, all subsequent MSUs up to the first one with a FSN equal to the FSN at FIB inversion time are counted as retransmitted MSUs.
Octet counters	fisu_o, lssu_o, msu_o, esu_o, rsu_o	The total number of octets in the packet types above.
Load meters	current, average and maximum load	
State	current state	(see table below)
	n_in service, n_congested, ...	The number of times the link was in each state.
	t_in service, t_congested, ...	The number of milliseconds spent in each state.

The current state and the state counters can be used to infer part of the MTP-2 transmitter's internal state:

Packet received	State entered
LSSU SIB	congested
LSSU SIO or SIOS	out of service
LSSU SIPO	processor outage
FISU or MSU	in service
no valid packet for 1s	no signal units

Link status information

MTP-2 monitors have five possible link states. Whenever the monitored link changes states, GTH sends an `event` to the monitoring job's owner:

⇐ <event>
<mtp2_message
id=string
value='in service'|'out of service'|'proc outage'|'congested'|'no signal units'/>
</event>

Load alarm events

The GTH issues an `event` whenever the *average_load* exceeds the *load_limit*:

⇐ <event>
<l2_alarm **id**=string **attribute**=string **state**=string **value**=string/>
</event>

The GTH also issues an `event` whenever the TCP socket carrying the signalling data is congested (buffer more than half full), overruns or is closed remotely:

⇐ <event>
<l2_socket_alert
reason='buffer_limit'|'buffer_overrun'|'remote_close'
ip_addr=int.int.int.int
ip_port=int/>
</event>

Error reasons

`refused`: The GTH is already decoding signalling at maximum capacity.

`refused`: The GTH already has the maximum number (12) of signalling sockets open.

`conflict`: The GTH is already decoding a different, incompatible protocol on the given input timeslots.

Standards: ITU-T Q.703, ANSI T1.111.1/3, JT-Q.702, JT-Q.703

Example: Monitoring an ordinary 64 kbit/s MTP-2 timeslot

```
⇒    <new>
       <mtp2_monitor tag="1234" ip_addr="172.16.2.1" ip_port="1234">
         <pcm_source span="2A" timeslot="16"/>
       </mtp2_monitor>
     </new>
⇐    <job id="m2mo17"/>
```

Example: Monitoring a 56 kbit/s ANSI MTP-2 timeslot

```
⇒    <new>
       <mtp2_monitor tag="1234" ip_addr="172.16.2.1" ip_port="1234">
         <pcm_source span="15B" timeslot="16" bandwidth="56"/>
       </mtp2_monitor>
     </new>
⇐    <job id="m2mo18"/>
```

Example: Monitoring a 48 kbit/s MTP-2 timeslot (Japanese MTP-2; NTT/TTC)

```
⇒   <new>
      <mtp2_monitor tag="1234" ip_addr="172.16.2.1" ip_port="1234">
        <pcm_source span="13A" timeslot="1" bandwidth="48" first_bit="1"/>
      </mtp2_monitor>
    </new>
⇐   <job id="m2mo19"/>
```

Example: Monitoring a 1920 kbit/s HSSL MTP-2 timeslot

```
⇒   <new>
      <mtp2_monitor tag="1234" ip_addr="172.16.2.1" ip_port="1234">
        <pcm_source span="8A" timeslot="1"/>
        ...
        <pcm_source span="8A" timeslot="15"/>
        <pcm_source span="8A" timeslot="17"/>
        ...
        <pcm_source span="8A" timeslot="31"/>
      </mtp2_monitor>
    </new>
⇐   <job id="m2mo733"/>
```

Example: Finding out how many FISUs have been filtered

```
⇒   <query>
      <job id="m2mo17"/>
    </query>
⇐   <state>
      <mtp2_monitor id="m2mo590" owner="apic89">
        <attribute name="span" value="1A"/>
        <attribute name="timeslot" value="3"/>
        <attribute name="n_fisu" value="571984"/>
        <attribute name="fisu_o" value="2859920"/>
        <attribute name="n_lssu" value="0"/>
        ...
      </mtp2_monitor>
    </state>
```

3.22 new player

⇒ <new>
<player loop='false'>
<clip **id**=string/>
...
<pcm_sink **span**=string **timeslot**=int bandwidth='64'/>
</player>
</new>

⇒ <new>
<player loop='false'>
<tcp_source **ip_addr**=int.int.int.int **ip_port**=int/>
<pcm_sink **span**=string **timeslot**=int bandwidth='64'/>
</player>
</new>

⇐ <job **id**=string/>

A player is used to play audio on an E1/T1 timeslot. The GTH plays the audio exactly as is, octet for octet. The input data can be voice, tones or even raw signalling data.

There are two possible sources of audio. Either, the audio is streamed in over a dedicated TCP socket, a `tcp_source`, or the audio is replayed from a `clip` which was previously stored in the GTH's memory.

loop: when playing from a `clip`, setting *loop*=`true` repeats the clip forever.

When all the audio has been played on the timeslot, the GTH automatically deletes the player job and issues an `event`:

⇐ <event>
<message_ended **id**=string/>
</event>

How to play from a `tcp_source`:

1. Establish a *listening* TCP socket on the controller. In C on Unix-like systems, this is done using the `listen()` call.
2. Issue the XML command to start a player. The GTH will now connect to the specified TCP socket on the controller.
3. On the controller, accept the inbound connection. In C, this is done using the `accept()` call.
4. On the controller, send the audio data. TCP has flow control, so the data being sent is not timing sensitive. Write data until the socket blocks.

Implicit Conferences

If another player is already playing audio to the sink, the subscriber will hear the linear sum of the two audio sources.

If a player is currently playing to the same sink as a connection, the subscriber will hear the linear sum of the player and the connection.

Error handling

Things can go wrong with players both during initial startup and while the player is running. If the error is detected early, the GTH returns an `error` response. If the error is detected after a `job_id` was returned, the GTH indicates errors by sending a `fatality`.

The possible *reason* attributes in an `error` response are

`refused`: The GTH does not currently have enough capacity to start another player.

`refused`: There are too many clips in the list of clips to be played (in gth2_system_33a and later, the limit is 500 clips).

The human-readable text accompanying the `error` provides more information to help distinguish between the different causes for *refused*.

A player using `clips` for input cannot fail once it has started. A player using a `tcp_source` can fail after starting. When this happens, the GTH sends a `fatality`:

```
⇐ <event>
    <fatality id="strp1" reason="cannot connect to given socket"/>
  </event>
```

Once a player has started, the TCP socket must provide at least enough data to keep a timeslot filled, i.e. 8000 Byte/s. The GTH provides about 2 seconds of buffering. The TCP stack on the controller provides additional buffering, the amount is dependent on the controller's operating system and configuration, but 8 – 16 seconds is typical. If the buffers underrun, the GTH sends an `event`:

```
⇐ <event>
    <fatality id="strp3" reason="underrun"/>
  </event>
```

The opposite to underrun, overrun, can never happen with a player since TCP flow control will block the sending socket.

See also

The *voice coding* attribute of the `board` resource (see 4.10) can be set to either alaw or mulaw voice coding. This setting affects conferences. If the network uses alaw but the GTH is configured for mulaw, or vice versa, conferences will sound so bad that it's almost impossible to hear what the participants are saying.

A player does not do any alaw/mulaw conversion.

The *auto conferences* attribute of the `board` resource (see 4.10) can be set to disable conferences.

Example: Playing a TCP stream on an E1/T1 Timeslot

⇒
```
<new>
  <player>
    <tcp_source ip_addr="172.16.2.1" ip_port="2222"/>
    <pcm_sink span="3A" timeslot="1"/>
  </player>
</new>
```

⇐
```
<job id="strp9"/>
```

Almost every application which uses recorded audio can use this type of TCP streaming to feed out the data on an E1/T1 timeslot.

A voicemail system can use it to play back recorded messages.

An IVR system can use it to play voice prompts

On-hold music can be streamed for indefinite periods. A simple on-hold music system could be implemented like this on a Unix server:

```
netcat -l -p 2222 < my_onhold_music.alaw
```

Netcat is a freely available package. A slightly more complex command line would allow on-the-fly MP3 decoding.

Example: Playing audio clips previously stored on the GTH

An alternative to streaming in audio over a TCP socket is to play short clips of audio which were previously stored on the GTH, like this:

⇒
```
<new>
  <clip id="DTMF 9"/>
</new>
```

⇐
```
<job id="clip DTMF 9"/>
```

(The audio data is sent by the client immediately after the end of the `new` command, in a block of content with content type `binary/audio`.)

To play the clip:

⇒
```
<new>
  <player>
    <clip id="clip DTMF 9"/>
```

```
      <pcm_sink span="4B" timeslot="17"/>
    </player>
  </new>
```

```
⇐ <job id="play7431"/>
```

Use the optional *loop* parameter to loop the audio from a clip forever—this is useful for continuous or repeating tones.

It's also possible to specify several clips. The GTH will join them seamlessly:

```
⇒ <new>
    <player>
      <clip id="clip DTMF 9"/>
      <clip id="clip 100ms silence"/>
      <clip id="clip DTMF 8"/>
      <pcm_sink span="4B" timeslot="17"/>
    </player>
  </new>
```

```
⇐ <job id="play7432"/>
```

Example: Error condition: attempting to play from an unreachable IP

In this example, 128.250.22.3 is not reachable from the example GTH:

```
⇒ <new>
    <player>
      <tcp_source ip_addr="128.250.22.3" ip_port="1234"/>
      <pcm_sink span="1A" timeslot="9"/>
    </player>
  </new>
```

```
⇐ <error reason="bad argument">
    cannot connect to given socket
  </error>
```

Example: Error condition: attempting to play from a non-listening port

In this example, the server at 172.16.2.1 has not established a listening socket on port 1234:

```
⇒ <new>
    <player>
      <tcp_source ip_addr="172.16.2.1" ip_port="1234"/>
      <pcm_sink span="1A" timeslot="9"/>
    </player>
  </new>
```

```
⇐ <job id="strp33"/>
```

```
⇐ <event>
    <fatality id="strp33" reason="cannot connect to given socket"/>
  </event>
```

3.23 new raw_monitor

```
⇒ <new>
        <raw_monitor
          id=string
          ip_addr=int.int.int.int
          ip_port='0'
          tag='0'
          format='plain'|'unpacked_4x16'>
            <pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
        </raw_monitor>
   </new>
```

⇐ <job **id**=string/>

A raw monitor extracts a bit-exact copy of the data on a 64 kbit/s timeslot or 8/16/32 kbit/s subrate timeslot and forwards it to a server over TCP. The data is sent in fixed-length packets with the same header as other signalling monitoring:

format: `unpacked_4x16` re-arranges the timeslot data into four equal-sized blocks. The first block consists of the data on the first subrate 16 kbit/s channel in the timeslot. The second, third and fourth blocks contain the remaining subrate channels, respectively. 4x16 is an experimental feature.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
octet 0x00	Length															
octet 0x02	Tag															
octet 0x04	protocol$_1$ = 7			reserved					protocol$_2$ = 4			reserved				
octet 0x06 octet 0x08 octet 0x0a	Timestamp															

length: indicates the number of octets following the length field, including the rest of the header.

tag: is specified by the user. It is used for identifying the channel.

timestamp: Timestamp marks the instant of time the packet ended, measured in milliseconds since the Unix epoch. It is a monotonically increasing integer.

Error reasons

`refused`: The GTH is already decoding signalling at maximum capacity.

`refused`: The GTH already has the maximum number (12) of signalling sockets open.

`conflict`: The GTH is already decoding a different, incompatible protocol on the given input timeslots.

See also

The `recorder` command, described on p. 76.

Example: A raw monitor on a 64 kbit/s timeslot

```
⇒   <new>
      <raw_monitor tag="1182" ip_addr="172.16.2.1" ip_port="1234">
        <pcm_source span="2A" timeslot="1"/>
      </raw_monitor>
    </new>
⇐   <job id="ramo7219"/>
```

Example: A raw monitor on a 16 kbit/s subrate timeslot

To monitor 8, 16 and 32 kbit/s subrate channels within a 64 kbit/s timeslot, specify the a *first_bit* and *bandwidth* in the `pcm_source`.

```
⇒   <new>
      <raw_monitor tag="1183" ip_addr="172.16.2.1" ip_port="1234">
        <pcm_source span="2A" timeslot="1" first_bit="2" bandwidth="16"/>
      </raw_monitor>
    </new>
⇐   <job id="ramo7220"/>
```

Subrate channels must be aligned. A 16 kbit/s channel must start at bit positions 0, 2, 4 and 6. A 32 kbit/s subrate channel must start at bit position 0 and 4.

Example: A raw monitor which delivers 4 subrate 16 kbit/s channels

```
⇒   <new>
      <raw_monitor tag="1184" ip_addr="172.16.2.1" ip_port="1234"
  format="unpacked_4x16">
        <pcm_source span="2A" timeslot="1"/>
      </raw_monitor>
    </new>
```

Example: A raw monitor on a 32 kbit/s timeslot

```
⇒   <new>
      <raw_monitor tag="1182" ip_addr="172.16.2.1" ip_port="1234">
        <pcm_source span="2A" timeslot="1" first_bit="4" bandwidth="32"/>
      </raw_monitor>
    </new>
⇐   <job id="ramo7221"/>
```

Example: A raw monitor on a 16 kbit/s timeslot

```
⇒   <new>
      <raw_monitor tag="1182" ip_addr="172.16.2.1" ip_port="1234">
        <pcm_source span="2A" timeslot="1" first_bit="2" bandwidth="16"/>
      </raw_monitor>
    </new>
⇐   <job id="ramo7221"/>
```

Example: A raw monitor on an 8 kbit/s timeslot

```
⇒   <new>
      <raw_monitor tag="1182" ip_addr="172.16.2.1" ip_port="1234">
        <pcm_source span="2A" timeslot="1" first_bit="3" bandwidth="8"/>
      </raw_monitor>
    </new>
⇐   <job id="ramo7222"/>
```

3.24 new recorder

⇒ <new>
<recorder>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
<tcp_sink **ip_addr**=int.int.int.int **ip_port**=int/>
</recorder>
</new>

⇒ <new>
<recorder>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
<udp_sink **ip_addr**=int.int.int.int **ip_port**=int/>
</recorder>
</new>

⇐ <job **id**=string/>

A `recorder` streams data from an E1/T1/J1 timeslot to a remote socket via TCP or UDP. The recording is stopped by deleting the `recorder` or, less cleanly, by closing the socket.

To use a `recorder` with a TCP destination:

1. Establish a *listening* TCP socket on the remote system, for instance the controller.
2. Issue the XML command to start a recorder. The GTH will now connect to the specified TCP socket.
3. On the remote system, accept the inbound connection.
4. On the remote system, read the audio data as it comes in.

To use a `recorder` with a UDP destination:

1. Open a UDP socket on the remote system, for instance the controller.
2. Issue the XML command to start a recorder. The GTH will now start sending UDP packets.
3. On the remote system, read the audio data as it comes in.

Data format

When using TCP, no headers or padding bytes are added. When using UDP, each UDP packet looks like this:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
octet 0x00	Reserved															
octet 0x02	Sequence															
octet 0x04	Reserved															
octet 0x06	Reserved															
octet 0x08	Reserved															
octet 0x0a	Reserved															
octet 0x0c	payload								payload							
octet ...	payload								payload							

`Sequence`: Increases by one for each packet sent for this job. The initial value is undefined.

Error handling

Recorders can encounter errors both during initial startup and while the recorder is running. If the error is detected early, the GTH returns an `error` response. If the error is detected after a `job_id` was returned, the GTH indicates errors by sending a `fatality`.

The possible *reason* attributes in an `error` response are

`refused`: The GTH does not currently have enough capacity to start another recorder.

When a recorder fails *after* starting, the GTH sends a `fatality`, for instance

```
⇐ <event>
   <fatality id="strr5" reason="overrun"/>
  </event>
```

The recorder-specific reasons for a `fatality` are:

overrun: the buffers on the GTH and in the TCP socket filled up, so the GTH terminated the job. The GTH provides about two seconds of buffering, the TCP stack on the remote system provides 8–16 seconds, depending on OS and configuration.

An overrun is normally the result of the remote system falling behind in reading from the socket, but can also be caused by a lossy or congested IP network between the GTH and the remote system.

This error cannot happen when UDP is used.

socket problem: the socket was closed by the remote server.

cannot connect to given socket: A `<new><recorder>...` command returns a `job` immediately. The TCP socket carrying the timeslot data may complete the connection process *after* the `job` is returned.

If the TCP connection attempt fails *after* the `job` has already been issued, the GTH sends a `fatality` with the reason `cannot connect to given socket`.

This error cannot happen when UDP is used.

Example: Recording an E1/T1/J1 timeslot to a TCP socket

```
⇒    <new>
       <recorder>
          <pcm_source span="3A" timeslot="1"/>
          <tcp_sink ip_addr="172.16.2.1" ip_port="2221"/>
       </recorder>
     </new>
⇐    <job id="strr11"/>
```

Example: Forwarding an E1/T1/J1 timeslot to a TCP socket at an unreachable address

```
⇒    <new>
       <recorder>
          <pcm_source span="3A" timeslot="1"/>
          <tcp_sink ip_addr="128.250.22.3" ip_port="2222"/>
       </recorder>
     </new>
⇐
      <error reason="bad argument">cannot connect to given socket</error>
```

The GTH determined that 128.250.22.3:2222 was not reachable and returned an error immediately.

Example: Forwarding an E1/T1/J1 timeslot to a TCP socket at an unused port number

```
⇒    <new>
       <recorder>
          <pcm_source span="3A" timeslot="1"/>
          <tcp_sink ip_addr="172.16.2.1" ip_port="2222"/>
       </recorder>
     </new>
⇐    <job id="strr12"/>
⇐    <event>
        <fatality id="strr12" reason="cannot connect to given socket"/>
     </event>
```

The GTH issued a job id and initiated an attempt to connect to TCP port 2222, but then failed to connect, so it issued a `fatality`.

3.25 new ss5_linesig_monitor

⇒ <new>
<ss5_linesig_monitor tag='0' **ip_addr**=int.int.int.int **ip_port**=int>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
</ss5_linesig_monitor>
</new>

⇐ <job **id**=string/>

SPECIFIC FIRMWARE REQUIRED.

SS5 signalling monitoring requires a firmware upgrade which is available on request. The firmware includes an API description for SS5.

3.26 new ss5_registersig_monitor

⇒ <new>
<ss5_registersig_monitor tag='0' **ip_addr**=int.int.int.int **ip_port**=int>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
</ss5_registersig_monitor>
</new>

⇐ <job **id**=string/>

SPECIFIC FIRMWARE REQUIRED.

SS5 signalling monitoring requires a firmware upgrade which is available on request. The firmware includes an API description for SS5.

3.27 new tone_detector

⇒ <new>
<tone_detector type='DTMF'|'custom' frequency=int length=int>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
</tone_detector>
</new>

⇐ <job **id**=string/>

A tone detector detects DTMF ("touch-tone") and other types of in-band signalling on a timeslot.

type: **DTMF** detects DTMF tones. This is the default. The frequency and length attributes should not be specified for DTMF.

type: **custom** detects tones of user-selectable *frequency* (in Hertz, useable range is 500–3500 Hz) and minimum *length* (in milliseconds, from 20ms to 2000ms). A custom detector can detect signals down to -30 dBm0.

Whenever a tone is detected, the GTH sends an `event` to the control socket which created the detector:

⇐ <event>
<tone **detector**=string **name**=string **length**=int/>
</event>

Error reasons

`refused`: The GTH does not currently have enough capacity to start another tone detector.

See also

The *voice coding* attribute of the `board` resource (see 4.10) can be set to either alaw or mulaw voice coding.

Standards: ITU-T Q.23, Q.24

Example: DTMF Tone Detection

⇒
```
<new>
  <tone_detector type="DTMF">
    <pcm_source span="2A" timeslot="13"/>
  </tone_detector>
</new>
```

⇐
```
<job id="tode9613"/>
```

When a tone is detected, the GTH sends a message like this:

```
⇐     <event>
         <tone detector="tode9613" name="9" length="81"/>
      </event>
```

The tone length is approximate, typically +/- 20ms.

Example: Tone detection interleaved with a command

```
⇒     <query>
        <resource name="cpu"/>
      </query>
⇐     <event>
        <tone detector="tode3" name="3" length="80"/>
      </event>
⇐     <state>
      ...
      </state>
```

This example shows how asynchronously generated events can arrive at any time at all, including while a command is pending. Controllers must be able to deal with this situation.

Example: Detecting a CED fax tone.

A terminating (answering) fax machine always transmits a 2100 Hz tone, the CED signal, for at least 2600ms. This example uses a length of 1800ms to provide an 800ms margin to carry out an action in response to detecting a fax call.

```
⇒     <new>
        <tone_detector type="custom" frequency="2100" length="1800">
          <pcm_source span="2A" timeslot="13"/>
        </tone_detector>
      </new>
⇐     <job id="tode83216"/>
```

When a tone is detected, the GTH sends a message like this:

```
⇐     <event>
         <tone detector="tode83216" name="2100" length="1809"/>
      </event>
```

Example: Detecting a CNG signal.

Some originating fax machines send the CNG signal at the start of the call.

```
⇒     <new>
        <tone_detector type="custom" frequency="1800" length="400">
          <pcm_source span="2A" timeslot="13"/>
        </tone_detector>
      </new>
```

⇐ `<job id="tode83216"/>`

3.28 new v110_monitor

⇒ <new>
<v110_monitor
id=string
ip_addr=int.int.int.int
ip_port='0'
tag='0'
rate='4800'|'9600'
ra0='no'>
<pcm_source span=string timeslot=int first_bit='0' bandwidth='64'/>
</v110_monitor>
</new>

⇐ <job **id**=string/>

V.110 monitor performs the RA1 and RA2 decoding for 4800 and 9600 bit/s data carried in 64 kbit/s timeslots. The monitor forwards the decoded data to a server over TCP, using the same header format as other signalling monitoring:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
octet 0x00	Length															
octet 0x02	Tag															
octet 0x04	$\text{protocol}_1 = 7$			reserved					$\text{protocol}_2 = 5$			reserved				
octet 0x06 octet 0x08 octet 0x0a	Timestamp															

length: indicates the number of octets following the length field, including the rest of the header.

tag: is specified by the user. It is used for identifying the channel.

rate: specifies the data signalling rate, either 4800 or 9600 bit/s.

timestamp: Timestamp marks the instant of time the packet ended, measured in milliseconds since the Unix epoch. It is a monotonically increasing integer.

ra0: Setting this to `yes` enables start/stop bit removal.

Error reasons

`refused`: The GTH is already decoding signalling at maximum capacity.

`refused`: The GTH already has the maximum number (12) of signalling sockets open.

`conflict`: The GTH is already decoding a different, incompatible protocol on the given input timeslots.

Standards: ITU-T V.110, I.460, V.14

Example: Creating a 4800 bit/s V.110 monitor on a 64 kbit/s timeslot.

```
⇒     <new>
        <v110_monitor tag="1182"
                      ip_addr="172.16.2.1"
                      ip_port="1234"
                      rate="4800">
          <pcm_source span="2A" timeslot="1" first_bit="0" bandwidth="8"/>
        </v110_monitor>
      </new>
⇐     <job id="v1mo7219"/>
```

This example also demonstrates that the GTH uses zero-based bit numbering in the pcm_source, unlike I.460.

Example: RA0 enabled on a 9600 bit/s V.110 monitor.

```
⇒     <new>
        <v110_monitor tag="1182"
                      ip_addr="172.16.2.1"
                      ip_port="1234"
                      rate="9600"
                      ra0="yes">
          <pcm_source span="2A" timeslot="1" first_bit="6" bandwidth="16"/>
        </v110_monitor>
      </new>
⇐     <job id="v1mo7221"/>
```

For 9600 bit/s, the firmware uses two contiguous bits from the input data. This implies that I.460 flexible timeslot assignment (arbitrary bits) in RA2 is not supported.

3.29 new wide_recorder

⇒ <new>
    <wide_recorder **span**=string **tag**=int>
        <udp_sink **ip_addr**=int.int.int.int **ip_port**=int/>
    </wide_recorder>
</new>

⇐ <job **id**=string/>

PRELIMINARY INFORMATION.

'wide recorder' is implemented for field testing on E1/T1 Monitor 3.0 and SDH/SONET Monitor 3.0. This interface may change when adopted into the supported API.

A wide_recorder streams one *entire* E1/T1 to a remote system via UDP. Deleting the recorder stops the recording. Only *one* `wide_recorder` can be run at a time.

To use a `wide_recorder`:

1. Establish a UDP socket on the remote system, for instance the controller.
2. Issue the XML command to start a wide_recorder.
3. On the remote system, read the audio data as it comes in.

When transmitting from an E1 line, each UDP packet payload looks like this:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
octet 0x00	Tag															
octet 0x02	Reserved															LD
octet 0x04	Sequence Number															
octet 0x06 octet 0x08 octet 0x0a	Timestamp															
octet 0x0c	TS0								TS1							
octet 0x0e	TS2								TS3							
...	...															
octet 0x2a	TS30								TS31							
octet 0x2c	TS0								TS1							
...	...															

Tag: The user-supplied value given in the XML command to start the wide_recorder.

LD: set to 1 if the E1/T1 is disabled or in any of the LOS, AIS, LMFA or LFA states. 0 in normal operation, i.e. if the E1/T1 is in the OK or RAI state.

Sequence Number: A 16-bit sequence number which increments by one for each packet.

Timestamp: Timestamp marks the instant of time the E1/T1 data in this packet started, measured in milliseconds since the Unix epoch.

TS0, TS1, ...: The timeslot data on the E1, presented frame by frame.

A new UDP packet is sent every 32 frames, i.e. 250 packets per second.

When transmitting from a T1 line, there are fewer timeslots and so the packet looks like this instead:

<table>
<tr><th>Bit</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>octet 0x00</td><td colspan="16">Tag</td></tr>
<tr><td>octet 0x02</td><td colspan="15">Reserved</td><td>LD</td></tr>
<tr><td>octet 0x04</td><td colspan="16">Sequence Number</td></tr>
<tr><td>octet 0x06
octet 0x08
octet 0x0a</td><td colspan="16">Timestamp</td></tr>
<tr><td>octet 0x0c</td><td colspan="7">Reserved</td><td>F</td><td colspan="8">TS1</td></tr>
<tr><td>octet 0x0e</td><td colspan="8">TS2</td><td colspan="8">TS3</td></tr>
<tr><td>...</td><td colspan="16">...</td></tr>
<tr><td>octet 0x24</td><td colspan="8">TS24</td><td colspan="8">TS0</td></tr>
<tr><td>octet 0x26</td><td colspan="8">TS1</td><td colspan="8">TS2</td></tr>
<tr><td>...</td><td colspan="16">...</td></tr>
</table>

F: The F-bit in T1

See also

The `recorder` command, described on p. 76.

Example: Recording an entire E1/T1 to UDP

```
⇒   <new>
      <wide_recorder span="3A" tag="39">
         <udp_sink ip_addr="172.16.2.1" ip_port="2221"/>
      </wide_recorder>
    </new>
⇐   <job id="strw11"/>
```

3.30 nop

⇒ <nop/>

⇐ <ok/>

The `nop` (no-operation) command does nothing. The GTH responds immediately with an `ok`.

Applications which want to supervise the GTH, i.e. detect a fault in the GTH or the ethernet connecting it, can send periodic `nop` commands to check that the GTH is responding.

See also

Section 5.4 describes how to use the `nop` command to supervise a GTH

3.31 query

⇒ <query verbose='false'>
<job **id**=string/>
...
</query>

⇒ <query verbose='false'>
<resource **name**=string/>
...
</query>

⇐ <state>
<atm_aal0_monitor ... >
<atm_aal0_layer ... >
<atm_aal2_monitor ... >
<atm_aal5_monitor ... >
<cas_r2_linesig_monitor ... >
<cas_r2_mfc_detector ... >
<clip ... >
<connection ... >
<controller ... >
<error ... >
<fr_layer ... >
<fr_monitor ... >
<job ... >
<lapd_layer ... >
<lapd_monitor ... >
<level_detector ... >
<mtp2_monitor ... >
<player ... >
<raw_monitor ... >
<recorder ... >
<resource ... >
<ss5_linesig_monitor ... >
<ss5_registersig_monitor ... >
<tone_detector ... >
<v110_monitor ... >
<wide_recorder ... >
</state>

`query` is used to obtain information about one or more of the GTH's resources and jobs.

The response to a `query` always contains exactly as many children (answers) as there were jobs and resources in the command. If one or more of the jobs or resources in the command result in an error, the corresponding child will be an `error`, but the query command itself will still succeed.

Verbose queries

Setting the *verbose* attribute to *true* makes the GTH provide detailed information about each *job*. The return includes all the information needed to start an identical job, in the same format as for the corresponding `new` command.

Jobs with counters, for example `mtp2_monitor` then add additional `attribute` children for counters and indicators related to the job.

Example: Monitoring temperature (the temperature is in degrees Celsius)

The temperature sensor is located on the warmest part of the module.

```
⇒     <query>
        <resource name="board"/>
      </query>
```

```
⇐     <state>
        <resource name="board">
          ...
          <attribute name="temperature" value="32.4"/>
          <attribute name="power consumption" value="6.8"/>
          ...
        </resource>
      </state>
```

Example: Finding out which resources a GTH has

```
⇒     <query>
         <resource name="inventory"/>
      </query>
```

```
⇐     <state>
        <resource name="sync"/>
        <resource name="cpu"/>
        <resource name="board"/>
        <resource name="os"/>
        <resource name="system_image"/>
        <resource name="failsafe_image"/>
        <resource name="application_log"/>
        <resource name="system_log"/>
```

```
    <resource name="eth1"/>
    <resource name="eth2"/>
    <resource name="http_server"/>
    <resource name="pcm1A"/>
    <resource name="pcm1B"/>
    ...
  </state>
```

Example: Reading the counters for an SDH/SONET hierarchy

```
⇒ <query>
    <resource name="sdh1"/>
  </query>
⇐ <state>
    <resource name="sdh1">
      <attribute name="STM" value="1"/>
      <attribute name="AU" value="4"/>
      <attribute name="TU" value="12"/>
      <attribute name="SONET" value="false"/>
      <attribute name="status" value="OK"/>
      <attribute name="OK_entered" value="1"/>
      <attribute name="OK_duration" value="46226"/>
      <attribute name="RS-LOF_entered" value="1"/>
      <attribute name="RS-LOF_duration" value="11"/>
      <attribute name="RS-BIP" value="0"/>
      <attribute name="MS-BIP" value="0"/>
      <attribute name="MS-REI" value="0"/>
      <attribute name="child" value="sdh1:hop1_1"/>
      ...
    </resource>
  </state>
```

Example: Reading the counters for an SDH/SONET HOP resource

```
⇒ <query>
    <resource name="sdh1:hop1_3"/>
  </query>
⇐ <state>
    <resource name="sdh1:hop1_3">
      <attribute name="status" value="OK"/>
      <attribute name="OK_entered" value="1"/>
      <attribute name="OK_duration" value="8335621"/>
      ...
      <attribute name="child" value="sdh1:hop1_3:lop1_1"/>
      <attribute name="child" value="sdh1:hop1_3:lop1_2"/>
      <attribute name="child" value="sdh1:hop1_3:lop1_3"/>
```

```
    ...
  </resource>
</state>
```

Example: Reading the counters for an SDH/SONET LOP resource

```
⇒    <query>
       <resource name="sdh1:hop1_3:lop1_1"/>
     </query>
⇐    <state>
       <resource name="sdh1:hop1_3:lop1_1">
         <attribute name="status" value="OK"/>
         <attribute name="OK_entered" value="1"/>
         <attribute name="OK_duration" value="733623864"/>
         ...
         <attribute name="trace_identifier" value="MSC_south"/>
       </resource>
     </state>
```

Example: Looking up the job ID of *this* API connection.

In a sophisticated system with fail-over support, a controller needs to be able to identify an API socket. The special query of the job called *self* reveals that:

```
⇒    <query>
       <job id="self"/>
     </query>
⇐    <state>
       <job id="apic232"/>
     </state>
```

Example: Finding out which jobs a GTH is currently running

```
⇒    <query>
        <resource name="schedule"/>
     </query>
⇐    <state>
       <job id="cnxn565" owner="apic232"/>
       <job id="m2mo153" owner="apic232"/>
     </state>
```

Example: Examining the GTH's logs

The GTH contains two sets of logs: the operating system log and the control system log. These can be retrieved using queries:

⇒
```
<query>
  <resource name="application_log"/>
</query>
```

⇐
```
<state>
  <resource name="application_log">
    <attribute name="verbosity" value="changes"/>
  </resource>
</state>
```

The log is delivered as a `text/plain` block immediately following the XML block.

Example: Querying two players

The query command accepts multiple jobs (and multiple resources). This provides a quicker answer than querying each job separately.

⇒
```
<query>
  <job id="play4"/>
  <job id="play5"/>
</query>
```

⇐
```
<state>
  <player id="play4" owner="apic7" octets_played="1001728"/>
  <player id="play5" owner="apic7" octets_played="148992"/>
</state>
```

Example: Querying two players and a nonexistant resource

`bogus` is not an actual resource. The response to the query contains answers to all three items queried, in the same order as queried.

⇒
```
<query>
  <job id="play3"/>
  <resource name="bogus"/>
  <job id="play1"/>
</query>
```

⇐
```
<state>
  <player id="play3" owner="apic27" octets_played="246784"/>
  <error reason="bad argument">no such resource</error>
  <player id="play1" owner="apic27" octets_played="1267712"/>
</state>
```

Example: A verbose query

Adding `verbose="true"` provides complete information about the job, including everything needed to start an identical job:

⇒
```
<query verbose="true">
  <job id="play4"/>
```

```
    </query>
⇐   <state><player id="play2" owner="apic1" loop="true">
      <clip id="clip leave_message_after_tone"/>
      <pcm_sink span="1A" timeslot="1"/>
      <attribute name="octets_played" value="192768"/>
    </player></state>
```

Verbose queries are intended for debugging. They could also be used in migrating jobs from one system to another in a warm standby setup. If you're writing a parser for GTH responses, you might want to skip verbose queries—the return values the generate are much more complex than anything else the GTH generates.

3.32 reset

⇒ <reset>
<resource **name**=string/>
</reset>

⇐ <ok/>

The resource must be `cpu`. This command reboots the GTH, which takes about 40s.

3.33 set

⇒ <set **name**=string>
    <attribute **name**=string **value**=string/>
    ...
</set>

⇐ <ok/>

`set` configures resources. The name/value tuples ("attributes") are the same as returned by `query`.

Section 4 lists the resources, how they work and which attributes can be read and written.

Error reasons

`no such resource`: the named resource doesn't exist. A `query` command on the `inventory` returns a list of valid resources on a GTH.

`invalid attribute or attribute value`: the named resources exists, but it either doesn't have one of the attributes being set, or the attribute is being set to an impossible value.

See also

The `enable` command (Section 3.5) is used to configure E1/T1/J1 and SDH/SONET interfaces. In old releases (2012 and earlier), the `set` command was used to do the same thing, this is now deprecated.

Example: Set the IP address on the second Ethernet interface

⇒
```
<set name="eth2">
   <attribute name="IP4 address" value="10.0.0.1"/>
   <attribute name="IP4 mask" value="255.0.0.0"/>
</set>
```

Example: Explicitly set an E1/T1/J1 sync source on hardware with E1/T1/J1 ports

By default, the GTH scans its ports (lowest port number first) until it finds an E1/T1/J1 with valid sync. If the sync source is lost (e.g. the cable is unplugged), the ports are scanned again.

In most monitoring applications, the default sync behaviour is satisfactory. In other applications it's often necessary to explicitly designate one port as the sync source:

⇒
```
<set name="sync">
  <attribute name="source" value="pcm1A"/>
</set>
```

⇐
```
<ok/>
```

3.34 takeover

⇒ <takeover>
<job **id**=string/>
...
</takeover>

⇐ <ok/>

All jobs have an *owner*. Initially, the owner is the socket which used a `new` command to create the job. Takeover allows ownership to be transferred to another socket.

Any events, for instance DTMF events, are then sent to the new owner.

When a socket is closed, all the jobs it owns are deleted. A system using failover could transfer ownership of jobs from one server to another to allow a system to be taken down for maintenance without interrupting calls.

Example: Taking over ownership of a switched connection

⇒
```
<takeover>
  <job id="cnxn14"/>
</takeover>
```

⇐
```
<ok/>
```

3.35 unmap

⇒ <unmap **name**=string/>

⇐ <ok/>

Remove a previously mapped E1/T1/J1.

Only used on hardware with SDH/SONET interfaces.

See also

The `map` command.

Example: Removing a previously mapped E1/T1/J1

⇒ `<unmap name="pcm13"/>`

⇐ `<ok/>`

3.36 update

⇒ <update>
<controller timeout=int backups=string **broadcast_events**='yes'|'no'/>
</update>

⇒ <update>
<mtp2_monitor
id=string
fisu='yes'
dup_fisu='no'
lssu='yes'
dup_lssu='no'
msu='yes'
esu='no'
mark_likely_retrans='no'
load_limit='50'
buffer_limit='256000'
average_period='30'
tag='0'
esnf='no'
ip_addr=int.int.int.int
ip_port='0'>
<pcm_source/>
...
</mtp2_monitor>
</update>

⇒ <update>
<lapd_monitor
id=string
su='yes'
esu='no'
load_limit='50'
buffer_limit='256000'
average_period='30'
tag='0'
timeout='15'
detect_abort='yes'
ip_addr=int.int.int.int
ip_port='0'>
<pcm_source/>
...
</lapd_monitor>
</update>

⇐ <ok/>

Update allows a job's attributes to be changed. The changeable attributes on a controller are *timeout*, *backups* and *broadcast_events*.

timeout is in milliseconds. The special value `infinity` disables the timeout.

backups is a space-delimited list of controller job-IDs. Section 5.4 describes how to build fault-tolerant systems by setting these attributes to non-default values. If *backups* is specified, then *timeout* must also be specified in the same command.

broadcast_events controls whether or not broadcast events, for example layer 1 status changes, are sent to *this* controller. If your system uses more than one or two concurrent API connections, set *broadcast_events* to `no` on most of them to cut the amount duplicated broadcast event traffic.

Section 5.4 describes how to build fault-tolerant systems by setting the *timeout* and *backups* attributes to non-default values. If *backups* is specified, then *timeout* must also be specified in the same command.

On an `mtp2_monitor` and `lapd_monitor`, the changeable attributes are *load_limit*, *buffer_limit* and the signal unit filter settings (*fisu, dup_fisu,* etc.)

See also

Section 3.21 describes the `mtp2_monitor` and its parameters in detail.

Example: Disabling LSSU filtering

In many applications, LSSUs do not provide useful information, so the GTH can be set up to discard them. When troubleshooting faulty MTP-2 links, it is useful to disable the filter:

```
⇒   <update>
      <mtp2_monitor id="m2mo19" lssu="yes"/>
    </update>
⇐   <ok/>
```

Example: Setting a backups list on a controller

This example configures the current API socket (called a controller) to time out after 20 seconds. On timeout, instead of deleting all jobs owned by the socket, the jobs will be transferred to apic13 and apic15.

```
⇒   <update>
      <controller timeout="20" backups="apic13 apic15"/>
    </update>
⇐   <ok/>
```

3.37 zero

⇒ <zero>
<resource **name**=string/>
</zero>

⇒ <zero>
<job **id**=string/>
</zero>

⇐ <ok/>

Reset counters and timers on

- SDH/SONET resources (sdh1, sdh1:hop1_1, sdh1:hop1_1:lop1_1, ...)
- E1/T1/J1 L1 resources (pcm1A, pcm2A, ...)
- Layer 2 monitors (CAS, MTP-2, LAPD, frame relay and ATM)

Example: Zeroing the layer 1 (E1/T1/J1) counters

⇒
```
<zero>
  <resource name="pcm1A"/>
</zero>
```
⇐
```
<ok/>
```

Example: Zeroing SDH/SONET

⇒
```
<zero>
  <resource name="sdh2"/>
</zero>
```
⇐
```
<ok/>
```

This sets all the counters in sdh2 (and all of its children, i.e. sdh2:hop1_1, and so on) to zero.

4 Resources

Resources are parts of the GTH which always exist: the ethernet interfaces, the firmware, the E1/T1/J1 interfaces, the SDH/SONET interfaces and the CPU.

Four commands work on resources: `set` modifies a resource's attributes (section 3.33). `query` command (section 3.31) shows a resource's settings, counters and indicators. `enable` and `disable` commands (section 3.5) turn E1/T1/J1 and SDH/SONET interfaces on and off.

Each resource has a table of attributes and possible values. For example, here's the table for the 'sfp1' resource on SDH hardware:

Attribute	Possible values in query	Allowed values in set
status	`OK` | `LOS` | `not present`	read-only
vendor	A text string	read-only
part_number	A text string	read-only
supported_rates	`OC-12`, `OC-3`, `GbE`	read-only
tx_enabled	`true` | `false`	read-only
rx_power	A number with a decimal point	read-only

This table tells us that there are six attributes for the sfp1 resource. The *status* can be *one* of `OK`, `LOS` and `'not present'`, as indicated by the vertical bars between the possible values. The *supported_rates* can be one or more of the listed options, since some SFPs can support both OC-12 and OC-3, which is indicated by listing the values with commas.

When a resource has configurable attribute with multiple choices, the default choice is listed first.

4.1 Inventory and Schedule

A list of the GTH's resources is always available by querying the `inventory`. Different models of GTH hardware have different resources, for instance the `sfp1` resource is only present on hardware with slots for SFP modules.

A list of the currently running jobs is always available by querying the `schedule` resource.

Page 90 has examples of querying the `schedule` and `inventory` resources.

4.2 E1 links

E1 lines are the basic transmission lines, i.e. *layer 1*, used for voice, signalling and data in most fixed and mobile telephone networks in the world.

E1/T1 Monitor 3.0 systems have 64 listen-only E1/T1/J1 interfaces named `pcm1A, pcm1B, pcm1C, pcm1D, ..., pcm16D`. The names indicate which connector the E1 is on (1, 2, 3 or 4) and which pair of pins within that connector (A, B, C or D). The *Getting Started* leaflet enclosed with every system shows the pinouts.

SDH Monitor 3.0 systems have up to 84 listen-only E1/T1/J1 resources named `pcm1, pcm2, pcm3, ...`. These resources appear after they are configured using the `map` command.

E1/T1 Messenger 3.0 systems have sixteen full-duplex E1/T1/J1 interfaces named `pcm1A, pcm1B, pcm2A, ...., pcm8B`.

Configurable attributes

Attribute	Possible values
framing	`doubleframe` | `multiframe`
idle_pattern	An integer.
impedance	`120` | `75`
line_coding	`HDB3`
mode	`E1`
monitoring	`false` | `true`
tx_enabled	`true` | `false`

The *idle_pattern, impedance, line_coding, monitoring* and *tx_enabled* attributes are only on hardware which attaches to E1 lines electrically, i.e. not optically.

framing selects which framing mode is used. SS7 links usually use doubleframe and most other links use multiframe.

impedance selects the receiver and transmitter impedance, which depends on the type of cable used. E1 on twisted pair always uses 120 ohm. E1 on coaxial cable uses 75 ohm.

idle_pattern selects the octet sent on unused timeslots. The default, 84, is the most commonly used value on E1 lines.

mode selects E1 or T1 operation.

On SDH hardware, the mode may be chosen freely for each interface.

On *E1/T1 Monitor 3.0*, there are four blocks of interfaces: 1A–4D, 5A–8D, 9A–12D and 13A–16D. The *mode* must be the same for all 16 interfaces within a block.

monitoring is used for non-intrusive monitoring of links, typically through a protected monitor point as per ITU-T G.772. Enabling monitoring increases the signal sensitivity by 20dB and also implicitly sets the *tx_enabled* attribute to `false`.

tx_enabled When set to `false`, transmit is disabled for that E1/T1/J1. On hardware which shares pins for transmit and receive, attempting to enable all four E1/T1/J1 receivers in one RJ45 connector without setting *tx_enabled* to `false` will result in a `conflict` error.

Read-only attributes

Attribute	Possible values
status	`disabled` | `LOS` | `LFA` | `LMFA` | `AIS` | `RAI` | `OK`
slip_positive	The number of times an extra frame was inserted.
slip_negative	The number of times a frame was skipped.
frame_error	The number of times a framing error occurred.
code_violation	Unused, present for historical reasons.
code_violation_seconds	The number of seconds in which there was at least one violation of the line coding rules.
crc_error	The number of L1 CRC errors. Some framing modes, for instance *doubleframe* do not have an L1 CRC, in those modes this counter is always zero.
LFA_entered	The number of times the LFA state was entered. There is a similar counter for each of the other possible states.
LFA_duration	The cumulative number of milliseconds the link was in the LFA state. There are analogous counters for each of the other possible states.
origin	An SDH LOP resource, only present when the E1/T1/J1 comes from an SDH/SONET interface. It indicates which LOP the E1/T1/J1 is mapped from.

status shows the link's current state, which is either `OK` or one of several error states:

State name	Notes
LOS	Loss of signal. The incoming line doesn't have a signal at all, or a signal too weak to detect. This state does not exist on *SDH Monitor 3.0*.
LFA	Loss of frame alignment. There is a signal on the line, but E1 framing cannot be recovered.
LMFA	Loss of multiframe alignment. There is a signal on the line, framing can be recovered, but not multiframe framing.
AIS	Alarm indication signal. The E1 receiver in the GTH has detected a fault on the signal it is receiving (too many 1-bits in a frame).
RAI	Remote alarm indication. The equipment at the other end of the link is signalling that *it* is not in the OK state.

Each E1/T1/J1 resource broadcasts an event to all API sockets whenever its state changes:

```
⇐ <event>
      <l1_message name="pcm1A" state="OK"|"LOS"|"LFA"|"LMFA"|"AIS"|"RAI"/>
  </event>
```

Whenever an E1/T1/J1 interface *slips*, it broadcasts an event:

```
⇐ <event>
      <slip name="pcm3A"/>
  </event>
```

See also

The `enable` command turns an E1/T1/J1 link on, with default values for all attributes which aren't specified in the command. Section 3.5 has an example of how to enable an interface in T1 mode and E1 mode.

The `disable` command turns an E1/T1/J1 link off.

4.3 T1/J1 links

T1 lines are the basic transmission lines, i.e. *layer 1*, used for voice, signalling and data in North America. J1 lines are used in Japan, they are similar to T1 lines.

E1/T1 Monitor 3.0 systems have 64 listen-only E1/T1/J1 interfaces named `pcm1A, pcm1B, pcm1C, pcm1D, ..., pcm16D`. The names indicate which connector the T1/J1 is on (1, 2, 3 or 4) and which pair of pins within that connector (A, B, C or D). The *Getting Started* leaflet enclosed with every system shows the pinouts.

SDH Monitor 3.0 systems have up to 84 listen-only E1/T1/J1 resources named `pcm1, pcm2, pcm3, ....` These resources appear after they are configured using the `map` command.

E1/T1 Messenger 3.0 systems have sixteen full-duplex E1/T1/J1 interfaces named `pcm1A, pcm1B, pcm2A, ...., pcm8B`.

Configurable attributes

Attribute	Possible values
framing	`extended superframe` | `superframe`
idle_pattern	An integer.
impedance	`100` | `110`
line_coding	`B8ZS` | `AMI`
mode	`T1` | `J1`
monitoring	`false` | `true`
tx_enabled	`true` | `false`

The *idle_pattern, impedance, line_coding, monitoring* and *tx_enabled* attributes are only on hardware which attaches to T1 lines electrically, i.e. not optically.

framing selects which framing mode is used. On T1/J1 lines, the valid options are the two `superframe` choices.

impedance selects the receiver and transmitter impedance. T1 always uses 100 ohm. J1 always uses 110 ohm.

idle_pattern selects the octet sent on unused timeslots. The default, 127, is common on T1/J1 lines in practice.

mode is used to select E1, T1 or J1 operation. T1 is mainly used in North America. J1 is used in Japan.

On SDH hardware, the mode may be chosen freely for each interface.

On *E1/T1 Monitor 3.0*, there are four blocks of interfaces: 1A–4D, 5A–8D, 9A–12D and 13A–16D. The *mode* must be the same for all 16 interfaces within a block.

monitoring is used for non-intrusive monitoring of links, typically through a protected monitor point as per ITU-T G.772. Enabling monitoring increases the signal sensitivity by 20dB and also implicitly sets the *tx_enabled* attribute.

tx_enabled When set to `false`, transmit is disabled for that E1/T1/J1. On hardware which shares pins for transmit and receive, attempting to enable all four E1/T1/J1 receivers in one RJ45 connector without setting *tx_enabled* to `false` will result in a `conflict` error.

Read-only attributes

Attribute	Possible values
status	`disabled` | `LOS` | `LFA` | `LMFA` | `AIS` | `RAI` | `OK`
slip_positive	The number of times an extra frame was inserted.
slip_negative	The number of times a frame was skipped.
frame_error	The number of times a framing error occurred.
code_violation	Unused, present for historical reasons.
code_violation_seconds	The number of seconds in which there was at least one violation of the line coding rules.
crc_error	The number of L1 CRC errors. Some framing modes, for instance *doubleframe* do not have an L1 CRC, in those modes this counter is always zero.
LFA_entered	The number of times the LFA state was entered. There is a similar counter for each of the other possible states.
LFA_duration	The cumulative number of milliseconds the link was in the LFA state. There are analogous counters for each of the other possible states.
origin	An SDH LOP resource, only present when the E1/T1/J1 comes from an SDH/SONET interface. It indicates which LOP the E1/T1/J1 is mapped from.

status shows the link's current state, which is either `OK` or one of several error states:

State name	Notes
LOS	Loss of signal. The incoming line doesn't have a signal at all, or a signal too weak to detect. LOS, LFA and LMFA are often called a *red alarm*. This state does not exist on *SDH Monitor 3.0*.
LFA	Loss of frame alignment. There is a signal on the line, but E1 or T1 framing cannot be recovered.
LMFA	Loss of multiframe alignment. There is a signal on the line, framing can be recovered, but not multiframe framing.
AIS	Alarm indication signal. The T1 receiver in the GTH has detected a fault on the signal it is receiving (too many 1-bits in a frame). This is often called a *blue alarm*.
RAI	Remote alarm indication. The equipment at the other end of the link is signalling that *it* is not in the OK state. This is often called a *yellow alarm*.

Each E1/T1/J1 resource broadcasts an event to all API sockets whenever its state changes:

```
⇐ <event>
      <l1_message name="pcm1A" state="OK"|"LOS"|"LFA"|"LMFA"|"AIS"|"RAI"/>
  </event>
```

Whenever an E1/T1/J1 interface *slips*, it broadcasts an event:

```
⇐ <event>
      <slip name="pcm3A"/>
  </event>
```

See also

The `enable` command turns an E1/T1/J1 link on, with default values for all attributes which aren't specified in the command. Section 3.5 has an example of how to enable an interface in T1 mode and E1 mode.

The `disable` command turns an E1/T1/J1 link off.

4.4 sfp1 and sfp2

The sfp1 and sfp2 resources are only present on hardware with slots for SFP modules.

An SFP module is an adaptor which converts a particular type of signal, usually optical from a fiber, to a signal the GTH hardware can process. Different types of SFPs work with different fiber types, speeds, connectors and wavelengths.

Read-only attributes

Attribute	Possible values
status	`OK` | `LOS` | `not present`
vendor	A text string
part_number	A text string
supported_rates	`OC-12`, `OC-3`, `GbE`
tx_enabled	`true` | `false`
rx_power	A number with a decimal point

The *status* attribute is normally `OK` in a functioning system. `LOS` indicates that the incoming signal is too weak for the SFP to reliably recover the data. `not present` indicates that no SFP is inserted.

The *vendor* and *part_number* attributes are strings set by the SFP manufacturer.

The *rx_power* is the received optical power in microwatts, as reported by the SFP's digital diagnostics interface, if available. Accuracy depends on the SFP, $\pm$ 3 dB is common. To convert to 'dBm', use the definition of the decibel scale:

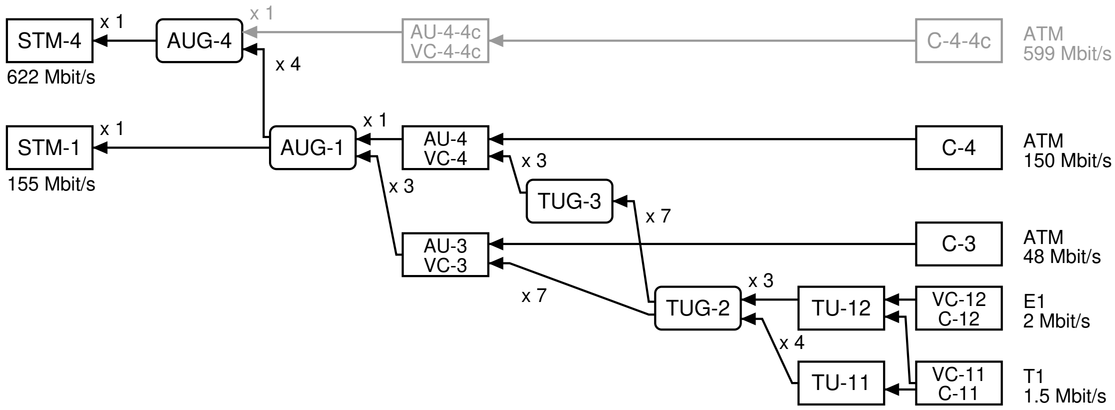
$$10 \times \log_{10} \frac{rx\_power}{1000}$$

The SFP resources broadcast an event to all API sockets when their state changes:

```
⇐ <event>
      <sfp_message name="sfp2" state="OK"|"LOS"|"not present"/>
  </event>
```

4.5 SDH Links

SDH is used on 155 Mbit/s links, usually optical. GTH modules with SFPs support SDH, which in turn can carry either a few ATM links or many E1/T1/J1 links. GTH modules with electrical E1/T1/J1 interfaces do not support SDH.



The SDH multiplexing structure

The `enable` command enables data receiption on an SDH link.

Each SDH link has resources at three of the levels in the multiplexing structure shown in the diagram above. The top of the hierarchy is called `sdh1` and `sdh2` (section 4.5.1). The level at AU-3/AU-4 is described in 4.5.2. The level at C-11/C-12 is described in 4.5.3.

GTH supports both ATM and E1/T1/J1 on SDH. ATM is carried in *C-4* or *C-3*, while E1 links are carried in *C-12* containers, shown on the right side of the diagram above. T1 and J1 links are carried in *C-11* containers. The `map` command makes an E1/T1/J1 carried in one of those containers accessible to other commands.

4.5.1 sdh1 and sdh2

`sdh1` represents the SDH structure associated with `sfp1`.

`sdh2` and `sfp2` are similarly associated.

Configurable attributes

Attribute	Possible values
STM	`1`
AU	`4` | `3`
TU	`12` | `11`
SONET	`false`
payload	`TUG` | `ATM`
daisy_chain	`false` | `true`

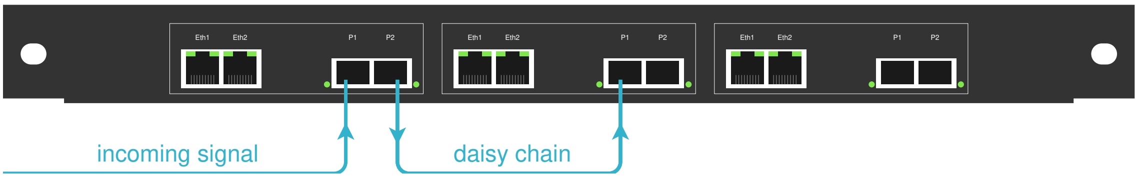
STM sets the STM level. `1` means the optical link operates at 155Mbit/s.

AU and *TU* select which of several possible paths through the SDH multiplexing structure are used.

SONET selects which names are used for attributes, states, counters and events. By default, the names used in the SDH standard are used. If *SONET* is `true`, names corresponding to the SONET standard are used instead (see 4.6).

payload selects the type of payload. The default, `TUG`, indicates that there is a *lower order path* present, carrying E1 or T1 lines. `ATM` is used to carry ATM. All SDH interfaces must be configured for the same payload.

daisy_chain controls whether or not the incoming data on this SDH resource is re-transmitted, false by default. Daisy-chaining occurs cross-wise, i.e. enabling *daisy_chain* on sdh1 means that the data received by sdh1 is re-transmitted on sdh2. We recommend daisy-chaining a signal no more than three times.



Cross-wise daisy-chaining

Read-only attributes

Attribute	Possible values
status	`OK` | `RS-LOF` | `MS-AIS` | `MS-RDI`
trace_identifier	A text string
_entered	The number of times the state was entered. There are four of these counters, one for each state.
_duration	The number of milliseconds the link has been in the state, in total. There are four of these counters.
RS-BIP	The RS-BIP counter (Regenerator Section, Bit Interleaved Parity).
MS-BIP	The MS-BIP counter (Multiplex Section, Bit Interleaved Parity).
MS-REI	The MS-REI counter (Multiplex Section, Remote Error Indication).
child	A text string

The *status* attribute can be in one of four states:

`OK` is the normal operating state.

`RS-LOF` (Regenerator Section, Loss of Framing) indicates that the STM-1/4 frame structure can not be recovered. One common reason is that the incoming signal is too weak, perhaps because the fiber isn't plugged in. Another common reason is that the incoming signal is not SDH—for instance it could be Ethernet.

`MS-AIS` (Multiplex Section, Alarm Indication Signal) indicates that the *remote* end can not recover the STM-1/4 frame structure. A typical reason is that the live link is directly connected to the Corelatus equipment instead of through a tap.

`MS-RDI` (Multiplex Section, Remote Defect Indication) indicates that the remote side is receiving MS-AIS.

Each SDH resource broadcasts an event when its state changes:

```
⇐ <event>
      <sdh_message name="sdh1" state="OK"|"RS-LOF"|"MS-AIS"|"MS-RDI"/>
  </event>
```

See also

The `enable` command (Section 3.5) turns on SDH interfaces. Running `enable` on an interface which is already enabled is equivalent to running `disable` then `enable`.

`disable` turns off SDH interfaces. All mappings and all jobs associated with the interface are cleared.

`query` reads status and counters.

4.5.2 HOP resources

An SDH link can carry many independent streams of signalling and data. These streams are organised into a multiplexing structure. The HOP resources represent the counters at the AU-3/AU-4 level. Each of these resources is named `sdhE:hopF_G`, where E, F and G are integers and the colon (:) separates path components.

HOP naming example: On an STM-1 link configured for AU=4, the resource `sdh1:hop1_1` represents the first (and only) AU-4 in the first (and only) AUG-1 in the link connected to the first SFP slot.

HOP naming example: On an STM-1 link configured for AU=3, the resource `sdh2:hop1_3` represents the last AU-3 in the first (and only) AUG-1 in the link connected to the second second SFP slot.

Read-only attributes

Attribute	Possible values
status	`OK` | `AU-AIS` | `AU-LOP` | `HP-AIS` | `TU-LOM` | `HP-PLM` | `HP-RDI` | `HP-UNEQ`
signal_label	The HOP signal label (payload type)
trace_identifier	A text string
_entered	The number of times the state was entered. There are four of these counters, one for each state.
_duration	The number of milliseconds the link has been in the state, in total. There are four of these counters.
HP-BIP	The Higher Order Path, Bit Interleaved Parity counter.
HP-REI	The Higher Order Path, Remote Error Indication counter.
child	A text string

The *status* attribute can be in one of these states:

`OK` is the normal operating state.

`AU-AIS` (Administrative Unit, Alarm Indication Signal) indicates that the remote end is in AU-LOP. It is also generated as a side effect of higher layer MS-AIS.

`AU-LOP` (Administrative Unit, Loss Of Pointer) indicates that the VC-3/4 payload pointer can not be recovered. Assuming the higher layer is OK, a typical cause is configuring for AU-3 when the link is actually carrying AU-4, or vice versa.

`HP-AIS` (Higher-order Path, Alarm Indication Signal). Generated by the remote end as a replacement signal if no incoming signal is available. It is also generated as a side effect of higher layer MS-AIS or AU-AIS.

`TU-LOM` (Tributary Unit, Loss Of Multiframe) Indicates that the TUG multiframe can not be recovered. Typical reason is that the VC-3/4 payload is something other than a TUG structure (E1/T1/J1).

`HP-PLM` (Higher-order Path, Payload Label Mismatch) indicates that the higher order payload is different than what is configured. Typical reason is a misconfiguration or a payload that is not (yet) supported by Corelatus.

`HP-RDI` (Higher-order Path, Remote Defect Indication) indicates that the remote end has detected a problem in the higher order path. The problem the remote end detected could be any of AIS, LOP, TIM (Trace Identifier Mismatch), PLM (Payload Mismatch) and UNEQ.

`HP-UNEQ` (Higher-order Path, Unequipped) indicates that the VC-3/4 does not contain any payload.

Section 3.31 shows an example of reading out a HOP resource's state and counters.

Each HOP resource broadcasts an event when its state changes:

```
⇐ <event>
      <sdh_message
          name="sdh1:hop1_1"
```

```
            state="OK"|"AU-AIS"|"AU-LOP"|"HP-AIS"
                |"TU-LOM"|"HP-PLM"|"HP-RDI"|"HP-UNEQ"/>
   </event>
```

4.5.3 LOP resources

On SDH links used for carrying E1/T1/J1 lines, the GTH exposes another level of resources representing the VC-11/VC-12.

When AU=3, these resources are named `sdhX:hopB_A:lopL_M`, where X, B, A, L and M are integers.

When AU=4, these resources are named `sdhX:hopB_A:lopK_L_M`.

Example: assuming an STM-1 with AU=3, the resource `sdh1:hop1_1:lop1_1` represents the first *VC-12* inside the first *TUG-2* inside the first and only *AU-3* in the first and only STM-1.

Example: assuming an STM-1 with AU=4, the resource `sdh1:hop1_1:lop3_7_3` represents the last *VC-12* inside the last *TUG-2* inside the last *TUG-3*.

Read-only attributes

Attribute	Possible values
status	`OK` | `TU-AIS` | `TU-LOP` | `LP-AIS` | `LP-PLM` | `LP-RDI` | `LP-UNEQ`
signal_label	The path signal label (payload type)
trace_identifier	A text string
_entered	The number of times the state was entered. There are four of these counters, one for each state.
_duration	The number of milliseconds the link has been in the state, in total. There are four of these counters.
LP-BIP	The Lower Order Path, Bit Interleaved Parity counter.
LP-REI	The Lower Order Path, Remote Error Indication counter.

The *status* values correspond to the similarly named values in the HOP resource.

The *trace_identifier* is a free-text string which describes the data being carried. In well-maintained networks, this often tells you exactly where the E1/T1/J1 comes from, e.g. it could be "HLR 3 London W".

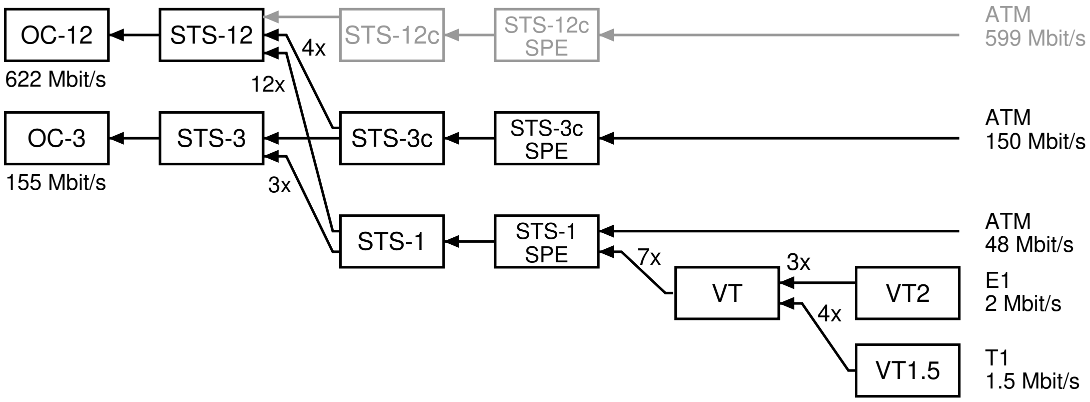
Section 3.31 shows an example of reading out a LOP resource's state and counters.

Whenever the LOP resource changes states, it broadcasts an event:

```
⇐ <event>
        <sdh_message
            name="sdh1:hop1_1:lop1_1"
            state="OK"|"TU-AIS"|"TU-LOP"
                |"LP-AIS"|"LP-PLM"|"LP-RDI"|"LP-UNEQ"/>
   </event>
```

4.6 SONET Links

SONET is used on 155 Mbit/s links, usually optical. GTH modules with SFPs support SONET, which in turn can carry either a few ATM links or many E1/T1/J1 links. GTH modules with electrical E1/T1/J1 interfaces do not support SONET.



The SONET multiplexing structure

Each SONET link has resources at three of the levels in the multiplexing structure shown in the diagram above. The top of the hierarchy is called `sdh1` and `sdh2` (see 4.6.1). The next level with GTH resources is at STS-1 (see 4.6.2). The lowest level with GTH resources is at VT2/VT1.5 SPE (see 4.6.3).

GTH supports both ATM and E1/T1/J1 on SONET. E1 links are carried in *VT2s*, shown on the right side of the diagram above. ATM is carried in either STS-3c (150 Mbit/s) or STS-1 (48 Mbit/s). T1 links are carried in *VT1.5s*. The `map` command makes an E1/T1/J1 carried in one of those VTs accessible to other commands.

4.6.1 sdh1 and sdh2

`sdh1` represents the SONET structure associated with `sfp1`.

`sdh2` and `sfp2` are similarly associated.

Configurable attributes

Attribute	Possible values
OC	`3`
VT	`1.5` | `2`
SONET	`true`
payload	`TUG` | `ATM`
daisy_chain	`false` | `true`

OC sets the OC level. OC-3 means the optical link operates at 155Mbit/s.

VT selects which path through the SONET multiplexing structure is used.

SONET selects which names are used for attributes, states, counters and events. By default, the names used in the SDH standard are used. If SONET is `true`, names corresponding to the SONET standard are used instead.

payload selects the type of payload. The default, `TUG`, indicates that there is a *lower order path* present, carrying E1 or T1 lines. `ATM` is used to carry ATM. All SONET interfaces must be configured for the same payload.

daisy_chain controls whether or not the incoming data on this SDH resource is re-transmitted, false by default. Daisy-chaining occurs cross-wise, i.e. enabling *daisy_chain* on sdh1 means that the data received by sdh1 is re-transmitted on sdh2.

Read-only attributes

Attribute	Possible values
status	`OK` | `LOF` | `AIS-L` | `RDI-L`
trace_identifier	A text string
_entered	The number of times the state was entered. There are four of these counters, one for each state.
_duration	The number of milliseconds the link has been in the state, in total. There are four of these counters.
B1	The B1 counter (Regenerator Section, Error Monitoring).
B2	The B2 counter (Line, Error Monitoring).
REI-L	The REI-L counter (Line, Remote Error Indication).
child	A text string

Normally, the *status* is `OK`. The remaining values indicate various errors.

`LOF` (Loss of Framing) indicates that the STM-1/4 frame structure can not be recovered. One common reason is that the incoming signal is too weak, perhaps because the fiber isn't plugged in. Another common reason is that the incoming signal is not SONET—for instance it could be Ethernet.

`AIS-L` (Line, Alarm Indication Signal) indicates that the *remote* end can not recover the STM-1/4 frame structure. A typical reason is that the live link is directly connected to the Corelatus equipment instead of through a tap.

`RDI-L` (Line, Remote Defect Indication) means that there is an incoming signal, but that the remote system has detected a problem.

Each SONET resource broadcasts an event when its state changes:

```
⇐ <event>
      <sdh_message name="sdh1" state="OK"|"LOF"|"AIS-L"|"RDI-L"/>
  </event>
```

See also

The `enable` command (Section 3.5) turns on SDH interfaces. Running `enable` on an interface which is already enabled is equivalent to running `disable` then `enable`.

`disable` turns off SDH interfaces. All mappings and all jobs associated with the interface are cleared.

`query` reads status and counters.

4.6.2 HOP resources

A SONET link can carry many independent streams of signalling and data. These streams are organised into a multiplexing structure. The HOP resources represent the counters at the STS-1 level. Each of this resources is named named `sdhE:hopF`, where E and F are integers and the colon (:) separates path components.

HOP naming example: On an OC-3 link there are three possible STS-1s to choose between. The resource `sdh1:hop1` represents the first STS-1 in the link connected to the first SFP slot on the GTH module.

Read-only attributes

Attribute	Possible values
status	`OK` | `AIS-P` | `LOP-P` | `LOM` | `PLM-P` | `RDI-P` | `UNEQ-P`
signal_label	The path signal label (payload type)
trace_identifier	A text string
_entered	The number of times the state was entered. There are seven of these counters, one for each state.
_duration	The number of milliseconds the link has been in the state, in total. There are seven of these counters.
B3	STS-Path, Error Monitoring counter.
REI-P	STS-Path, Remote Error Indication counter.
child	A text string

The *status* attribute can be in one of these states:

`OK` is the normal operating state.

`AIS-P` (Alarm indication signal, path) indicates that the remote end is in AU-LOP. It is also generated as a side effect of higher layer AIS.

`LOP-P` (Loss of pointer, path) indicates that the STS-3 payload pointer can not be recovered.

`LOM` (Loss of multiframe) indicates that the VT multiframe can not be recovered. Typical reason is that the fiber is carrying something other than E1/T1/J1 lines.

`PLM-P` (Payload label mismatch, path) indicates that the higher order payload is different than what is configured. Typical reason is a misconfiguration or a payload that is not (yet) supported by Corelatus.

`RDI-P` (Remote defect indication, path) indicates that the remote end has detected a problem in the higher order path. The problem the remote end dected could be any of AIS, LOP, TIM (Trace Identifier Mismatch), PLM (Payload Mismatch) and UNEQ.

`UNEQ-P` (Unequipped, path) indicates that the VC-3/4 does not contain any payload.

Section 3.31 shows an example of reading out a HOP resource's state and counters.

Each HOP resource broadcasts an event when its state changes:

```
⇐ <event>
       <sdh_message
           name="sdh1:hop1"
           state="OK"|"AIS-P"|"LOP-P"|"AIS-P"
               |"LOM"|"PLM-P"|"RDI-P"|"UNEQ-P"/>
   </event>
```

4.6.3 LOP resources

On SONET links used for carrying E1/T1/J1 lines, the GTH exposes another level of resources representing the VT1.5/VT2. These resources are named `sdhJ:hopK:lopL_X`, where J, K, L and X are integers.

LOP naming example: assuming an OC-3, the resource `sdh1:hop1:lop1_1` represents the first *VT1.5* inside the first *VT-group* inside the first *STS-1*.

LOP naming example: assuming an OC-3, the resource `sdh1:hop3:lop7_4` represents the last *VT1.5* inside the last *VT-group* inside the last *STS-1*.

Read-only attributes

Attribute	Possible values
status	`OK` | `AIS-V` | `LOP-V` | `PLM-V` | `RDI-V` | `UNEQ-V`
signal_label	The path signal label (payload type)
trace_identifier	A text string
LP-BIP	The Lower Order Path, Bit Interleaved Parity counter.
LP-REI	The Lower Order Path, Remote Error Indication counter.

The *status* values correspond to the similarly named values in the HOP resource.

The *trace_identifier* is a free-text string which describes the data being carried. In well-maintained networks, this often tells you exactly where the E1/T1/J1 comes from, e.g. it could be "HLR 3 London W".

Section 3.31 shows an example of reading out a LOP resource's state and counters.

Whenever the LOP resource changes states, it broadcasts an event:

```
⇐ <event>
      <sdh_message
          name="sdh1:hop1:lop1_1"
          state="OK"|"AIS-V"|"LOP-V"|"PLM-V"|"RDI-V"|"UNEQ-V"/>
  </event>
```

4.7 Sync sources

The `sync` resource lets you control and monitor how the GTH synchronises itself to the telecom network frequency and to the absolute time.

Absolute time sync

GTH can adjust the absolute (wall) time using the Network Time Protocol (NTP) or by explicitly setting the time attribute. NTP is the preferred method, if an NTP server is configured, the GTH will automatically adjust its time approximately once every 15 minutes.

The attributes related to the absolute time are:

Attribute	Possible values in query	Allowed values in set
primary ntp	an IP4 address (a dotted-quad)	as for query
secondary ntp	"	"
ntp status	the last clock adjustment	read-only
time	the current time (UTC)	(see below)

The NTP status indicates the last time NTP was successfully used to adjust the clock, followed by the clock error in seconds.

Time is indicated (and set) using a format which places the components of the time in order of magnitude. For example, 11:34am on the 8. January 2002 is represented by the string `2002.1.8-11:34:00`.

SDH/SONET network sync

On hardware with SDH/SONET interfaces, the telecom network sync is always automatic. Sync is taken from the first interface which is enabled and has status `OK` or `MS-RDI`.

Attribute	Possible values in query	Allowed values in set
mode	`auto`	read-only
source	`sdh1|sdh2|none`	read-only
status	`locked`	read-only

The special *source* `none` indicates that the internal frequency source is used because no better sync source is available.

E1/T1/J1 network sync

On hardware with E1/T1/J1 interfaces, the *source* controls which E1/T1/J1 interface the GTH uses as a frequency reference. The special value `none` causes the GTH's internal, temperature-compensated frequency source to be used. The special value `auto` allows the GTH to decide which of the E1/T1/J1 interfaces to use as a reference.

mode indicates whether the sync source was chosen by the GTH (`auto`) or set by the user (`manual`).

Attribute	Possible values in query	Allowed values in set
mode	`auto|manual`	read-only
source	`none|pcm1A|pcm2A, ....`	as for query, plus `auto`
status	`trying|locked|dead`	read-only

The sync state `trying` indicates that the GTH is attempting to sync to the given E1/T1/J1 port, when it successfully syncs, the state changes to `locked`. If the port does not contain a usable sync, the state changes to `dead`.

Events

If an NTP server fails to respond to NTP requests, the GTH sends an event:

```
⇐ <event>
    <alert reason="ntp">(human readable text)</alert>
  </event>
```

If the E1/T1/J1 sync system changes states, the GTH sends an event showing the new state:

```
⇐ <event>
      <sync_message state="trying"|"locked on"|"dead"|"limit reached"/>
  </event>
```

The pseudo-state `limit_reached` is reported when the sync system reaches the limit of its frequency adjustment range.

4.8 Ethernet Interfaces

The GTH's two Ethernet interfaces are named `eth1` and `eth2`. They are used to control the GTH.

Configurable attributes

Attribute	Possible values
IP4 address	An IP address, e.g. 172.16.1.10
IP4 mask	An IP mask, e.g. 255.255.0.0
default gateway	An IP address, e.g. 172.16.1.10
load limit	An integer.

The *IP4 address, mask* and *gateway* settings persist across reboots. In most cases, the IP4 address and mask should be set in the same operation:

The *default gateway* setting makes the GTH module accessible in a routed network. Use this with caution, as per section *routed networks* below.

The special value `none` disables an interface or clears the default gateway.

The *load limit* configures the alarm threshold, in percent of maximum interface capacity. The default value is 75.

```
⇒ <set name="eth1">
     <attribute name="IP4 address" value="128.250.22.3"/>
     <attribute name="IP4 mask" value="255.255.255.0"/>
   </set>
⇐ <ok/>
```

Read-only attributes

<table>
<tr><th>Attribute</th><th>Possible values</th></tr>
<tr><td>MAC address</td><td>A text string</td></tr>
<tr><td>collisions</td><td>The number of ethernet packet collisions</td></tr>
<tr><td>TX errors</td><td>The number of transmit errors</td></tr>
<tr><td>TX load</td><td>0–100</td></tr>
<tr><td>TX average load</td><td>0–100</td></tr>
<tr><td>TX maximum load</td><td>0–100</td></tr>
<tr><td>TX octets</td><td>The number of octets transmitted</td></tr>
<tr><td>link status</td><td><code>up</code> | <code>down</code> | <code>unknown</code></td></tr>
<tr><td>link speed</td><td><code>10</code> | <code>100</code> | <code>unknown</code></td></tr>
<tr><td>link duplex</td><td><code>FD</code> | <code>HD</code> | <code>unknown</code></td></tr>
</table>

The *TX load* and *TX load limit* show how many percent of the interface's capacity are being used. E.g.: 5% of a 100Mbit/s ethernet interface is 5Mbit/s (about 600kByte/s).

The *locked* attribute is an alias for an attribute of the same name in the in-built WWW server (see 4.15). The alias is kept for backwards compatibility.

Events

Whenever the *TX load* exceeds the load limit, GTH sends an event:

```
⇐ <event>
        <alarm
            name="eth1"
            attribute="TX load"
            state="set"|"clear"
            value=string/>
   </event>
```

Restrictions

For the IP configuration, the GTH API checks for some common mistakes and then allows anything else. The common mistakes are:

- Setting an IP or mask which isn't valid, e.g. 1.b.3.c is not an IP address.
- Disabling both ethernet ports (*painted into a corner*).
- Setting an IP which is a broadcast or loopback address.
- Setting an unreachable gateway.

The on-board HTTP interface also checks for:

- Both addresses on the same subnet.
- Changing an IP such that the gateway would be unreachable.

Routed Networks

The best location for a GTH is on the same subnet as the server controlling it, i.e. connected to the same switch, or connected by a direct ethernet cable.

It is also possible to use a GTH on a routed network by setting the *default gateway*. This is convenient for accessing the CLI or WWW interfaces remotely. It is even possible to place the server controlling the GTH remotely. The price is additional complexity:

Security The GTH's security is appropriate for a safe network, i.e. free of attackers. When setting a *default gateway*, you must be confident that the entire routed network is still safe.

TCP timeouts A routed network may contain firewalls or NAT devices which discard idle TCP connections. If the firewall terminates a TCP connection carrying the API, the GTH will respond by terminating all jobs—e.g. signalling decoding— started on that API connection. The GTH has a heartbeat feature (5.4) which can be used avoid this problem.

Performance Reasoning about TCP performance on a single subnet is straightforward, there is no ethernet packet loss and round-trip times are short. In contrast, a routed network may suffer from packet loss, congestion and have long round-trip times. You need to be sure that the network's characteristics are good enough for your reliability requirements.

4.9 CPU

The `cpu` resource represents the CPU inside the GTH.

Attributes

Attribute	Possible values
load limit	An integer.
load	the unix load average
total memory	in octets
free memory	in octets

The *load* is the unix load average, i.e. the mean number of runnable jobs in the run queue. It is possible for this to exceed 1.0.

The *load limit* sets the alarm threshold in percent. It defaults to 100.

Events

If the CPU load exceeds the limit, the GTH sends an event to all API sockets:

```
⇐ <event>
      <alarm
          name="cpu"
          attribute="load"
          state="set"|"clear"
          value=string/>
  </event>
```

4.10 Board

The `board` resource is a catch-all resource for all of the attributes which affect the whole module and don't fit elsewhere.

Configurable attributes

Attribute	Possible values
auto conferences	`enabled` | `warn` | `disabled`
LED mode	`normal` | `flashing`
PCM LED assignment	`universal` | `sequential`
voice coding	`alaw` | `mulaw`

The *auto conferences* setting tells the GTH what to do when multiple `connection` or `player` jobs send output to the same timeslot. `enabled` means that the GTH will sum the audio, so that the subscriber hears all sources. `warn` behaves the same as `enabled`, but also writes a warning to the `application_log`. `disabled` causes the GTH to reject any new job which would create such a conference.

The *LED mode* can be set to `flashing` to make all the LEDs flash on and off. This is useful for finding a particular module in a rack with many modules.

The *PCM LED assignment* changes the mapping between PCMs and LEDs in the connectors and on the front panel. The default setting, `universal`, is backwards compatible with all Corelatus hardware and agrees with the diagrams in Corelatus documentation.

The *voice coding* affects the tone detector, conferencing and players. It defaults to `alaw`. In general, public telephone systems in Europe use `alaw` while systems in the USA use `mulaw`.

Read-only attributes

Attribute	Possible values
power consumption	A number with a decimal point
power source	`A` | `B` | `both` | `none`
POE source	`eth1`, `eth2`
ROM ID	A text string
temperature	A number with a decimal point
architecture	`gth1` | `gth2.0` | `gth2.1` | `gth3.0` | `sth3.0`

power consumption indicates the module's current power consumption in Watts. This value typically fluctuates between readings, depending on what the GTH is doing at that moment.

power source indicates which of the DC power inputs currently have power. It can be one of `A, B` or `both`. If more than one input is powered up, the GTH uses the one with the higher voltage.

POE source indicates which of the power-over-ethernet inputs are active. This attribute is present on GTH 3.0 and STH 3.0.

ROM ID is an arbitrary value which is guaranteed to uniquely identify a particular GTH.

temperature indicates the PCB temperature at the part of the GTH with the greatest power dissipation. In a properly ventilated rack, the PCB temperature is typically 15–25^o above the air temperature.

The *architecture* reveals the hardware generation. In very old releases, i.e. before gth2_system_33a (December 2008), both gth2.0 and gth2.1 are reported as `gth2`.

Events

If the temperature goes outside the allowed range ($10-60^o$ Celsius, $10-70^o$ on STH Monitor 3.0), an event is broadcast to all API sockets:

```
⇐ <event>
      <alarm
          name="board"
          attribute="temperature"
          state="set"|"clear"
          value=string/>
  </event>
```

If the power is lost on one of the two 48VDC inputs or on an input with power-over-Ethernet:

```
⇐ <event>
      <alert
          reason="power A lost"|"power B lost"
              |"power eth1 lost"|"power eth2 lost"/>
  </event>
```

If power returns on a power input:

```
⇐ <event>
      <alert
          reason="power A recovered"|"power B recovered"
              |"power eth1 recovered"|"power eth2 recovered"/>
  </event>
```

4.11 OS

The GTH operating system is represented by `os`.

Configurable attributes

Attribute	Possible values
boot mode	`normal` | `failsafe`
remote login	`enabled` | `disabled`
API whitelist	A text string

The GTH includes two complete installations of the operating system and application firmware. The *boot mode* controls which of those two installations is booted. In the `normal` boot mode, the `system image` is used.

remote login enables or disables SSH on port 22

API whitelist is a list of IP addresses from which the GTH will accept connections on port 2089. IP addresses are represented as dotted quads and separated by spaces. An empty list (the default) means all IP addresses are allowed. Example:

```
⇒ <set name="os">
      <attribute name="API whitelist" value="128.250.22.3 172.16.2.1"/>
  </set>

⇐ <ok/>
```

Read-only attributes

Attribute	Possible values
uptime	the number of seconds since the last reboot
last restart	A text string
restart type	`hard` | `soft` | `watchdog`
restart cause	`kernel reboot` | `power failure` | `unknown`

last restart is the wall clock time (UTC!) of the most recent reboot.

restart type and *restart cause* indicate why the GTH restarted. This is useful for forensics, i.e. answering “why did the system restart?”.

4.12 System Image and Failsafe Image

The two firmware images on the GTH are called `system` and `failsafe`. During everyday operation, the `system` image is used, it supports the full set of GTH commands. In exceptional circumstances, for instance while upgrading, the `failsafe` image is used.

The `failsafe` image provides just enough functionality to support upgrading and troubleshooting. The only commands it understands are `bye, install, nop, query, reset` and `set`.

If the GTH fails to boot after five attempts, it switches boot images—e.g. to `failsafe`—and tries five more times.

Attributes

Attribute	Possible values
locked	`true` | `false`
version	A text string
busy	`true` | `false`

An empty image has the special version string `empty`, cannot be locked and cannot be busy. A *locked* image must be unlocked before it can be overwritten.

There are three circumstances under which the failsafe image may be booted:

1. The system image is damaged, for instance as a result of a failed upgrade.
2. The GTH has attempted more than 5 consecutive boots without successfully booting. "Successfully booting" is defined as completing the boot process, starting the API and staying up for two minutes.
3. The *boot mode* has been manually set to `failsafe` before the most recent boot.

4.13 Start Script

At boot time, the GTH runs a `start_script` consisting of a sequence of `custom` commands, separated by blank lines. By default, the start script is empty.

The `custom` command (3.2) describes the format and contents of the start script.

The `install` command (3.6) writes a start script to the firmware.

A `query` command on the `start_script` shows the contents of the start script.

4.14 Control System and Operating System Logs

The GTH divides event and fault information into two logs: `application_log` and `system_log`. These logs can be inspected using a query on the respective resource, which will cause the log (as ASCII text) to be returned in a *text/plain* block immediately following the query response.

The *verbosity* attribute controls the level of logging in the application log. Starting with the greatest amount of logging, the available levels are:

Log level	Information logged
`all`	All logging enabled. All XML commands are logged. All XML responses are logged. All XML events are logged. Internal system status messages are logged. In applications which send many XML commands per second, logging all of the above will significantly reduce GTH performance and wrap logs quickly.
`changes`	XML commands apart from `nop` and `query` are logged. Internal system status messages are logged. This setting is the default.
`info`	As for `changes`, except all successful commands are suppressed. The `info` setting provides the best performance, at the expense of leaving less debugging information.

The resources `standby_application_log` and `standby_system_log` provide access to the logs from the firmware image which is *not* running.

4.15 HTTP Server

The on-board HTTP server.

Configurable attributes

Attribute	Possible values
enabled	`true` | `false`
locked	`false` | `true`
passwords	A text string
pcm_L1_zero_buttons	`false` | `true`
pcm_L2_zero_buttons	`false` | `true`

The *locked* attribute is used to control configuration changes from the **webserver**. If *locked* is true, the web server cannot change the IP address and mask, nor can it upgrade the firmware image. The API is not affected by the *locked* attribute. The *locked* attribute persists across system restarts.

The *passwords* attribute is a comma delimited sequence of `username:password` pairs, for example:

```
user1:password1,user2:password2,user3:password3
```

The webserver uses HTTP 1.1 digest authentication (RFC 2617) to avoid exposing passwords to ethernet sniffers. Passwords *are* vulnerable to sniffing *when they are loaded into the system* via a `set` or `custom` command.

pcm_L1_zero_buttons and *pcm_L1_zero_buttons* can be set to enable buttons in the WWW interface which zero out counters on the L1 and L2 interfaces. Enabling this feature is **not recommended** because it leads to situations where a supervising application cannot distinguish between a counter wrapping and a counter being manually zeroed.

URL rewriting

All of the pages on the HTTP server can be overridden with user-defined pages by setting an attribute to a URL:

Attribute	Purpose
top	The page reached at http://gth:8888/
ethernet	The ethernet configuration page
l2_status	Layer 2 (MTP2 and LAPD) status page
os	The operating system and logs page
pcm_overview	The PCM interface overview page
status	The top-level system status page. By default, this is the same as *top*

Example: to set the top-level page such that the client browser visits 'google':

```
⇒ <set name="http_server">
      <attribute name="top" value="http://www.google.com"/>
   </set>
⇐ <ok/>
```

4.16 Performance

Different generations of hardware and firmware can decode different numbers of layer 2 channels. This resource shows the maximum decoding capacity for each protocol.

Systems can also be field-upgraded from *Basic* performance to *Pro*, using the *identity*, *challenge* and *response* attributes:

Attributes

Attribute	Possible values
response	A text string
atm_channels	The ATM decoding capacity (channels).
atm_bandwidth	The ATM decoding capacity (kbit/s).
frame_relay_channels	The frame relay decoding capacity.
frame_relay_bandwidth	An integer.
challenge	A text string
identity	A text string
lapd_channels	The LAPD decoding capacity.
lapd_bandwidth	An integer.
mtp2_channels	The SS7 MTP-2 decoding capacity.
mtp2_bandwidth	An integer.
raw_channels	The raw timeslot decoding capacity.

Example: an STH 3.0 submodule with a 'Pro' licence is capable of decoding 240 channels of MTP-2, so *mtp2_channels* will show 240.

5 Fault Tolerant Systems

GTH provides support for building systems which tolerate hardware and software failure. Which features you use depends on which failures you want to be able to handle and on how much effort you want to expend on fault tolerance. Each of the following sections covers an aspect of fault tolerance, the sections are ordered approximately in order of difficulty.

5.1 Startup Checks

Automatically checking that the GTH's state using `query` commands at startup time is a simple way to catch misconfigurations. A typical set of attributes to check would include all of the following:

Resource	Attribute	Why
system_image	*version*	Avoid wasting time debugging old bugs.
	busy	`false` is bad, it means the GTH is in failsafe mode. Proceeding is pointless, manual investigation is in order.
os	*restart type*	If it is `watchdog`, the card crashed. Send a bug report to Corelatus.
	restart cause	If it is `power failure`, investigate the cause.
schedule		The schedule is a list of jobs currently running on the GTH. In most systems, this will contain just one job at startup: the controller itself.
sync	*ntp status*	Without time synchronisation, the log timestamps will be wrong, which makes debugging harder.

When restarting after an error, reading out and saving the GTH's logs (the application_log and system_log) for later analysis is good practice.

5.2 Runtime Checks

The GTH performs many continuous runtime checks on resources and jobs and reports potentially interesting status changes via asynchronous `event` messages. The important `alarm` and `alert` events are:

Type	When/Where	Notes
alarm	board temperature	Running hardware outside its rated temperature range shortens hardware lifetime and can result in temporary or permanent failure.
alarm	cpu load	An unexpectedly high CPU load can be a symptom of an underlying problem.
alarm	eth1 load	Unexpectedly high ethernet load can be a symptom of an underlying problem.
alarm	eth2 load	
alert	ntp	Losing contact with an NTP server will cause the GTH's time to drift. In signalling monitoring applications, this will complicate correlating timestamps from different GTHs.
alert	power inputs	In high reliability installations, losing one power input indicates that either the site power supply has failed or the battery backup has failed.

A simple general policy for dealing with events is to automatically deal with expected events and report all other events to an operator for manual attention.

5.3 E1/T1/J1 Layer 1 Runtime Checks

In all GTH applications, a correctly functioning E1/T1/J1 (PDH) network is crucial. Once a system is running, all E1/T1/J1 resources should stay in the `OK` state. If they leave the OK state, GTH sends an `l1_message` to warn that calls and signalling are likely to be degraded or fail completely.

A second symptom of E1/T1/J1 network problems is difficulty maintaining synchronisation. The GTH reports changes in E1/T1/J1 synchronisation status via the `sync_message`. Per-E1/T1/J1 `slip` messages indicate lost data on that E1/T1/J1, which will degrade speech and signalling.

For each E1/T1/J1 interface, these counters should be sampled at regular intervals—e.g. once every 15 minutes—and reported if they go outside the normal range.

Counter	Normal range	Comments
slip_positive, slip_negative	0	A slip *always* causes data loss or corruption. It is possible to build national PDH networks with sync so stable that they never have a slip. Typical carrier-class networks run for months without a slip.
frame_error	0–1	
code_violation_seconds	0–5	A code violation almost always causes a bit error. A code violation indicates that the incoming PCM signal followed a pattern which is not allowed by the (configurable) line coding. The most common cause is line noise, especially when using -20dB monitor points.
crc_error	0–5	In `doubleframe` mode, the CRC is not present so this counter is always zero. In `multiframe` mode, it indicates errors.
state counters	(none)	PCM L1 can be in one of a number of states: LOS, LFA, LMFA, RAI, AIS and OK. The GTH counts how long each PCM was in each state. There should never be an L1 state change without an explanation. Typical explanations are that PCM cabling was unplugged or a remote system was restarted.

Several of the events these counters detect have causal relations, for instance in a multiframe system, every code violation is expected to cause a CRC error.

5.4 Heartbeat Supervision

A controller can supervise a GTH, as well as the network between the GTH and the controller, by periodically sending a `nop` command. If the GTH does not reply with `ok` within one second, the controller can assume there is a fault.

A controller can also request supervision by the GTH. The GTH implicitly creates a job with the ID 'apicXXX' whenever a port 2089 API connection is started. Configure this implicit job using `update`. Look up its ID by querying the special job 'self'.

One of the controller job's attributes is the *timeout*, which controls supervision. If the *timeout*, in milliseconds, is nonzero, then the GTH starts a timer after each command is received. If the timer expires before the next XML command arrives, the socket is closed, an error logged and all jobs started on (*owned by*) that socket deleted, i.e. the GTH falls back to a known state:

```
⇒ <update>
      <controller timeout="10000"/>
   </update>
⇐ <ok/>
```

```
...(8s elapses)...

⇒ <nop/>
⇐ <ok/>

...(10s elapses)...

⇐ <error reason="timeout"/>
```

5.5 Duplicating Ethernet

GTH hardware has two ethernet interfaces, so it is possible to build systems with completely duplicated IP networks: two switches, two ethernet interfaces on the controller and two ethernet interfaces on the GTH.

The only restriction the GTH imposes is that its ethernet interfaces must be on separate subnets (this is a general restriction in Linux and other operating systems). Some ways to use duplicated IP networks are:

- Duplicated network with cold failover. The controller normally uses one of the two networks. If that network fails, typically detected with heartbeat supervision, the controller restarts the system using the second network, dropping all calls.
- Duplicated network with hot failover. This is like cold failover, except that the controller uses `transfer` to move jobs to the new port 2089 socket. With this method, it is possible to recover from an IP network failure without dropping calls.
- Duplicated network and replicated controllers. This is discussed in the next section.

5.6 Controller Replication and Failover

The GTH allows multiple concurrent API socket connections. This can be exploited to provide fault containment within an application, but it also allows a system with several cooperating controllers to recover from a controller failure without dropping calls.

Imagine a system consisting of one GTH and two servers. Each server opens an API socket to the GTH, each with its own GTH-supplied job ID, for this example `apic1` and `apic2`. The first server is the live one, it enables E1/T1/J1 resources, issues `new connection` and `new player` commands and processes DTMF tones. The second controller is in standby, i.e. it does nothing. Both servers configured a 10s *timeout* and send `nop` commands at 5s intervals as a heartbeat.

What happens if the power supply for the first server spontaneously combusts? In less than 10s, the GTH will timeout on `apic1`, log an error and `delete` all the jobs started on `apic1`, i.e. all the players will stop and the connections will be broken. In this case, the `transfer` command could not have saved the situation because the second server cannot always carry out the transfer before the GTH automatically deletes the jobs.

The solution is to configure the GTH to `transfer` the jobs to `apic2` instead of deleting them, by setting the *backups* list:

```
⇒ <update>
     <controller timeout="20000" backups="apic2"/>
  </update>
⇐ <ok/>
```

In this configuration, when `apic1` **terminates abnormally**, i.e. without a `bye` command, the GTH will `transfer` all jobs owned by `apic1` to `apic2`, i.e. to the second server. All events associated with those jobs will now go to `apic2`. The GTH will also send an event to `apic2` to inform it of the automatic transfer:

```
⇐ <event><backup>
     <job id="play19413"/>
     <job id="cnxn16448"/>
  </backup></event>
```

6 XML Tools

There are several tools to help with validating XML. If the GTH is rejecting an apparently valid command, try passing it through a validator together with the provided DTD.

6.1 Internet Explorer and Mozilla/Firefox

These browsers can be used to check whether a document is well-formed or not. Loading this file:

```
<?xml version = "1.0"?>
<!DOCTYPE gth_in SYSTEM "https://www.corelatus.com/gth/api/gth_in.dtd">

<gth_in>
  <new>
     <player>
        <clip id="clip please try again"/>
        <pcm_sink span="3" timeslot"12"/>
      </player>
  </new>
</gth_in>
```

produces the following error message in the browser:

```
XML Parsing Error: not well-formed
Location: file:///home/matthias/bad.xml
Line Number 8, Column 36:
        <pcm_sink span="3" timeslot"12"/>
-----------------------------------^
```

Browsers only check for well-formedness. They verify whether or not a file is valid XML, but they do not check that the file follows the GTH DTD.

6.2 W3 Online Validator

The W3 consortium has an online validator at http://validator.w3.org/. Validating the above example produces this error report:

```
# Line 8, column 35:  an attribute value literal can
occur in an attribute specification list only after a
VI delimiter

          <pcm_sink span="3" timeslot"12"/>
                                    ^
```

The W3 validator checks using the GTH DTD, so it will find more errors than a browser.

6.3 xmllint

xmllint is a freely-available command-line driven XML parser. It produces error output similar to the W3 validator and it is straightforward to use in automated test systems:

```
>xmllint --valid  play.xml
play.xml:8: parser error : Specification mandate value
                           for attribute timeslot
<pcm_sink span="3" timeslot"12"/>
                           ^
```

xmllint is free. Download it from http://www.xmlsoft.org/. Many Linux distributions include it, e.g. in Debian-based distributions it's in libxml2-utils. For Windows and Mac users, there's a graphical application which does the same thing, tkxmllint, available from http://tclxml.sourceforge.net/tkxmllint.html.

7 Changelog

The changelog shows changes made in the past few years.

since March 2023

- Official support for V.110 decoding.
- Corrections and clarifications.

since January 2019

- Support for J1 interfaces (Japan).
- Support for 48 kbit/s subrate MTP-2 (Japan).
- Support for 8, 16 and 32 kbit/s `<raw_monitor>`.
- Support for V.110 decoding in `<v110_monitor>`.
- Expand the explanation of 'cross-wise daisy-chaining'.

since May 2015

- Support for ATM AAL5 in VC-4 (150 Mbit/s) and VC-3 (48 Mbit/s) on SDH is documented. Using it requires a firmware update to `sth3_system_42a`.
- Broadcast events, e.g. E1/T1/J1 and SDH/SONET status changes, can now be suppressed on a per-API-connection basis. The `<update>` command has a new parameter, `broadcast_events`, which controls this. This feature is present in `sth3_system_42a` and `gth3_system_42a`.

8 Acronyms and Initialisms

AAL0 ATM Adaptation Layer Zero. A stream of raw ATM cells.

AAL5 ATM Adaptation Layer Five. Defined in ITU-T I.363.5.

AIS Alarm Indication Signal. Used in both E1/T1/J1 and SDH/SONET. See 4.2 and 4.5.

API Application Programming Interface.

ATM Asynchronous Transfer Mode. A family of signalling protocols defined in a series of ITU standards including ITU-T I.361.

AU SDH/SONET Administrative Unit.

B8ZS Bipolar Eight Zero Substitution. See section 4.2.

CPCS ATM Common Part Convergence Sublayer. An ATM AAL5 sublayer.

CRC Cyclic Redundancy Check. A type of checksum.

DTD An XML Document Type Definition.

DTMF Dual Tone Multi-Frequency (touch-tone in-band signalling).

E1 A 2 Mbit/s link as described in ITU-T G.703.

ESU Errored-signal-unit. Part of MTP-2, LAPD and frame relay signalling.

FISU MTP-2 Fill-in-signal-unit.

GbE Gigabit Ethernet.

GTH Generic Telecom Hardware. A family of systems made by Corelatus to connect to the telephone network.

HDB3 High-Density-Bipolar-3. See section 4.2.

HOP SDH/SONET Higher Order Path.

HP-AIS SDH/SONET state Higher order Path, Alarm Indication Signal.

HP-PLM SDH/SONET state Higher order Path, Payload Label Mismatch.

HP-RDI SDH/SONET state Higher order Path, Remote Defect Indication.

HP-REI SDH/SONET state Higher order Path, Remote Error Indication.

HP-UNEQ SDH/SONET state Higher order Path, Unequipped.

HTTP Hyper-Text Transfer Protocol. Defined in RFC 2068.

IVR Interactive Voice Response. The automated systems you ring up and communicate with by pressing buttons on your phone.

J1 A 1.544 Mbit/s link used in Japan, specified in JT-G704.

L1 Layer 1, e.g. an E1/T1/J1 line.

L2 Layer 2, e.g. MTP-2 or IP or ATM.

LAPD The protocol described in ITU-T Q.920 and Q.921.

LFA Loss of frame alignment. See section 4.2.

LOP SDH/SONET Lower Order Path.

LP-PLM SDH/SONET state Lower order Path Payload Label Mismatch.

LP-RDI SDH/SONET state Lower order Path Remote Defect Indication.

LP-UNEQ SDH/SONET state Lower order Path Unequipped.

LMFA Loss of multi-frame alignment. See section 4.2.

LOF SDH/SONET Loss of Framing.

LOS Loss of signal. See section 4.2.

LSSU Link-status-signal-unit. Part of MTP-2.

MS-AIS SDH Multiplex Section, Alarm Indication Signal.

MS-BIP SDH Multiplex Section, Bit Interleaved Parity.

MS-RDI SDH Multiplex Section, Remote Defect Indication.

MSU MTP-2 Message Signal Unit.

MTP-2 Message Transfer Protocol, layer 2. Described in ITU-T Q.703 and ANSI T1.111.1.

NTP Network Time Protocol. A method for synchronising clocks over IP networks, defined in RFC 1305.

OC-3 Optical carrier, rate 3. A 155 Mbit/s optical fiber.

OC-12 Optical carrier, rate 12. A 622 Mbit/s optical fiber.

PLM-P SDH/SONET Payload Mismatch, Path.

PLM-V SDH/SONET Payload Mismatch, Virtual Tributary.

PoE Power over Ethernet.

RAI Remote Alarm Indication. See section 4.2.

RDI SDH/SONET Remote Defect Indication.

RS-BIP SDH Regenerator Section, Bit Interleaved Parity.

RS-LOF SDH Regenerator Section, Loss Of Framing

SDH Synchronous Digital Hierarchy.

SDU ATM Service Data Unit.

SFP Small Form-factor Pluggable. A plug-in transceiver module used by SDH/SONET capable hardware to connect to optical fiber links.

SONET Synchronous Optical Networking.

SPE SONET Synchronous Payload Envelope.

SS7 Signalling System 7. The signalling protocol used in telephone networks.

SSH Secure SHell. A remote login protocol.

STM-1 Synchronous Transport Module, level-1. An optical transmission standard which runs at 155 Mbit/s and can carry 63 E1 links or 84 T1/J1 links.

STM-4 Synchronous Transport Module, level-4. An optical transmission standard which runs at 622 Mbit/s.

STS- SONET Synchronous Transport Signal.

T1 A 1.544 Mbit/s link as described in ITU-T G.703.

TCP Transmission Control Protocol. Defined in RFC793.

TU SDH Tributary Unit.

TUG SDH Tributary Unit Group.

UNEQ SDH/SONET Unequipped.

UTC Coordinated Universal Time. A method of defining absolute time. It is almost identical to GMT.

VC SDH/SONET Virtual Container.

VCI ATM Virtual Channel Identifier. Part of the cell address.

VPI ATM Virtual Path Identifier. Part of the cell address.

VT SONET Virtual Tributary.

XML Extensible Markup Language, http://www.w3.org/TR/REC-xml