

GTH API (December 2008)

Part number 10-0003

The latest version is available at <http://www.corelatus.com/gth/api/>**Contents**

1	Introduction	2
1.1	Three Typical Applications	2
2	Principles and Terms	2
2.1	Sending Commands to the GTH	3
2.2	Commands and Responses	4
2.3	Events	5
2.4	Voice	6
2.5	Signalling	7
2.6	Handling Errors	8
3	Command Reference	10
3.1	bye	11
3.2	custom	12
3.3	delete	14
3.4	install	15
3.5	new atm_aal0_monitor	16
3.6	new atm_aal2_monitor	20
3.7	new atm_aal5_monitor	23
3.8	new cas_r2_linesig_monitor	27
3.9	new cas_r2_mfc_detector	29
3.10	new clip	31
3.11	new connection	32
3.12	new fr_monitor	34
3.13	new lapd_layer	37
3.14	new lapd_monitor	39

3.15 new level_detector	42
3.16 new mtp2_monitor	43
3.17 new player	47
3.18 new recorder	50
3.19 new ss5.linesig_monitor	51
3.20 new ss5.registersig_monitor	53
3.21 new tone_detector	55
3.22 nop	57
3.23 query	58
3.24 reset	62
3.25 set	63
3.26 takeover	64
3.27 update	65
3.28 zero	67
4 Resources	68
4.1 Inventory and Schedule	68
4.2 E1/T1 links	68
4.3 Sync Sources	70
4.4 Ethernet Interfaces	71
4.5 CPU	72
4.6 Board	73
4.7 OS	74
4.8 System Image and Failsafe Image	74
4.9 Control System and Operating System Logs	75
4.10 HTTP Server	75
5 Fault Tolerant Systems	76
5.1 Startup Checks	77
5.2 Runtime Checks	77
5.3 Heartbeat Supervision	78
5.4 Duplicating Ethernet	79
5.5 Controller Replication and Failover	79

6	Worked Example: Upgrading	81
6.1	The System and Failsafe Images	81
6.2	Upgrading	81
6.3	Upgrading the Failsafe System	82
7	XML Tools	83
7.1	Internet Explorer 6, Mozilla/Firefox	83
7.2	W3 Online Validator	83
7.3	RXP	84
8	Changelog	84
8.1	since February 2008	84
8.2	since June 2006	84

1 Introduction

A Corelatus GTH is a rack-mounted system for interfacing to E1/T1 lines used for voice, signalling and data. The GTH is controlled by sending it commands over an ethernet interface. This note defines the command interface.

Impatient? Skip to the command reference starting on page [10](#). Looking at the commands and examples is the quickest way to see what's possible.

1.1 Three Typical Applications

Using the command interface, you can build a scalable voicemail system. The GTH handles:

- Recording timeslots for later playback, for instance as the message left in a mailbox.
- Switching timeslots among the GTH's 8 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.

A second use of the command interface is to build a signalling analysis system where the GTH connects to a G.772 monitor point to sniff and decode SS7-MTP2, ATM, LAPD, Frame Relay and CAS signalling. The decoded signalling can be sent to an external system for analysis or logging.

A third application is to build a voice logging system where the GTH forwards a copy of selected conversations to an external system for storage. 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.

2 Principles and Terms

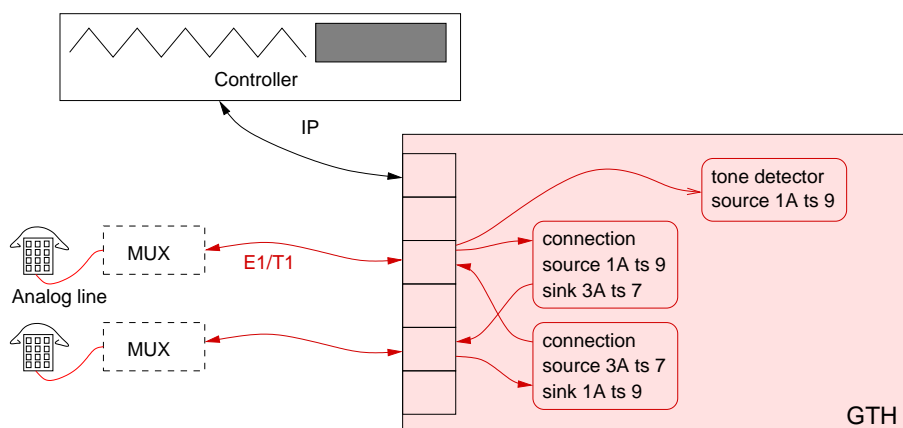
A *GTH* is one self-contained module from Corelatus. It has 6 RJ45 connectors on the front. The leftmost two RJ45 connectors are standard 10/100Mbit ethernet. The other four RJ45 connectors each have two E1/T1 interfaces.

A *controller* is the system controlling, i.e. issuing commands, to the GTH. It could be a Java system running on a 1U SUN 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: E1/T1 interfaces, the CPU and the DSP are 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. Multiple API sockets can be open at the same time, the GTH will serve them 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:

<code>custom</code>	Modifies the software on the GTH module. This lets you put your own logo on the in-built webserver, for instance.
<code>delete</code>	Remove a previously started job
<code>install</code>	Install software on the GTH module's flash memory
<code>new</code>	Create a job. There are jobs which perform signalling, for instance <code>atm_aal5_monitor</code> and <code>lapd_layer</code> , and jobs which manipulate audio, such as <code>connection</code> and <code>player</code> . Normally, jobs run until the <code>delete</code> command kills them, or the current API session is closed by shutting down the port 2089 socket.
<code>nop</code>	The no-operation command
<code>query</code>	Check the status of jobs and resources in the GTH
<code>reset</code>	Restart part of the GTH
<code>set</code>	Change a resource's attributes
<code>takeover</code>	Transfer control of a job from one controller to another
<code>update</code>	Change parameters in a job
<code>zero</code>	Set counters and timers to zero

Section 3 (p. 10) 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`.
- `state` The successful reply to `query` commands.

2.3 Events

The GTH informs the controller of asynchronous events, i.e. events which didn't happen 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.

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° C.
- `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.
- `l1_message` The Layer 1 state of an E1/T1 span has changed.
- `slip` A E1/T1 span has 'slipped'.
- `sync_message` The E1/T1 sync subsystem has changed state.

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

<code>atm_message</code>	An ATM channel changed state
<code>fatality</code>	A job died
<code>f_relay_message</code>	A frame relay channel changed state
<code>info</code>	An informative message, for instance an indication of progress during software upgrade
<code>l2_alarm</code>	A layer 2 signalling job has altered its alarm state
<code>l2_socket_alert</code>	A TCP socket carrying L2 signalling to a controller has encountered a problem
<code>lapd_message</code>	An LAPD timeslot changed state
<code>level</code>	A voice level detector has tripped
<code>message_ended</code>	A voice prompt has completed
<code>mtp2_message</code>	An MTP-2 timeslot changed state
<code>tone</code>	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.

2.4 Voice

The GTH's connection to the telephone network is via its eight E1/T1 interfaces. 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 3.11) 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 the `new player` command (section 3.17) and record a timeslot using `new recorder` (section 3.18). 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 the `new tone_detector` command (section 3.21). The DTMF tones could be a dialled number, or they could be the user navigating in a menu system.

2.5 Signalling

GTH can monitor (passively sniff) all common layer 2 signalling protocols used on E1/T1 links: SS7-MTP-2, ATM AAL5 (used in SS7-HSSL), ATM AAL2, ATM AAL0, frame relay (used in GSM GPRS), 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 L3 support to set up and tear down calls.

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.

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.

Multiple signalling jobs can share one socket to the controller by all indicating the same IP address and port number to the socket. Different channels can be distinguished by the tag value supplied by the controller in the `new` command.

The rest of the header is protocol-specific and defined separately for each protocol, for instance section 3.12 shows the complete header for frame relay.

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:

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

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 64kbit/s MTP-2 link, 7000 for a 56kbit/s ANSI MTP-2 link, 2000 for a 16kbit/s subrate LAPD link and 240000 (for ATM-over-E1 links).

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

$$average_load(t) = 100 \times \frac{M_t - M_{t-30s}}{30 \times B} \quad (2)$$

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

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

2.6 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. These features eliminate several classes of difficult problems. 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 port which doesn't exist:

```
⇒<set name="pcm972">
  <attribute name="status" value="enabled"/>
</set>

⇐<error reason="bad argument">invalid PCM</error>
```

These sorts of errors are fairly common during development. They're easy to deal with as long as 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 3.23).

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 software images. (see section 6).

A second, 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 interfaces, are not returned to their default state—i.e. everything done via `set` is remembered. Jobs started by other API sockets are not terminated either.

GTH Crashes

If both 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 <http://www.corelatus.com/contact.html>

3 Command Reference

This chapter describes each GTH command in detail. Each section starts with a semi-formal 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` tag 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` tag always contains at least one `attribute` tag; the dots indicate that you can have more than one `attribute` tag. The `attribute` tag has two string parameters, both bold and therefore both mandatory.

A few commands, such as `update` (section 3.27) 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 leftwards arrow:

```
⇐ <ok/>
```

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. are described in this section. The DTDs can be downloaded from

<http://www.corelatus.com/gth/api/>

A validating XML parser such as RXP[2] can use the DTDs to find syntax errors in your XML commands. Section 7 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 controller, 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` ([3.26](#))

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* can be 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	<i>physical</i> numbering uses the E1/T1 interface names pcm1A, pcm1B, pcm1C, pcm1D, ... pcm4D, where the number corresponds to the RJ45 port and the letter to the position within that port. <i>sequential</i> numbering uses the scheme pcm1, pcm2, pcm3, pcm4, pcm5 ... pcm16.
board PCM LED assignment	universal, sequential	GTH 2.x cards used in monitoring applications have four PCM receivers in each RJ45 connector, but only two LEDs. The default <i>universal</i> scheme maps PCM1A and PCM1C to the left LED and PCM1B and PCM1D to the right LED, which is backwards compatible with (obsolete) IEB modules. The <i>sequential</i> scheme maps 1A and 1B to the left and 1C and 1D to the right.
http_server		All the attributes in the <code>http_server</code> resource may be customised, see section 4.10 .
os remote login	enabled, disabled	
os API whitelist	IP addresses	A space-delimited list of IP addresses from which the API can be accessed.

Example: A start script

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

<custom name="http_server">
  <attribute name="pcm_overview" value="http://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.

See also: `install` (3.4)

Example: Altering PCM numbering with a custom command at runtime

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

Keep in mind that sending in `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` and `query` (schedule query)

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 **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 <http://www.corelatus.com/gth/releases/>

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

Name	Description	Content-type
<code>failsafe_image</code>	the failsafe flash image	binary/filesystem
<code>system_image</code>	the normal flash image	binary/filesystem
<code>httpd_passwd</code>	the webserver password file	binary/file
<code>logo</code>	the webserver logo	binary/file
<code>start_script</code>	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` 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
<code>logo</code>	The logo displayed in the left margin on most of the web pages. It is an image file in PNG format.
<code>https_passwd</code>	A file containing username/password pairs for the in-built webserver
<code>start_script</code>	A customisation file executed when the GTH starts. Section 3.2 describes the syntax.

See also: `custom` (section [3.2](#)) and section [6](#), which has a step-by-step example of how to install a new release.

3.5 new atm_aal0_monitor

```

⇒ <new>
  <atm_aal0_monitor
    cell='yes'
    oam_cell='no'
    corrupt_cell='no'
    idle_cell='no'
    scrambling='yes'
    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>
⇐ <job id=string/>

```

An `atm_aal0_monitor` monitors the interface above the ATM layer on ATM-over-E1 links, as specified in ITU-T I.432.3 and I.432.1 and ANSI T1.111.1 section 2.2.3A.

Any combination of the timeslots within one span can be used to make up a channel. In practice, E1 installations use timeslots 1–15 and 17–31 while T1 installations use timeslots 1–24.

`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.

`buffer_limit`: If the socket carrying the signalling to the controller has a backlog greater than this limit, the GTH issues an `event`.

`average_period`: specifies the time, in seconds, the average load is calculated over.

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

Counters and Indicators

Counters and indicators can be read by issuing a `query` command on the signalling job.

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 <code>sync</code> or <code>hunt</code>
	n_sync, t_sync,	The number of times the 'sync' state was entered and the number of milliseconds spent in the 'sync' and 'hunt' states, respectively.
	t_hunt	

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 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:

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 = 4				reserved				CR		reserved				Ver	
octet 0x06	Timestamp															
octet 0x08																
octet 0x0a																
octet 0x0c	GFC				VPI				VCI...							
octet 0x0e	...VCI				PTC											

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. Within one channel, this value is nondecreasing.

The **GFC**, **VPI**, **VCI** and **PTC** fields make up the ATM cell header. ITU-T I.361 [3] describes the purpose of these fields in ATM networks.

See also: section [2.5](#)

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, an XML event is sent:

```
⇒ <event>
    <atm_message id=string value='hunt'|'sync'/>
</event>
```

The GTH 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, i.e. the buffer is more than half full:

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

Example: Monitoring a standard 1980kbit/s ATM AAL0 channel

```
⇒ <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: 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>
```

3.6 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>
⇐ <job id=string/>

```

An ATM AAL2 monitor monitors the interface above the AAL2 layer on ATM-over-E1 links. The GTH allows 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`: If the socket carrying the signalling to the controller has a backlog greater than this limit, the GTH issues an `event`.

`average_period`: specifies the time, in seconds, the average load is calculated over.

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

Counters and Indicators

Counters and indicators can be read by issuing a `query` command on the signalling job.

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

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 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:

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 = 6			reserved			CR		reserved				Ver			
octet 0x06	Timestamp															
octet 0x08																
octet 0x0a																
octet 0x0c	GFC				VPI						VCI...					
octet 0x0e	...VCI										PTC					
octet 0x10	CID						LI				UUI					
octet 0x12	UUI			HEC				(Payload)								

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 [4]. 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, an XML event is sent:

```
← <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, i.e. the buffer is more than half full:

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

Standard: ITU-T I.363.2

3.7 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>
⇐ <job id=string/>

```

An ATM AAL5 monitor monitors the interface above the GPCS sublayer in AAL5 on ATM-over-E1 links.

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.

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

buffer_limit: If the socket carrying the signalling to the controller has a backlog greater than this limit, the GTH issues an event.

average_period: specifies the time, in seconds, the average load is calculated over.

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

Counters and Indicators

Counters and indicators can be read by issuing a `query` command on the signalling job.

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

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 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:

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 = 5		res.		CG	AB	CR	reserved						Ver		
octet 0x06	Timestamp															
octet 0x08																
octet 0x0a																
octet 0x0c	GFC				VPI						VCI...					
octet 0x0e	...VCI						PTC									
octet 0x10	CPCS-UU						CPI									
octet 0x12	CPCS-Length															
octet 0x14	CPCS-CRC...															
octet 0x16	...CPCS-CRC															

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

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. Within one channel, 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 [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, an XML event is sent:

```
← <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, i.e. the buffer is more than half full:

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

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

Example: Monitoring a standard 1980kbit/s ATM AAL5 channel. 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: 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="cdu_o" value="134"/>
        <attribute name="n_sync" value="1"/>
        <attribute name="t_sync" value="134134613"/>
        <attribute name="t_hunt" value="1231"/>
        <attribute name="current state" value="sync"/>
        <attribute name="current load" value="18"/>
        <attribute name="average load" 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>

```

3.8 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. The GTH reports each time the bits change, and stay changed for at least 10ms.

CAS line signalling normally occupies timeslot 16 on an E1. The GTH accepts 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 an `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 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:

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 = 2				reserved			
octet 0x06	Timestamp															
octet 0x08																
octet 0x0a																
octet 0x0c	reserved															
octet 0x0e	Channel								Bits							

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, so *c* and *d* will normally be fixed at 0 and 1, respectively.

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"/>
```

3.9 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 an `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 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	Timestamp															
octet 0x08																
octet 0x0a																
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).

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"/>
```

3.10 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 dialtone, or some voice, for instance a voicemailbox greeting.

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. It must have content type `binary/audio` and it may be up to 500000 octets (60 seconds of audio) long.

Once a clip exists, a `player` can play it.

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 which are sensitive to such delays can use a second socket for the timing-critical operations.

Example: Defining a clip

```
⇒<new><clip id="dtmf9"/></new>
```

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

```
⇐<job id="clip dtmf9"/>
```

3.11 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.

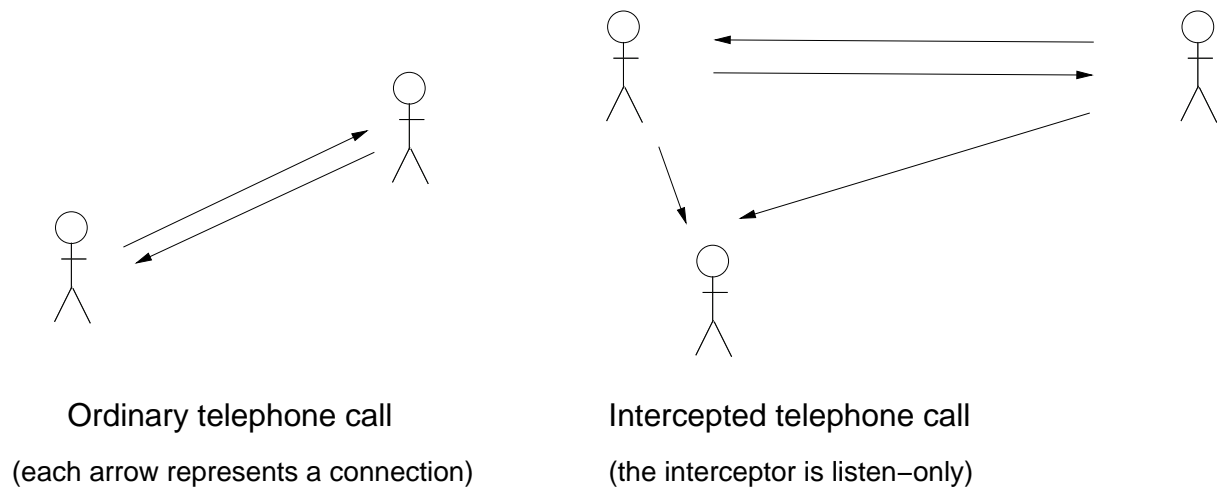


Figure 1: Two examples of call switching

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"/>
```

3.12 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 passively extracts frame relay signal units from a frame relay link. The most common input bandwidths are 256kbit/s and 1980kbit/s. The GTH supports all multiples of 64kbit/s.

All timeslots in a channel must be from the same E1/T1 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: If the socket carrying the signalling to the controller has a backlog greater than this limit, the GTH issues an *event*.

average_period: specifies the time, in seconds, the average load is calculated over.

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

Counters and indicators can be read by issuing a *query* command on the signalling job.

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 <i>up</i> or <i>down</i>
	n_up, t_up, n_down, t_down	The number of times the <i>up</i> and <i>down</i> 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`. Whenever the link switches from one to the other, an XML event is sent:

```

<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, i.e. the buffer is more than half full:

```

<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 signal units extracted from a link using an `fr_monitor` are forwarded to the external host given by the `ip_addr` over a TCP socket separate to the socket used

for API commands. The GTH expects the external host to be listening on this socket before the `new` command is issued.

Each signal unit is delivered on a dedicated TCP connection 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		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`: is specified by the user. It is used for identifying the channel.

`protocol`: indicates which protocol is being monitored. 0 = MTP2, 1 = LAPD, 2 = Frame Relay

`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 started, measured in milliseconds since the unix epoch. Within one channel, it is a monotonically increasing integer.

Example: Monitoring a quad-timeslot (256kbit/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"/>

```

3.13 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 (ISDN Layer 2) layer requires both a sink and source. Normally these will be the same E1/T1 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

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"/>

```

3.14 new lapd_monitor

```

⇒ <new>
  <lapd_monitor
    su='yes'
    esu='no'
    load_limit='50'
    buffer_limit='256000'
    average_period='30'
    tag='0'
    timeout='15'
    ip_addr=int.int.int.int
    ip_port=int>
    <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. The GTH forwards the signal units to the TCP port specified by the IP address and port number, using the protocol described in the frame relay section on page 36.

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

buffer_limit: If the socket carrying the signalling to the controller has a backlog greater than this limit, the GTH issues an *event*.

average_period: specifies the time, in seconds, the average load is calculated over.

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

timeout: The number of seconds to set the timer for the state transition from the *up* to the *down* state.

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 <i>up</i> or <i>down</i>
	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*. Whenever the link switches from one to the other, an XML event is sent:

```
← <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, i.e. the buffer is more than half full:

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

Standards: ITU-T Q. 921

Example: Monitoring a 64kbit/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 a 16kbit/s LAPD timeslot

Some GSM interfaces use 16kbit/s subrate LAPD monitoring. In this example, the signalling is in bits 4 and 5 in each octet, i.e. a 16kbit/s channel. The GTH also supports subrate LAPD monitoring at 32kbit/s.

```
⇒ <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"/>
```

3.15 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 level (the power on that timeslot) crosses the specified threshold, an event appears on the socket:

```

⇐ <event>
  <level id=string state=string/>
</event>

```

threshold: -60 .. +9. The power threshold, in dB, relative to the "digital milliwatt". +9 is the loudest possible sound, -60 is as silent as a timeslot can be.

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.

Example: Level Detection

```

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

⇐ <job id="lede1"/>

```

3.16 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. Both single timeslot (56 or 64kbit/s) and a list of timeslots (ITU-T Q.703 Annex A, or ANSI T1.111.1 section 2.2.2) are supported.

The GTH forwards the signal units to the TCP port specified by the IP address and port number, using the protocol described in the frame relay section on page 36.

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: sets 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 the same for all *mtp2_monitor* jobs, i.e. it is always set to either the default or whatever was specified in the most recently created *mtp2_monitor*.

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-specified integer in the range [0, 65535].

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

Counters and Indicators

Counters and indicators can be read by issuing a `query` command on the signalling job.

Category	Members	Description
Packet counters	<i>n_fisu</i> , <i>n_issu</i> , <i>n_msu</i> , <i>n_esu</i> , <i>n_rsu</i>	The number of packets of different MTP-2 types which have arrived on the interface. <i>n_esu</i> is the number of errored packets, which is defined as all packets which are too short, too long, non octet-aligned or incorrect CRC. <i>n_rsu</i> 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	<i>fisu_o</i> , <i>issu_o</i> , <i>msu_o</i> , <i>esu_o</i> , <i>rsu_o</i>	The total number of octets in the packet types above.
Load meters	<i>current</i> , <i>average</i> and <i>maximum</i> <i>load</i>	
State	<i>current state</i> <i>n_in service</i> , <i>n_congested</i> , ... <i>t_in service</i> , <i>t_congested</i> , ...	(see table below) The number of times the link was in each state 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, an XML message is sent 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, i.e. the buffer is more than half full:

```

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

```

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

Example: Monitoring an ordinary 64kbit/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 56kbit/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: 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>
```

3.17 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 could therefore be voice, tones or even raw signalling data.

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

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 in-built flow control, so the data being sent isn't timing sensitive.

`loop`: when playing from a `clip`, `loop` 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>

```

Error handling

The GTH might be unable to connect to the specified socket, for instance because that socket was not in a listening state. When this happens, the GTH issues a `fatality` event:

```

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

```

After a player has started, the TCP socket must provide at least enough data to fill a timeslot, i.e. 8kByte/s. If there isn't enough data, the GTH's buffer will underrun and the GTH will send this 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.

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.

Example: Playing a TCP stream on a PCM 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 a PCM 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. It is sent 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"/>
```

The optional `loop` parameter can be used to loop the audio from a clip forever.

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"/>
```

3.18 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>
⇐ <job id=string/>
```

A recorder streams data from a PCM timeslot to a remote socket via TCP. The recording can be stopped by deleting the recorder or, less cleanly, by closing the socket. To use a recorder:

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

If the controller doesn't read fast enough, the GTH will send a `fatality` event:

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

Example: Forwarding a PCM timeslot to a TCP socket

```
⇒ <new>
    <recorder>
      <pcm_source span="3A" timeslot="1"/>
      <tcp_sink ip_addr="172.16.2.1" ip_port="2222"/>
    </recorder>
  </new>
⇐ <job id="strr713"/>
```

An octet-by-octet copy of the given timeslot is sent to the TCP socket 2222. This could be used to record messages and greetings to semi-permanent storage in a voicemail system. A logging system could permanently record and store conversations. A network monitoring system could record failed calls or failed signalling for later analysis.

For prototyping on unix-like systems, the 'netcat' tool can take care of the socket operations:

```
netcat -p 2222 -l > /tmp/audio_data
```

3.19 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/>

```

PRELIMINARY INFORMATION.

SS5 signalling is implemented for field testing. You need at least gth2_system_33a to use it. This interface may change when adopted into the supported API.

An SS5 line signalling monitor detects the SS5 tones F1 and F2. SS5 line signalling is normally used together with SS5 register signalling.

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 an `ss5_linesig_monitor` is forwarded to a socket on a remote machine defined by `ip_addr` and `ip_port`. This socket is independent of the API socket. The GTH expects the external host to be listening on this socket before the `new` command is issued.

Each Line signalling 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 = 3				reserved			
octet 0x06	Timestamp															
octet 0x08																
octet 0x0a																
octet 0x0c	reserved															
octet 0x0e	Type								Tone							

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-line-signalling-tone, 2=end-of-tone

Tone: 0: 2400Hz 1: 2600Hz 2: 2400 + 2600 Hz

Standards: ITU-T Q.144 (previously CCITT C5)

Example: Monitoring SS5 line signalling

```
⇒ <new>
    <ss5_linesig_monitor tag="1234" ip_addr="172.16.2.1" ip_port="1234">
        <pcm_source span="2A" timeslot="3"/>
    </ss5_linesig_monitor>
</new>
⇐ <job id="s5lm98"/>
```

3.20 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/>

```

PRELIMINARY INFORMATION.

SS5 signalling is implemented for field testing. You need firmware gth2_system_33a or later to use it. This interface may change when adopted into the supported API.

An SS5 register signalling monitor detects the 2-of-6 SS5 register signalling digits. SS5 register signalling is normally used together with SS5 line signalling.

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 an `ss5_registersig_monitor` is forwarded to a socket on a remote machine defined by `ip_addr` and `ip_port`. This socket is independent of the API socket. The GTH expects the external host to be listening on this socket before the `new` command is issued.

Each Register signalling 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 = 3				reserved			
octet 0x06	Timestamp															
octet 0x08																
octet 0x0a																
octet 0x0c	reserved															
octet 0x0e	Type = 3								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: 3 (Register signalling digit)

Digit: 0–12 correspond to digits 0–12 as per table 5 of ITU-T Q.151. 13, 14 and 15 correspond to KP1, KP2 and ST, respectively.

Standards: ITU-T Q.154 (previously CCITT C5)

Example: Monitoring SS5 register signalling

```
⇒ <new>
    <ss5_registersig_monitor tag="1234" ip_addr="172.16.2.1"
ip_port="1234">
    <pcm_source span="2A" timeslot="3"/>
    </ss5_registersig_monitor>
</new>
⇐ <job id="s5rm7768"/>
```

3.21 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. The frequency and length attributes should not be specified for DTMF.

type: **custom** detects tones of user-selectable frequency (in Hertz) and minimum length (in milliseconds).

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>
```

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>
  <resource name="cpu">
    <attribute name="load" value="0.3"/>
  </resource>
</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"/>

```

3.22 **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 [5.3](#) describes how to supervise a GTH

3.23 query

```

⇒ <query>
    <resource name=string/>
    ...
</query>
⇒ <query>
    <job id=string/>
    ...
</query>
⇐ <state>
    <atm_aal0_monitor id=string owner=string>
        <attribute name=string value=string/>
        ...
    </atm_aal0_monitor>
    ...
</state>
⇐ <state>
    <atm_aal2_monitor id=string owner=string>
        <attribute name=string value=string/>
        ...
    </atm_aal2_monitor>
    ...
</state>
⇐ <state>
    <atm_aal5_monitor id=string owner=string>
        <attribute name=string value=string/>
        ...
    </atm_aal5_monitor>
    ...
</state>
⇐ <state>
    <controller id=string ip_addr=int.int.int.int ip_port=int timeout=string
        backups=string/>
    ...
</state>
⇐ <state>
    <error reason=string>
        <pcdata/>
    </error>
    ...
</state>
⇐ <state>
    <f_relay_monitor id=string owner=string>
        <attribute name=string value=string/>
    ...

```

```

    </f_relay_monitor>
    ...
</state>
⇐ <state>
    <job id=string owner=string/>
    ...
</state>
⇐ <state>
    <lapd_monitor id=string owner=string>
        <attribute name=string value=string/>
    ...
</lapd_monitor>
    ...
</state>
⇐ <state>
    <mtp2_monitor id=string owner=string>
        <attribute name=string value=string/>
    ...
</mtp2_monitor>
    ...
</state>
⇐ <state>
    <player id=string owner=string octets_played=int/>
    ...
</state>
⇐ <state>
    <resource name=string>
        <attribute name=string value=string/>
    ...
</resource>
    ...
</state>

```

A `query` is used to obtain information about the GTH's internal state.

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

```

⇒ <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="pcmlA"/>
      <resource name="pcmlB"/>
      <resource name="pcmlC"/>
      ...
    </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"/>
      <job id="m2mo15"/>
    </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 transferred over the API using queries.

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

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

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

3.24 **reset**

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

```
⇐ <ok/>
```

The resource must be *cpu*. This command reboots the GTH.

3.25 set

```
⇒ <set name=string>
    <attribute name=string value=string/>
    ...
  </set>
⇐ <ok/>
```

Resources can be reconfigured at any time using set. 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.

Example: Enabling an E1

```
⇒ <set name="pcm1A">
    <attribute name="status" value="enabled"/>
    <attribute name="framing" value="multiframe"/>
  </set>
⇐ <ok/>
```

Example: Enabling a T1

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

Example: Explicitly setting up an E1/T1 sync source

By default, the GTH scans its ports (lowest port number first) until it finds an E1/T1 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/>
```

3.26 takeover

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

```
⇐ <ok/>
```

All jobs have an *owner*. Initially, the owner is the socket which sent the command to create them. 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/>
```

3.27 update

```
⇒ <update>
    <controller timeout=int backups=string/>
</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>
```

```
⇐ <ok/>
```

Update allows a job's attributes to be changed. The changeable attributes on a controller are *timeout* and *backups*. *timeout* is in milliseconds, with 0 meaning disabled. *backups* is a space-delimited list of controller job-IDs. Section 5.3) describes how to build fault-tolerant systems by setting these attributes to non-default values.

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

See also: `mtp2_monitor` (section 3.16)

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 can be 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/>
```

3.28 zero

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

```
⇒ <zero>  
    <job id=string/>  
  </zero>
```

```
⇐ <ok/>
```

Reset counters and timers on

- E1/T1 L1 resources (pcm1A, pcm2A, ...)
- Layer 2 monitors (CAS, MTP-2, LAPD, frame relay and ATM)

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

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

```
⇐ <ok/>
```

4 Resources

Resources are parts of the GTH which always exist. A resource's attributes can be modified using the `set` command (section 3.25) and read using the `query` command (section 3.23).

4.1 Inventory and Schedule

A list of the GTH's resources is always available by querying the *inventory* resource.

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

Page 60 has an example of these two types of query.

4.2 E1/T1 links

E1 and T1 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. GTH 2.x systems have eight full-duplex E1/T1 interfaces named `pcm1A`, `pcm1B`, `pcm2A`,, `pcm4B`. In monitoring applications, GTH 2.x systems have 16 listen-only E1/T1 interfaces named `pcm1A`, `pcm1B`, `pcm1C`, `pcm1D`, . . . , `pcm4D`.

The configurable attributes on an E1/T1 link are:

Attribute	Possible values in query	Allowed values in set
<code>status</code>	disabled, LOS, LFA, LMFA, AIS, RAI, OK	enabled, disabled
<code>mode</code>	E1, T1	"
<code>framing</code>	E1 mode: doubleframe, multiframe T1 mode: extended superframe, superframe	as for query as for query
<code>impedance</code>	120, 100, 75	"
<code>line_coding</code>	E1 mode: HDB3 T1 mode: B8ZS, AMI	" "
<code>idle_pattern</code>	0–255	E1 mode: 84 (0–255) T1 mode: 127 (0–255)
<code>monitoring</code>	false, true	"
<code>tx_enabled</code>	true, false	"

status shows the link's current state, which is either OK or an error state:

LOS	Loss of signal. The incoming line doesn't have a signal at all, or a signal too weak to detect. In North America, this condition is often called a <i>red alarm</i> .
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 E1 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 <i>blue alarm</i> in North America.
RAI	Remote alarm indication. The equipment at the other end of the link is signalling that <i>it</i> is not in the OK state. This is often called a <i>yellow alarm</i> in North America.

To enable a link, `set` the status to *enabled* and supply all non-default attributes. The default value for each attribute is listed first in the table above. Section 3.25 has an example of how to enable an interface in T1 mode and E1 mode.

mode is used to select E1 or T1 operation. T1 is mainly used in North America, whereas most of the rest of the world uses E1. In T1 mode, there are 24 useable timeslots, numbered 1–24. In E1 mode, there are 31 useable timeslots, numbered 1–31.

framing can usually be left at the default setting.

impedance selects the receiver and transmitter impedance, which depends on the type of cable used. T1 installations always use 100 ohm, E1 on twisted pair uses 120 ohm and E1 on coaxial cable 75 ohm.

idle_pattern selects the octet sent on unused timeslots. Common values in practice are 0x54 and 0xff.

monitoring is used for non-intrusive monitoring of links, typically through a resistive pad as per ITU-T G.772. Enabling monitoring increases the signal sensitivity by 20dB and also disables transmit.

tx_enabled is only used on GTH 2.x modules with 16 E1/T1 RX inputs. When set to *false*, it disables transmit for that E1/T1. Attempting to enable all four E1/T1 receivers in one RJ45 connector without setting *tx_enabled* to *false* will result in a *conflict* error.

The read-only attributes on an E1/T1 link are:

Attribute	Notes
slip_positive	The number of slips where an extra frame was inserted.
slip_negative	The number of slips where a frame was skipped.
frame_error	The number of times a framing error occurred.
code_violation	The number of line symbols which were in violation of the line coding rules.
crc_error	The number of L1 CRC errors. Some framing modes, for instance <i>doubleframe</i> do not have an L1 CRC, so this counter will always be zero when the framing is <i>doubleframe</i> .
LFA_entered	The number of times the LFA state was entered. There is a similar counter for each of the OK, LOS and RAI states.
LFA_duration	The cumulative number of milliseconds the link was in the LFA state. There are analogous counters for each of the OK, LOS and RAI states.

Each E1/T1 interface broadcasts an event to all controllers whenever its state changes:

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

4.3 Sync Sources

GTH can synchronise its E1/T1 frequency reference to any one of the E1/T1 interfaces, or it can use its internal, temperature-compensated frequency source.

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.

Attribute	Possible values in query	Allowed values in set
mode	auto, manual	read-only
source	none, pcm1A, pcm2A,	as for query, plus <i>auto</i>
state	trying, locked, dead	read-only
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 sync state *trying* indicates that the GTH is attempting to sync to the given E1/T1 port, when it successfully syncs, the state changes to *locked*. If the port does not contain a usable sync, the state changes to *dead*.

If the sync source is set to *none*, the GTH's internal, temperature-compensated frequency source is used.

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`.

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 sync system changes states, the GTH sends an event showing the new state:

```
← <event>
    <sync_message state="trying|locked on|dead"/>
</event>
```

In addition, the pseudo-state `limit_reached` can be reported when the sync system reaches the limit of its frequency adjustment range.

4.4 Ethernet Interfaces

The GTH's two Ethernet interfaces are named *eth1* and *eth2*.

Attribute	Possible values in query	Allowed values in set
MAC address	a string of hex digits	read-only
IP4 address	a dotted quad	as for query
IP4 mask	a dotted quad	"
default gateway	a dotted quad	"
locked	true, false	"
collisions	an integer	read-only
TX errors	an integer	"
TX load	an integer	"
TX average load	an integer	"
TX maximum load	an integer	"
TX octets	an integer	"
link status	up, down or unknown	"
link speed	10, 100 or unknown	"
link duplex	HD, FD or unknown	"
load limit	an integer	an integer

The *IP4 address*, *mask*, *gateway* and *locked* settings persist across reboots. In most cases, the IP4 address and mask should be set in the same operation:

```
⇒ <set name="eth1">
    <attribute name="IP4 address" value="128.250.22.3"/>
    <attribute name="IP4 mask" value="255.255.255.0"/>
</set>
⇐ <ok/>
```

In some situations, for instance demonstrations, the *default gateway* setting can be used to make the GTH module accessible in a routed network. Setting the *default gateway* attribute in live systems is not recommended because of potential performance and security problems.

The *locked* attribute is used to control configuration changes from the webserver. If *locked* is true, the IP address and mask cannot be changed from the web server. The API is not affected by the *locked* attribute.

The special value *none* can be used to disable an interface or clear the default gateway. Disabling both ethernet interfaces is not allowed.

The *TX load* and *TX load limit* are expressed as a percentage of nominal capacity; hence the default limit of 40 corresponds to 40 MByte/s on a 100 Mbit interface, but only 4 MByte/s if the interface is running at 10 Mbit/s

Events

Whenever the *TX load* exceeds the load limit, GTH sends an event:

```
⇐ <event>
    <alarm name="eth1" attribute="TX load" state="set"|"clear" value=int/>
</event>
```

4.5 CPU

The GTH CPU includes the attached memory:

Attribute	Possible values in query	Allowed values in set
load	the unix load average	read-only
total memory	integer, in octets	read-only
free memory	integer, in octets	read-only
load limit	an integer	an integer

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.

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=int/>
```

```
</event>
```

4.6 Board

The rest of the attributes which affect a whole module are grouped in the *board* resource:

Attribute	Possible values in query	Allowed values in set
LED mode	normal, flashing	normal, flashing
PCM LED assignment	universal, sequential	universal, sequential
power consumption	power in W	read-only
power source	A, B, or the string 'both'	"
ROM ID	A hex number	"
temperature	temperature in degrees Celsius	"
voice coding	alaw, mulaw	alaw, mulaw
architecture	gth1, gth2.0, gth2.1	read-only

The LED mode can be changed to *flashing*. This makes all LEDs on the board flash, which can be useful when looking for a particular GTH in a rack of GTHs.

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.

power consumption indicates the module's current power consumption. This value typically fluctuates between readings, depending on what the GTH is doing at the instant of reading.

power source indicates which of the two power inputs currently has power. If both inputs are powered up, the GTH uses the one with the higher voltage.

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 well ventilated rack, the PCB temperature is typically 15–25° above the air temperature.

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*.

The *architecture* reveals the hardware generation. In releases before `gth2.system_33a`, both `gth2.0` and `gth2.1` are reported as 'gth2'.

Events

If the temperature goes outside the allowed range (10 – 60° Celsius), an event is broadcast to all controllers:

```

⇐ <event>
  <alarm name="board" attribute="temperature"
    state="set"|"clear" value=int/>
</event>

```

If the power is lost on one of the two inputs:

```

⇐ <event><alert reason="power A lost"|"power B lost"/></event>

```

If power returns on a power input:

```

⇐ <event><alert reason="power A recovered"|"power B recovered"/></event>

```

4.7 OS

The GTH operating system has a number of attributes:

Attribute	Possible values in query	Allowed values in set
boot mode	normal, failsafe	normal, failsafe
remote login	enabled, disabled	enabled, disabled
API whitelist	a list of IP addresses	as for query
uptime	a time	read-only
last restart	a time	read-only
restart type	hard, soft, watchdog	read-only
restart cause	kernel reboot, power failure, unknown	read-only

The GTH includes two complete installations of the operating system and application software. 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/>

```

uptime is the number of seconds since the last reboot, the wall clock time (UTC!) of that reboot is given by *last restart*.

restart type and *restart cause* indicate why the GTH restarted. This is useful for forensics, i.e. answering “why did the system restart?”.

4.8 System Image and Failsafe Image

The GTH has two completely independent versions of its on-board software, each with its own three attributes: independently.

Attribute	Possible values in query	Allowed values in set
locked	true, false	true, false
version	string	read-only
busy	true, false	read-only

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.

Section 6 covers upgrading the internal software.

4.9 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. Internal system status messages are logged. This setting is not recommended for systems in a production environment because it reduces performance and wraps logs quickly.
changes	As for <i>all</i> , except successful <code>query</code> commands, <code>nop</code> commands and all XML responses are suppressed. This setting is the default.
info	All successful XML commands are suppressed

The resources *standby_application_log* and *standby_system_log* provide access to the logs from the software image which is *not* running.

4.10 HTTP Server

The on-board webserver can be configured to override the default pages. Any page accessible by the client browser can be substituted for the top-level pages:

Attribute	Possible Values	Purpose
enabled	true, false	Enable/disable the webserver
top	A URL	The page reached at http://gth:8888/
ethernet	A URL	The ethernet configuration page
l2_status	A URL	Layer 2 (MTP2 and LAPD) status page
os	A URL	The operating system and logs page
pcm_overview	A URL	The PCM interface overview page
status	A URL	The top-level system status page. By default, this is the same as <i>top</i>
passwords	See below	Alter the passwords used to access the in-built webserver.

The top-level page is overridden by using an HTTP 302 response. The other pages are overridden by rewriting the links in the displayed pages.

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/>
```

In addition, counter zeroing buttons can be enabled on the WWW interface for PCM L1 and L2. 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.

Attribute	Possible Values	Purpose
pcm_L1_zero_buttons	false, true	If true, buttons are shown on the WWW interface which can be used to reset the PCM L1 counters.
pcm_L2_zero_buttons	false, true	If true, buttons are shown on the WWW interface which can be used to reset the PCM L2 counters.

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[7] 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.

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 busy	Avoid wasting time debugging old bugs. <i>false</i> is bad, it means the GTH is in failsafe mode. Proceeding is pointless, manual investigation is in order.
os	restart type	If it is <i>watchdog</i> , the card crashed. Send a bug report to Corelatus.
os	restart cause	If it is <i>power failure</i> , why was there a power failure?
schedule		The schedule is a list of jobs currently running on the GTH. In most systems, this should be empty at startup, apart from 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.

In all GTH applications, a correctly functioning E1/T1 (PDH) network is crucial. Once a system is running, all E1/T1 interfaces should stay in the `OK` state. If they leave the `OK` state, `gth` sends an `ll_message` to warn that calls and signalling are likely to be degraded or fail completely.

A second symptom of E1/T1 network problems is difficulty maintaining synchronisation. The GTH reports changes in E1/T1 synchronisation status via the `sync_message`. Per-E1/T1 `slip` messages indicate lost data on that E1/T1, which will degrade speech and signalling.

A good 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 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 global ID 'apicXXX' and implicit local ID 'self', whenever a port 2089 API connection is started. This implicit job can be configured using `update`. One of its attributes is the `timeout`, which controls supervision.

If the `timeout` is nonzero, then the GTH starts a timer after each command is received. The timeout value is given in milliseconds. 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"/>

```

5.4 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 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.

5.5 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 interfaces, 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><backups>
    <job id="play19413"/>
    <job id="cnxn16448"/>
    ...
</backups></event>
```

6 Worked Example: Upgrading

The GTH software can be upgraded both on-site and remotely. There are always two complete software images installed on a GTH. This allows the GTH to recover from a failed upgrade.

6.1 The System and Failsafe Images

The two software images on the GTH are called *system* and *failsafe*. During everyday operation, the system image is used. Check which image is running by querying:

```
⇒ <query>
    <resource name="system_image"/>
</query>
⇐ <state>
    <resource name="system_image">
        <attribute name="version" value="development_release_9"/>
        <attribute name="locked" value="false"/>
        <attribute name="busy" value="true"/>
    </resource>
</state>
```

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 four minutes.
3. The *boot mode* has been manually set to *failsafe* before the most recent boot.

6.2 Upgrading

Boot the failsafe image

Images are not upgraded in-place, so if we want to upgrade the system image, we need to boot *failsafe* first. We do this by setting the *failsafe* boot mode

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

and then issuing a reset

```
⇒ <reset>
```

```
    <resource name="cpu"/>
  </reset>
<=> <ok/>
```

Unlock the system image

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

Install the new release

Upgrades to the image are delivered as standard compressed 'tar' archives.

They are installed into the system using the install command:

```
=> <install name="system_image"/>
<=> <ok/>
```

The actual archive is sent immediately following the install command, in a block with content type 'binary/filesystem'.

The GTH sends an `<ok/>` response after the filesystem has finished transferring. The actual install process continues after the transfer, an event is sent when it completes:

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

Rebooting the system without waiting for the *install_done* message will usually result in a corrupted installation. Start again from the top.

Verify that the install completed

A query of the *system image*, as shown above, reveals the version number of the installed image. An install which has not completed shows up as *empty*.

Finally, we reboot again. If the system fails to boot after five attempts, the failsafe system starts.

6.3 Upgrading the Failsafe System

Upgrading the failsafe image is analogous to upgrading the system image. A special failsafe software release file is used.

7 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.

7.1 Internet Explorer 6, 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 "http://www.corelatus.se/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>
```

will produce 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.

7.2 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.

7.3 RXP

RXP[2] 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 and therefore highly recommended.

8 Changelog

The changelog shows changes made in the past 12–18 months.

8.1 since February 2008

- Updated for `gth2_system_33a`. The GTH can now monitor SS5 signalling and detect custom in-band tones such as FAX and modem tones.
- Corrected errors in the `<install>` command description

8.2 since June 2006

- Large-scale conferences have been removed from the software and from the API manual. (this was a beta feature introduced in 2004)
- The API manual has been revised and updated.
- The information previously contained in the DTD comments has been merged with this manual.

Acronyms

AAL0 ATM Adaptation Layer Zero. A stream of raw ATM cells.

AAL5 ATM Adaptation Layer Five. Defined in [5].

AIS Alarm Indication Signal. See 4.2

API Application Programming Interface.

ATM Asynchronous Transfer Mode. A family of signalling protocols defined in a series of ITU standards including [3]

B8ZS Bipolar Eight Zero Substitution. See section 4.2

CPCS 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)
- DSP** Digital Signal Processor
- E1** A 2Mbit/s PCM link as described in ITU-T G.703.
- ESU** Errored-signal-unit. Part of MTP-2, LAPD and frame relay signalling.
- FISU** Fill-in-signal-unit. Part of MTP-2.
- HDB3** High-Density-Bipolar-3. See section [4.2](#)
- 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.
- L1** Layer 1, e.g. an E1/T1 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](#)
- LMFA** Loss of multi-frame alignment. See section [4.2](#)
- LOS** Loss of signal. See section [4.2](#)
- LSSU** Link-status-signal-unit. Part of MTP-2.
- MSU** Message Signal Unit. Part of MTP-2.
- 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.
- PCM** Pulse Code Modulation (used in 2Mbit E1 and 1.5Mbit T1 connections)
- RAI** Remote Alarm Indication. See section [4.2](#)
- SS7** Signalling System 7. The signalling protocol used in telephone networks.
- SSH** Secure SHell. A remote login protocol.
- T1** A 1.544Mbit/s PCM link as described in ITU-T G.703
- TCP** Transmission Control Protocol. Defined in RFC793.
- UTC** Coordinated Universal Time. A method of defining absolute time. It is almost identical to GMT.
- VCI** Virtual Channel Identifier. Part of the ATM cell address.
- VPI** Virtual Path Identifier. Part of the ATM cell address.
- XML** Extensible Markup Language[1]

References

- [1] *The XML 1.0 Spec (Third Edition)* <http://www.w3.org/TR/REC-xml>
- [2] *The RXP validating XML parser* <http://www.cogsci.ed.ac.uk/richard/rxp.html>
- [3] *ITU-T I.361: B-ISDN ATM layer specification*
- [4] *ITU-T I.363.2: B-ISDN ATM Adaptation Layer: Type 2 AAL*
- [5] *ITU-T I.363.5: B-ISDN ATM Adaptation Layer: Type 5 AAL*
- [6] *ITU-T Q.703: Message Transfer Part Layer 2*
- [7] *RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication*
<http://www.faqs.org/rfcs/rfc2617.html>