Global Connect TCP API Specification

Version 0.1

TABLE OF CONTENTS

1. GLOBAL CONNECT – MODULAR I/O DESIGN CONCEPT

The Global Connect product family has infrared, serial, and relay, and HDMI switching modules. While a single Global Connect chassis may contain any number of each of these modules, from an API perspective each module can be treated as its own unit on its own IP address, and as such its own TCP endpoint.

The modularity of the Global Connect product family offers two significant API benefits. First, it allows for a simple API that doesn't change regardless of how many control ports of each type are contained within a chassis. Second, it allows for backwards compatibility with TCP commands of previous product lines.

2. NETWORK CONFIGURATION

Global Connect is available with either a single port Ethernet (RJ45 100 Mbit/s) or a two port Ethernet switch. Both variants are API equivalent.

The Global Connect can be configured using embedded setup webpages or direct API requests. Factory default settings provide specific predefined configurations. Default settings can be restored at any time by depressing the first (left-most) lightpipe on a module for more than 10 seconds. Factory default settings enable initial or recovery access to the device so it can be configured with the end-user's desired operating network settings.

Global Connect defaults to DHCP IP address mode and simply requires physical connection of an Ethernet network cable to the network. It can then be accessed for customized network configuration as necessary (For example, static IP). If a DHCP server is not available, the Global Connect modules will acquire IP addresses 192.168.1.70 through 192.168.1.(70+(n-1)), where 'n' is the number of control modules in the chassis.

3. NETWORK DISCOVERY

Global Connect provides two methods of network discovery – a periodic UDP beacon and multicast DNS (mDNS) mechanism.

To determine the IP address of a Global Connect already connected to the network, the recommended method is to use the iHelp utility available for download at http://www.globalcache.com/downloads. iHelp has a number of useful functions, including the ability to discover and display all Global Caché devices connected to the network. This is accomplished through a mechanism where the Global Connect sends periodic (at 10 second intervals) beacon messages over the network which contain identifying information such as IP address, firmware revision, and MAC ID. The iHelp utility listens for these beacon messages, then displays the information to the user in a device list. iHelp also provides additional functions such as firmware updates. Please refer to iHelp documentation for further details.

Global Connect's periodic UDP beacon message (used by iHelp) allows discovery of Global Connect units on the network. The beacon is sent as a UDP packet to the multicast IP address 239.255.250.250, multicast group destination MAC 01:00:5E:7F:FA:FA, and destination port 9131. The beacon is sent shortly after successful network connection and then at regular intervals of 10 seconds. Any network device that subscribes to the multicast group will receive the periodic beacon message.

The beacon message has the following format (field values are for example only):

```
AMXB
<-UUID=GlobalCache_000C1E5C0A5C>
<-SN=0000000001>
<-SDKClass=Utility>
<-Make=GlobalCache>
<-Model=GCSL232>
<-Revision=710-4002-00a52.3>
<-Config-URL=http://192.168.1.127>
<-Status=Net-Eth:Linked,Error:MACID invalid>
<-Mod_Addr=1>
<-IO_Class=serial>
<-IO_ID=SL232>
<-PCB_PN=025-0042-13>
```

The `Model` field can be: `SL232|RL3A|HMX3|IR3`

The `UUID` field value contains the Global Connect unique MAC address.

Global Connect also provides local name resolution using the multicast DNS mechanism which also uses the UDP protocol. (See RFC 6762 for additional details, https://tools.ietf.org/html/rfc6762.) This mechanism makes the Global Connect discoverable using a network name of the format <networkName>, where <networkName> is the value assigned in the Name field of the Global Connect network settings.

4. API CONNECTION

All Global Connect TCP API requests and responses (with the exception of serial data as noted below) are communicated through TCP socket connection on port 4998. Up to 8 simultaneous TCP API client connections are supported. All TCP API requests are sent as a single line ASCII text string terminated with a carriage return. The requests follow a consistent format, which requires a command, and typically one or more associated parameters specifying module and connector/port. Often additional comma delimited parameters for command-specific options are required. See the Section 5 API Requests for detailed explanation.

Each available Global Connect module has its own associated TCP API commands. See specific details in the following sections for each Global Connect module types.

Note that in the unique case of serial modules, serial data is sent and received on TCP port 4999. In this case, data is communicated directly between the network TCP connection and Flex Link Serial cable connector in a TCP-to-serial bridge mode, and is not interpreted or altered by the Global Connect in any way. See Section 5.3 for additional details.

5. API REQUESTS

The structure of all TCP API requests for Global Connect and Global Connect (GC) modules are described in following section and subsections.

In general, all TCP API requests adhere to the following format:

```
<command>,<module>:<port>,<parameter1>,<parameter2>,…,<parameterN>↵
```

In the above line, the following details are important to note:

- Each parameter is represented by a name enclosed in brackets.

- Each parameter is separated by commas.
- Various commands require different number of parameters.
- The request must be a single line ending with a carriage return, represented here by "↵".
- `<command>` is always a fixed text command string.
- `<module>:<port>` is sometimes a fixed value.
- `<parameterX>` always represents a user specified text value and must be replaced with a value as specified for the specific command. When multiple choices are available for the parameter they will be shown delimited by a vertical bar separator "|" character.

Note: Commands and parameters are case sensitive.

Example: The get_NET command is used to query network settings:

```
get_NET,<module>:<port>↵
```

where:
`<module>:<port> = 0:1` (the Global Connect host network module address is 0:1)

So, the literal TCP API request sent to the Global Connect is:

```
get_NET,0:1↵
```

Note: get_NET command is fully documented in Section 5.1 below.

Errors

An ERR response is returned when a TCP API request is invalid.

Example invalid request:

```
setstate,1:1,↵
```
(the <state> parameter is missing - see Section 5.4)

Response:

```
ERR [error code]↵
```

Note: Error Code definitions are provided in the Error Codes table in Section 6.

5.1 GENERAL COMMANDS

getdevices

Query current Global Connect modules. The response is multi-line, and lists each module with its address and type, including the Host (0,0), and configured Global Connect module (1,1). A complete response ends with an `endlistdevices` line.

Request:

```
getdevices↲           (query for modules and capabilities)
```

Response:

```
device,0,0 <Host type>↲
device,1,1 <Module type>↲
endlistdevices↲

where:
<Host type>         = MODULE
<Module type>       = IR|SERIAL|RELAY_6_3A|SWITCH_HDMI_3:1
```

Following are several example responses:

```
device,0,0 MODULE↲  (Infrared Module response)
device,1,1 IR↲
endlistdevices↲


device,0,0 MODULE↲  (Serial Module response)
device,1,1 SERIAL↲
endlistdevices↲
```

getversion

Query Global Connect firmware version.

Request:

```
getversion↲
```

Response:

```
<textversionstring>↲

where:
<textversionstring>       = firmware version part number
```

Example response from Global Connect:

```
710-4003-00
```

get_NET

Query current network settings. The response is a single comma delimited line.

Request:

```
get_NET,0:1↵
```

Response:

```
NET,0:1,<configlock>,<ipsetting>,<ipaddress>,<subnet>,<gateway>↵
```

where:

<configlock>	= LOCKED|UNLOCKED
<ipsetting>	= DHCP|STATIC
<ipaddress>	= IP address
<subnet>	= subnet mask
<gateway>	= network gateway

5.2 IR (INFRARED)

Global Connect, when used with IR emitters or IR blaster cables, can output standard IR protocol signaling and control a wide range of IR controlled devices. TCP API commands for IR functionality are detailed in the following subsections.

5.2.1 IR COMMANDS

set_IR

This command allows configuration of the IR Ports to match the selected IR cable.

Note: Only IR port 3 supports IR_BLASTER.

Request:

```
set_IR,<module>:<port>,<mode>↵
```

where:

<module>	= 1
<port>	= 1|…|3
<mode>	= IR|IR_BLASTER

Response:

```
IR,<module>:<port>,<mode>↵
```

Example request and response:

```
set_IR,1:1,IR↵            Configures IR port 1 for a Global IR Emitter cable.

IR,1:1,IR↵


set_IR,1:3,IR_BLASTER↵  Configures IR port 3 for an IR blaster cable.

IR,1:3,IR_BLASTER↵
```

get_IR

Retrieve the current mode setting for the IR module.

Request:

```
get_IR,<module:port>↵

where:
<module>      = 1
<port>        = 1|…|3
```

Response:

```
IR,<module:port>,<mode>↵

where:
<mode>        = IR|IR_BLASTER
```

5.2.2 IR STRUCTURE

An IR transmission is created by sending an IR timing pattern to Global Connect devices. The pattern is a series of <on> and <off> states modulated with a carrier frequency (ƒ) which is present only during the <on> state. A carrier frequency is typically between 35 to 45 KHz, with some equipment manufacturers using as high as 500 KHz. The length of time for an <on> or <off> state is calculated in units of the carrier frequency period. For example, an <off> value of 24 modulated at 40 KHz produces an <off> state of 600µS, as calculated below.

A period is 1 /ƒ or 1/40000 or .000025 seconds or 25µS,
and a value of 24 periods is 600µS

Figure 5.2.1 illustrates an IR timing pattern that would be created for the value shown below. IR timing patterns typically have a long final <off> value (or rest state) to ensure the next IR command is interpreted as a separate IR transmission.

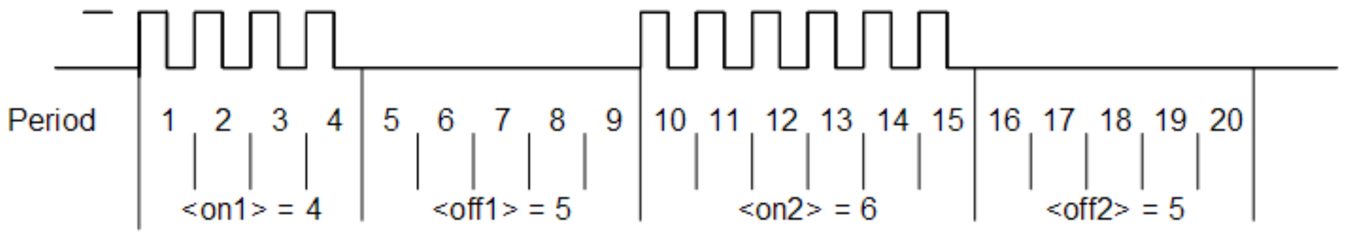

Figure 5.2.1

5.2.3 SENDING IR

Controling IR devices is accomplished through use of the `sendir` command. IR commands in some cases may take several hundred milliseconds to complete, so Global Connect provides a `completeir` acknowledgment to indicate when it is ready to accept the next IR command.

sendir

Request:

```
sendir,<module:port>,<ID>,<freq>,<repeat>,<offset>,<on1>,<off1>,<on2>,<off2>,…,<onN>,<offN>↵
```

where:

`<module:port>`	= [1] address of the desired IR channel (see below)
`<ID>`	= `0|1|2|…|65535` [2] (for the completeir response, see below)
`<freq>`	= `15000|…|500000` (in hertz)
`<repeat>`	= `1|2|…|20` [3] (the IR command is sent <repeat> times)
`<offset>`	= `1|3|5|…|383` [4] (used if <repeat> is greater than 1, see below)
`<on1>`	= `4|5|…|50000` [5] (number of pulses)
`<off1>`	= `4|5|…|50000` [5] (absence of pulse periods of the carrier frequency)
N	= the count of `<on>,<off>` pairs, which must be less than 260 (total of 520 numbers)

[1] Global Connect IR3 provides three (3) discrete channels of IR output. In these cases, `sendir` requests to any of the 3 IR channels must use the `port` component of the `<module:port>` parameter to address each of the 3 IR channels. Thus, for Global Connect, each of the 3 IR channels is addressed as follows (not all `sendir` parameters are shown):

```
sendir,1:1,…
sendir,1:2,…
sendir,1:3,…
```

[2] `<ID>` is a number provided by the `sendir` request, which is later included in the `completeir` response to indicate successful completion of the requested IR transmission.

[3] `<repeat>` is the number of times an IR transmission is sent, if not aborted via a `stopir`, or subsequent IR command (see section 5.4). Values above 20 are accepted, but the maximum number of transmitted repeats is capped at 20. In all cases, the preamble is only sent once (see `<offset>` below).

[4]An `<offset>` applies when the `<repeat>` is greater than 1. For IR commands that have preambles, an `<offset>` is employed to avoid repeating the preamble during repeated IR timing patterns. The `<offset>` value indicates the location within the timing pattern to start repeating the IR command as indicated below. The `<offset>` will always be an odd value since a timing pattern begins with an `<on>` state and must end with an `<off>` state.

`<offset>` odd value	Repeat start locations	
1	`<on1>`	no preamble
3	`<on2>`	
….	….	
n-1	<on((n/2)-1)>	

[5] Since IR transmissions end in an `<off>` condition, there must be an equal number of `<on>` and `<off>` states. Also, every `<on>` and `<off>` state must meet an 80µS minimum time requirement for the IR codes to be sent properly.

Example: With a carrier frequency of 60 KHz, the minimum value for `<on>` and `<off>` states is calculated below.

$$\texttt{<off>}_{min} = \texttt{<on>}_{min} \geq 80\mu S * f = 80\mu S * 60\ KHz = 4.8$$

For precise replication of an IR code at 60 KHz, all `<on>` and `<off>` values in the timing pattern must be 5 or higher.

All of the conditions above must be met for a valid `sendir` request. When a variable is missing or outside the accepted range, an error will be returned. For example, the sample `sendir` commands below will result in an error response.

Request:
```
sendir,10:3,3456,23400,1,1,24,48,24,960↵
```
(invalid module number for Global Connect)

Response:
```
ERR 003↵
```

Request:
```
sendir,1:1,23333,40000,2,3,24,48,24,48,960↵
```
(not an equal number of <on> and <off> parameters)

Response:
```
ERR IR006↵
```

Request:
```
sendir,1:1,0,40000,2,2,24,48,24,960↵
```
(`<offset>` is an even number)

Response:
```
ERR IR004↵
```

GLOBAL CACHÉ COMPRESSED IR FORMAT

The compressed IR format interprets the first 15 unique `<on>,<off>` pairs and assigns a single uppercase letter (A,B,C, and on) to each of those unique pairs. The first time a unique pair occurs within an IR command it is assigned a single ASCII character. Any subsequent occurrences of that same pair are then represented with that same ASCII character instead of the `<on>,<off>` representation. This allows the `sendir` request string to be significantly shorter, which can be advantageous in the case of a long IR code.

Example:

```
sendir,1:1,2445,40000,1,1,4,5,4,5,8,9,4,5,8,9,8,9↵
```

The unique pairs are underlined in the example request above. By assigning A to 4,5 and B to 8,9, the request can be rewritten as follows:

```
sendir,1:1,2445,40000,1,1,4,5A8,9ABB↵
```

Both commands are syntactically correct and are accepted by Global Connect. Both will transmit an identical IR code.

completeir

After the Global Connect successfully processes and transmits an IR command, it responds by sending the `completeir` response to the requester. The `completeir` response can be associated with the originating `sendir` command by way of the <ID> parameter. When utilized, the `<ID>` can provide a unique identifier to determine which IR transmissions have completed.

Sent from Global Connect in response to successful `sendir`:

```
completeir,<module>:<port>,<ID>↵
```

where:

`<module>`	= 1	
`<port>`	= 1|…|3	
`<ID>`	= 0|1|2|…|65535	(value is generated by the sender of the `sendir` request)

Examples:

Request:

```
sendir,1:1,2445,40000,1,1,4,5,6,5↵
```

Response:

```
completeir,1:1,2445↵
```

The following examples demonstrates two commands to send the same simple IR timing pattern of 24,12,24,960, repeated four times, and with a preamble of 34,48:

```
sendir,1:1,4444,34500,1,1,34,48,24,12,24,960,24,12,24,960,24,12,24,960,24,12,24,960↵
sendir,1:1,34,34500,4,3,34,48,24,12,24,960↵
```

Responses to the above requests:

```
completeir,1:1,4444↵
completeir,1:1,34↵
```

The same IR command is repeated four (4) times by either `sendir` request, but since the `<ID>` is different, a different `completeir` response is returned. The `<ID>` also allows associating each request to its response. Note that the second IR request is the recommended format, since it is shorter and also allows the command to be halted in between repeats, if necessary.

stopir

This request will stop an active IR transmission. Any remaining <repeat> cycles will be aborted. A `stopir` command sent to a connector not configured as an IR output will return an error message. An IR transmission halted with the `stopir` command will return a `stopir` response. Furthermore, if an IR command is halted before its completion by another connection, the originating IR connection and the connection sending `stopir` will both receive a `stopir` response. If stopped, the originating connection will not receive a `completeir` response.

Request:

```
stopir,<module>:<port>↵
```

Response:

```
stopir,<module>:<port>↵
```

where:

```
<module>     = 1
<port>       = 1|…|3
```

A `stopir` command always returns a `stopir` response, regardless of whether the port is idle, or an IR active transmission is halted. A `stopir` response means only that the `stopir` command was successfully sent to Global Connect and any transmission has been halted.

busyIR

A `busyIR` response occurs when an IR command is received for a port which is actively transmitting an IR code. This might occur, for example, if multiple IP connections are present (such as from multiple network users). In this case, the `sendir` command that receives the `busyIR` response will have its IR code transmitted.

Response to an attempt to interrupt IR transmission by another IP connection or socket:

```
busyIR,1:1,<ID>↵
```

where:

`<ID>`	= |0|1|2|…|65535|	(ID is specified in `sendir` command)

Note: The `busyIR` response is sent to the originator of the failed IR command. A `stopir` command will only return a `stopir` response. A `stopir` command will never return a `busyIR` response.

5.2.4 SMOOTH CONTINUOUS IR COMMANDS

A general discussion is necessary to better understand how smooth continuous IR commands are executed by Global Connect. This feature is utilized for smooth volume control or repeating an operation without the appearance of choppy actions. The approach of sending an IR command with large repeat counts and stopping it upon request will work, but can lead to undesirable incidents. Consider an IR command with a large repeat count used to increase volume in a smooth fashion. The IR command continues repeating while volume continuously increases. But if the controlling client/application unexpectedly disconnects, the volume could continually increase (possibly damaging equipment) until the client reconnects and issues a `stopir` command.

Global Connect's solution is to limit the repeat count. To create smooth IR operation, Global Connect resets the IR repeat count each time the identical IR command (from the same IP connection) is resent. This method will not interrupt and restart the IR command, but reset the IR repeat count back to the original value.

Example: If the IR repeat count is set to 5, and the IR command has transmitted 3 times, receipt of the same command causes the repeat count to be reset back to 5. This process can continue indefinitely while a volume button is held down to create smooth operation. However, at no time can the command repeat more than 5 times after the button is released or an IP connection is inadvertently lost, preventing a potentially serious issue.

By selecting an appropriate <repeat> value, the need for a `stopir` command is eliminated. In this example the volume continues to increase smoothly by retransmitting repeated IR commands due to the volume button being pressed. As long as the next repeated IR command is received before the previous command finishes, smooth operation is realized. By choosing a low repeat value, the volume increase will stop when the volume button is released. Also, proper IR operations happen even with unintended network delays due to traffic or WiFi connectivity. In this unlikely event, only small hesitations will be experienced during IR operation.

If the identical command is not received before the original command is finished, the command will be registered as a brand new command, and is sent as such. The command in question will operate functionally the same, but delays between commands may be evident when used in this way. Increasing the <repeat> value will likely eliminate these discrepancies.

5.3 SERIAL

The SL232 serial module allows for bi-directional serial communications over a RS232 interface. Serial communication parameters are configured using Global Connect embedded webpages or by direct API commands.

To send and receive serial data, a TCP socket connection must be established on port 4999. Data is communicated directly between the TCP connection and Flex Link Serial cable connector in a TCP-to-serial bridge mode. The data is passed directly and is not interpreted or altered in any way by the Global Connect.

If Global Connect serial communication parameters are not configured correctly to match the connected device, buffer overflows (indicating data loss) or parity errors may occur.

5.3.1 MULTIPLE SERIAL CONNECTIONS

The Global Connect supports up to eight (8) simultaneous, bidirectional TCP-to-serial connections on port 4999.

SENDING DATA TO A SERIAL DEVICE

In order to support multiple TCP clients simultaneously sending data to a single serial-connected device, Global Connect handles received TCP data on a packet basis for efficient transmission to a serial-connected device. When a data packet is received by the Global Connect it is transmitted completely to the serial-connected device before any subsequent packets (received from the same or different socket) are sent. This is important when considering the case where TCP client A sends a single command in two (2) separate packets while a different TCP client B simultaneously sends a packet. In this case, the packet from TCP client B may be serviced in between the 2 separate packets received from TCP client A, possibly interrupting the data or command from TCP client A. It is very important for TCP clients sending data to the Flex to send a complete serial data sequence or command(s) within a single packet. This ensures complete data sequences or commands are received properly by the serially-connected device.

RECEIVING DATA FROM A SERIAL DEVICE

Serial data received by the Global Connect from a serial-connected device is also handled as packets. But unlike TCP-to-Serial data flow, wherein packets are based on TCP protocol, serially received data is packetized according to timing and size. The initial character of received serial data initiates a receive packet. The completion of the packet occurs when either the Global Connect receive buffer reaches a certain capacity or no characters are received for a calculated time delay (based on current baud rate). The completed serial data packet is then immediately transmitted to all active/open TCP socket connections. This allows all connected TCP clients to synchronously receive all data sent from the serial device.

5.3.2 SERIAL COMMANDS

`set_SERIAL`

Configure Global Connect serial communication settings.

Requests:

```
set_SERIAL,<module>:<port>↵

set_SERIAL,<module>:<port>,<baudrate>,<flowcontrol >,<parity>,[stopbits]↵
```

Response:

```
SERIAL,1:1,<baudrate>,<flowcontrol>,<parity>,<stopbits>↵
```

where:

`<module>:<port>`	= `1:1`
`<baudrate>`	= `300|…|115200`
`<flowcontrol >`	= `FLOW_HARDWARE|FLOW_NONE`
`<parity>`	= `PARITY_NO|PARITY_ODD|PARITY_EVEN`
`[stopbits]`	= `STOPBITS_1|STOPBITS_2` (optional parameter)

Example request and response:

```
set_SERIAL,1:1,38400,FLOW_NONE,PARITY_NO↵

SERIAL,1:1,115200,FLOW_NONE,PARITY_NO,STOPBITS_1↵
```

get_SERIAL

Query current serial communications settings.

Request:

```
get_SERIAL,<module>:<port>↵
```

Response:

```
SERIAL,1:1,<baudrate>,<flowcontrol >,<parity>,<stopbits>↵
```

where:

`<module>:<port>`	= `1:1`
`<baudrate>`	= `300|…|115200`
`<flowcontrol >`	= `FLOW_HARDWARE|FLOW_NONE`
`<parity>`	= `PARITY_NO|PARITY_ODD|PARITY_EVEN`
`<stopbits>`	= `STOPBITS_1|STOPBITS_2`

Example request and response:

```
get_SERIAL,1:1

SERIAL,1:1,115200,FLOW_NONE,PARITY_NO,STOPBITS_1↵
```

5.4 RELAYS

The GCRL3A relay module provides dry contact closure relay outputs capable of switching a variety of devices. See the Global Connect data sheet for detailed hardware specification.

This section describes the TCP API commands used for configuration and control of the relay outputs.

5.4.1 RELAY COMMANDS

get_RELAY

Query the type of a specified relay port.

Request:

```
get_RELAY,<module>:<port>↵

where:
<module>      = 1
<port>        = 1|…|6
```

Response:

```
RELAY,<module>:<port>,<type>↵

where:
<type>        = SPST
```

Example request and response:

```
get_RELAY,1:4

RELAY,1:4,SPST↵
```

set_RELAY

Set the type of a specified logical relay port.

Note: This command is for backwards compatibility with the Flex Link Relay and Sensor Cable. It provides no functional purpose otherwise.

Request:

```
set_RELAY,<module>:<port>,<type>↵
```

where:

`<module>`	= 1
`<port>`	= 1|…|6
`<type>`(1)	SPST

Response:

```
RELAY,<module>:<port>,SPST↵
```

Example request and response:

`get_RELAY,1:3↵`	Query the current configuration for logical relay port 3.
`RELAY,1:3,SPST↵`	

getstate

Query the current state of a logical relay port.

Request:

```
getstate,<module>:<port>↵
```

where:

`<module>`	= 1 (1)
`<port>`	= 1|…|6

Response:

```
state,<module>:<port>,<state>↵
```

where:

`<state>`

0	= off	(open)
1	= on	(closed)

(1) A module value of 3 is also accepted for GC-100 backwards compatibility.

setstate

Set the state of a logical relay port.

Request:

```
setstate,<module>:<port>,<state>↵
```

where:

`<module>`	= 1 [1]		
`<port>`	= 1|…|6		
`<state>`			
	`0`	= off	(open)
	`1`	= on	(closed)

Response:

```
state,<module>:<port>,<state>↵
```

Note: Relay states are not preserved if the Global Connect module is unplugged or reinitialized or if Global Connect is reset and/or power cycled. Under any of those conditions, all relays will return to their default inactive (open) state.

[1] A module value of 3 is also accepted for GC-100 backwards compatibility.

6. HDMI SWITCHING

The GCHMX3 HDMI switching module supports a UHD 4K 60 fps 3:1 switcher. TCP commands are available for switching the HDMI switch.

setstate

Set the HDMI input state.

Request:

```
setstate,<module>:<port>,<state>↵
```

where:

`<module>`	= 1		
`<port>`	= 1|…|4		
`<state>`			
	`0`	= off	(input disabled)
	`1`	= on	(switched to output)

Note: Enabling an input will automatically disable any other enabled inputs.

Example request and response:

```
setstate,1:1,1↵    //Switch the input 1 to the output
state,1:1,1↵

setstate,1:3,0↵    //Disable input 3 if enabled
state,1:3,0↵
```

getstate

Get the HDMI input state.

Request:

```
setstate,<module>:<port>,<state>↵

where:
<module>     = 1
<port>       = 1|…|4
<state>
               0  = off   (input disabled)
               1  = on    (switched to output 1)
```

Example request and response:

```
getstate,1:2,1↵
state,1:2,1↵       //Input 2 is enabled
```

getactive

Returns the state of the connected HDMI devices. A true state value indicates that an HDMI cable is connected to the port and that the connected source/sink device is powered on.

Request:

```
getactive,<module>↵

where:
<module>     = 1
```

Response:

```
active,<module> ↵
OUT:
1,<state>
IN:
1,<state> ↵
2,<state> ↵
3,<state> ↵
endactive,1↵
```

```
where:
<module>     = 1
<port>       = true|false
```

Example request and response:

```
getactive,1↵

active,1↵
OUT: ↵
1,true ↵
IN: ↵
1,true ↵
2,false ↵
3,false ↵
endactive,1 ↵
```

7. ERROR CODES

The table below lists of error messages returned by Global Connect in response to various invalid TCP API requests. See section 5 for an example invalid request and resulting response format.

General Errors	Explanation
ERR 001	Invalid request. Command not found.
ERR 002	Bad request syntax used with a known command.
ERR 003	Invalid or missing module and/or connector address.
ERR 004	No carriage return found.
ERR 005	Command not supported by current port setting.
ERR 006	Settings are locked.

IR Errors	Explanation
ERR IR001	Invalid ID value.
ERR IR002	Invalid frequency.
ERR IR003	Invalid repeat.
ERR IR004	Invalid offset.
ERR IR005	Invalid IR pulse value.
ERR IR006	Odd amount of IR pulse values (must be even).
ERR IR007	Maximum pulse pair limit exceeded.
Serial Errors	**Explanation**
ERR SL001	Invalid baud rate.
ERR SL002	Invalid flow control setting.
ERR SL003	Invalid parity setting.
ERR SL004	Invalid stop bits setting.
Relay	**Explanation**
ERR TBD	Invalid logical relay type.
ERR TBD	Invalid logical relay state.
ERR TBD	Unsupported operation.

ERR TBD	Logical relay disabled or not available.
ERR TBD	Invalid sensor notify port value.
ERR TBD	Invalid sensor notify timer value.
HDMI	Explanation
TBD	TBD