CONTROLLING THE FUTURE OF LIGHTING

| | |
|---|---|
| Title: | Gateway Interface Volume 1 - Standard |
| Version: | 2.0.3 |
| Author: | Jeff Heine |
| Date: | 16 August 2024 |

# History

| Date | Ver | Initials | Changes |
|---|---|---|---|
| 24-Mar-19 | 2.0.0 | Jph | Initial draft |
| 6-Sep-19 | 2.0.1 | Jph | Clarify text on format of rgb and wrgb |
| 14-Dec-21 | 2.0.2 | Jph | Add extra values to the Enumeration tables<br>Modification of colour and tuneable white messages for Gateway Ver 2.01 |
| 28-Feb-22 | 2.0.3 | Jph | Update plate sections to include rotary plate<br>Add Channel Error status code enum for event processing<br>Add EVO-INT232 connections.<br>Messages for Gateway Ver 2.02 |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Table of Contents

# 1 Introduction

## 1.1 Purpose and scope

This document contains details for using the eDIN+ GATEWAY interface. This document is part of a set of Volumes that cover the full GATEWAY interface.

| | |
|---|---|
| Volume 1 – Standard | How to access the GATEWAY and commands and queries that all users can use. |
| Volume 2 – Advanced | Advanced commands and queries that require more advanced knowledge of the eDIN+ system and are only needed to access these advanced features. Some are restricted to users with Administration access. |
| Volume 3 – Developer | These are a set of commands and queries that allow some aspects of the eDIN+ system to be tightly integrated with third party products. They are restricted to users with Administrator rights, and require specialist knowledge of the eDIN+ system. |

## 1.2 Terminology

| Phrase | Description |
|---|---|
| API | Application Programming Interface. A set of protocols that define how to access a system. |
| Controller | A processor that controls a lighting system. The eDIN+ system has 2 types of controller: normal NPU and Standalone Mode. |
| DALI | Digital Addressable Lighting Interface. An industry standard physical interface/network for control of lighting equipment. It allows bi-directional data exchange, allowing its use with sensors, switches and emergency lighting as well as dimmable luminaries. |

## 1.3 References

[1] BS EN 62386-102:2014 Digital addressable lighting interface. Part 102: General Requirements – Control gear.
[2] BS EN 62386-202:2009 Digital addressable lighting interface. Part 202: Particular requirements for control gear — Self-contained emergency lighting (device type 1).

## 1.4 Changes from Gateway Version 1.x

The core of the GATEWAY messages and protocols have not changed between v1.x and v2.x. However there are some significant additions and changes in some areas that are not backwards compatible. These are listed below.

- `?SCN` query has had an extra `<level>` parameter added
- The `!SCN` event is replaced with `!SCNSTATE` event
- All DMX support has changed and extended to support RGB colour and Tuneable White on both `DMX` zones and DBST `CHAN`nels

- The response to all channel status queries, e.g. `?CHAN`, `?DALI`, `?DMX`, `?INP` have changed from a single reply message to multiple messages reporting different items such as health, level, colour etc.
- Relay support has been improved. `$CHANSTATE` and `!CHANSTATE` messages have been replaced with `$CHANFADE`, `$CHANPULSE` and `!CHANPULSE` messages.
- Channel discovery during scene setting has changed significantly to accommodate RGB & Tuneable White colour support.
- The behaviour of User login has changed significantly. Non-encoded passwords are now the norm.
- The System Health API and Developer DALI Bus Test & Repair API went through a number of iteration in v1.x These interface is now stable.
- There is a new developer `XDALI` API that replaces the trial `EMTEST` API

# 2 Introduction to the eDIN+ system and GATEWAY interface

The purpose of the GATEWAY interface is to allow external systems to monitor the status and control various aspects of the eDIN+ lighting system; it is to provide real time (sometimes called runtime) access to the system. This chapter gives an introduction to the interface to orient you and discusses various aspects of it.

There are different techniques and ways to use the GATEWAY interface. These are discussed in this chapter.

The commands, queries and events (often called the API) of the GATEWAY interface are grouped in to a number of categories and documentation arranged in to Volumes. This chapter explains these and how to find your way around.

Before you use the GATEWAY interface on a system you need to configure the hardware and other lighting features you are going to use, but this cannot be done via the GATEWAY. This chapter touches on this aspect.

## 2.1 Overview

The GATEWAY interface allows external systems to monitor the status and control various aspects of the eDIN+ lighting system. The interface is to provide real time (sometimes called runtime) control and monitoring of the system. With a few exceptions, you cannot inspect or set the configuration of the eDIN+ system through the interface.

The GATEWAY interface is character/text based and can be used over

- an RS232 connection,
- a raw TCP/IP session over Ethernet, or
- a set of HTTP requests over Ethernet.

Later chapters discuss these connections further and how to troubleshoot connections.

The interface provides

- Commands to control the eDIN+ system
- Queries to request the current state of the eDIN+ system
- Events sent by the eDIN+ system notifying you when there is a change to its state.

The syntax of the command, query and event messages make using the interface relatively straightforward. As examples,

- Everything is in normal (printable) text using Unicode UTF-8 formatting.
- It is simple to identify commands, queries and events:
  - All commands begin with a $ and end with a ; character
  - All queries begin with a ? and end with a ; character
  - All events and responses to queries begin with a ! and end with a ; character
- All events and responses to queries are ‘self-documenting’ – they contain all the information you need to process them. You do not need to know what

query made them, what the order of multiple queries were or the result of previous queries or events.
- Every command and query sent is acknowledged by the GATEWAY interface, so that you know it has been successfully sent.

The syntax is further described in a later chapter.

There are a number of ways you can interact with an eDIN lighting system using these messages:
- Control-only: the simplest way but only if you do not want feedback about the state of the eDIN+ system. Simply send commands as required.
- Control-with-polling-queries: the most applicable and straightforward technique. Send queries as required to request the current the state of the eDIN+ system. Send commands as you required.
- Control-with-asynchronous-event-driven-monitoring: the most demanding technique. Your event handler continuously processes events sent by the eDIN system. Commands are sent by you as required.

These techniques are discussed further in later sections of chapter.

The commands, queries and events are grouped in to API categories for convenience:
- Interface API – this is used to control the information sent through the connections. These are used to tailor the connection to make it easier for you to use the GATEWAY.
- Scene and System API – this talks to the in-built controller of the eDIN+ system (if it exists), and allows access to things such as scenes and areas.
- Channel API – this accesses the actual hardware. It allows you to monitor and control individual lighting channels, (although controlling channels may fight any in-built controller which is also controlling these channels.)

The documentation for the GATEWAY interface is organised in to a number of volumes.
- Volume 1 is the place to start and may be all that you need to provide control and monitoring of a specific eDIN+ system.
- Volume 2 is if you need to access advanced features of the eDIN+ system.
- Volume 3 is primarily for software developers that are building generic or highly integrated Apps and/or systems, and need advanced features of the GATEWAY interface to achieve this.

All eDIN+ systems need to be configured before they can be controlled and monitored via this GATEWAY interface. Although configuring an eDIN+ system is beyond the scope of this document, this is touched upon briefly at the end of this chapter.

Detailed information on the API are given later in this document and in the other volumes.

## 2.2 Before you begin

There are 2 aspects that you need to consider when deciding to use the GATEWAY interface:

1) How much control are you going to do, and how much are you going to leave to the in-built eDIN+ system controller?

   Generally, it is better to let the in-built controller control as much as possible for a number of reasons including:

   i) The in-built controller is always present. You only need to be in contact with the eDIN+ system when you or your end-user wants to know the state of the system or to do immediate control. This is the best option if you are e.g. writing an App for a mobile phone.
   ii) Alternatively, even if your controller is always physically present, it may occasionally lose connection with the eDIN+ system. With the in-built controller, it is always providing continuous control and is able to turn lights on & off when wall plate buttons are pressed even when your connection is lost.
   iii) The eDIN+ product is focused on providing sophisticated lighting control in a simple manner, and has extensive years for lighting control know-how built in. It may be easier to achieve the desired control via the eDIN controller than having to do all the hard work yourself.

   However, it is possible to set up a minimal configuration and perform all of the control yourself. In that way you get the exact control you want that may be different from the eDIN+ control.

   It is also possible to set up a 'back-up' configuration in the in-built controller, but suspend the in-built controller so that it only takes over if it detects your control is not present. This is an advanced topic and further details of this back-up technique is in Volume 3.

2) How do you want to interact with the GATEWAY interface?
   This primarily comes down to the requirements for and the capabilities of your control system. There are 3 techniques. They control the system in the same way but differ in how to monitor and obtain the live or current state of the eDIN lighting system.

   i) Control-only (no feedback)
      This is the simplest technique, but only suitable when you do not need to know the current or live state of the eDIN+ system. Simply send commands without any querying of the system.

      This technique is suitable for all 3 types of connection: RS232, HTTP and Raw IP – simply ignore any incoming data or response.

   ii) Control with polled feedback
      This is the most straightforward and generally applicable technique for determining the state of the eDIN+ system. It is easier programming/scripting to code and can be performed from almost all third party systems. You do not need a continuous connection.

Commands are sent to the system as above. Feedback about the state of the eDIN+ system is only as a response to a query that you send when you need to know it.

Only receiving and processing information from the eDIN+ system when you want it can be an advantage, but if you want to continuously monitor the system you have to regularly and periodically query or 'poll' the system.

This technique is particularly suited to using HTTP connections, but can also be used on RS232 and Raw IP connections.

iii) Control with asynchronous event-driven feedback
This allows the most integration between your system and the eDIN system, but your system must be able to be handle event-driven programming. Also this style of coding is harder to debug when problems arise.

With this technique, the state of the eDIN+ system is determined not with repeated queries but by having a continuous connection open and handling 'events' that are sent by the eDIN+ system whenever things happen and its state changes. Commands (and queries) are sent from one process; a second process receives and processes all responses sent by the eDIN+, including events and responses to queries.

This technique allows your system to respond instantly to changes in the eDIN+ system, and only respond and react when changes occur. However, as it is responds only to changes in state there has to be additional code to (synchronise to) determine the initial state of the eDIN+ system and maintain synchronisation when connections are lost and re-established.

You can only use this technique with a continuous connection to the eDIN+ system. You must use either RS232 or Raw IP connections. You cannot use HTTP connections with this technique.

Volume 1 documents the commands to control the system. The events sent by the eDIN+ system are documented in Volume 2.

## 2.3 Finding your way around the GATEWAY API

The GATEWAY interface and group in to APIs for convenience. All consist of:

- Commands and query messages sent by you to control and monitor the state of the eDIN+ system, and
- Response messages to your queries and event messages that notify you when there is a change of state and are sent by the eDIN+ system.

These commands, query and event messages are grouped in to the following API categories for convenience:

- Interface API
  This is used to control the information sent through the connections. These are used to tailor the connection to make it easier for you to use the GATEWAY.

- Scene and System API
  This talks to the in-built controller of the eDIN+ system (if it exists), and allows access to things such as scenes and areas.

- Channel API
  This accesses the actual hardware. It allows you to monitor and control individual lighting channels, (although controlling channels may fight any in-built controller which is also controlling these channels.)

Again for your convenience, each API has aspects that are either standard or advanced, and these levels are documented separately.

- The standard messages access the main features of the eDIN+ system and you should have knowledge of these. All eDIN+ systems (with the odd exception) have these standard features. The majority, if not all, of your control and monitoring will be with these standard level messages.

  Volume 1 documents these standard messages and is the place to start.

- The eDIN+ system and the GATEWAY interface both have advanced features that go beyond the normal control and monitoring. You may never find it necessary or appropriate to use these advanced features, which is why they are kept separate from the standard set of messages. However they can be applied to most eDIN+ systems.

  Volume 2 is if you need to access advanced features of the eDIN+ system.

- There are some eDIN+ systems that work in very specific or niche markets, and have very specific requirements. This includes working in partnership with developers to produce highly integrated Apps and/or systems, and need advanced features of the GATEWAY interface to achieve this.

  Volume 3 documents these messages for completeness. If you think you need to use any of these features, contact Mode to discuss your requirements and for further support with them.

There are a few things to bear in mind when picking a command or query.

- You can freely use Interface API as you need to tailor your connection. The default settings mean you may never need to use this API but there are a few situations when you might including:
  i) You need to debug or troubleshoot a connection – see the chapter on this for further details.
  ii) (Advanced) If you want to use events on a connection, you need to turn them on. They are off by default.
  iii) (Advanced) If you need to log on as a specific user if your configuration has set up user accounts.

- You should in general use the System API for your control & monitoring, and not the Channel API:
  - *Only* and *always* use the System API *commands* if there *is* an active in-built controller.
    - if you use Channel API *commands* in a system with an active in-built controller, you will be overriding the in-built control and confusion may result.
  - You *must* use the Channel API if there is no active in-built controller. The System API will do nothing.

- You can safely use System API *and* Channel API *queries and events* to monitor an eDIN+ system:
  - use the System API queries and/or events to monitor System features such as lighting scenes
  - use the Channel API queries and/or events to monitor low level Channel states

There is one more things to consider before you get underway – configuring the eDIN+ system.

## 2.4 Knowing your eDIN+ system configuration

To control a lighting system you have to know what there is to control. For eDIN+ systems this means it must have a configuration.

In general the GATEWAY interface expects you to already know what you have in your system. This often fits well with building custom control front-ends designed around a specific system.

There is some but limited discovery queries in the GATEWAY interface. These allow you to adapt some aspects of your control to work across many systems or to changes in your system over time. These are described in Volume 1 section called System Discovery.

It is also possible to obtain fuller information about what is in the configuration in text files following the Comma-Separated-Variable format. This can be done manually or via a different 'Info' NPU web service. Further details of this topic are in Volume 3.

## 2.5 Configuring the eDIN+ system

All eDIN+ systems need to be configured to some extent:

- All hardware modules and channels properties need to be set up correctly. Depending on modules, the channel type of configurable IO channels need to be set. Most channels have a number of properties that control how they operate, such as cup & cap values.
- Controllers need to have scenes defined, possibly grouped in to areas.
- Controllers can have rules defined that link input buttons and channels to scene recall actions.

The eDIN+ lighting system cannot be configured through the GATEWAY interface. One of the reasons for this is to keep the interface small and simple to use.

Configuration happens once, with the occasional adjustment. Monitoring and control happen continuously.

The simplest way to configure the eDIN+ system is by creating a NPU configuration. This can be done on a live system via the NPU web pages, or with the eDIN+ offline editor that creates a file that can be uploaded to the NPU.

The configuration defines what hardware modules the system has and sets the properties of the module and its associated lighting and sensor channels. You can also define other features such as lighting scenes and areas to use, and also rules that connect input channel events such as button presses to actions such as recalling a particular scene. It is also possible to use more advanced eDIN+ controller features to provide more sophisticated automation.

The details of configuring the eDIN+ lighting system is beyond the scope of this document. Contact our customer support team or our web site for further information.

# 3 General Syntax and Operation

This chapter details the syntax and operation of the GATEWAY interface that is common to all APIs.

The syntax and operation is designed to be used either:

- manually connected to a terminal emulation app such as HyperTerm or PuTTY, or
- programmatically via user apps such as Savant or Roomie.

## 3.1 General Syntax

All messages are Unicode UTF-8 text strings and conform to the following general syntax.

| | |
|---|---|
| `$<msgId>[,<parameter>,<parameter>,…];` | for commands |
| `?<msgId>[,<parameter>,<parameter>,…];` | for queries |
| `!<msgId>[,<parameter>,<parameter>,…];` | for acknowledgements, replies and events |

- All messages start with either `$`, `?` or `!`.
- All messages end in a semicolon character `;`.
- There are no space characters in a message (except if spaces exist in the text parameters).
- The `<msgId>` is the name of the command, query or event.
- Each `<msgId>` may have 0 or more `<parameter>` (usually numeric decimal) values separated by the comma character `,`.
- Messages are case-insensitive (except the text parameters), i.e. any combination of upper and lower case letters are accepted.
- Any extra characters sent between the end `;` of one message and the start of the next are ignored. Thus it is permissible to add `<CR>` and `<LF>` characters *after* the `;` end of each message.

Commands and queries *sent to* the interface can vary in format:

- messages must start with a $ or ? and end with ;. However,
- they are case insensitive,
- numeric parameters can be variable length with or without leading zeros,
- all extra characters between messages (such as `<CR>` and `<LF>`) are tolerated and ignored.

Acknowledgements, replies and events *received from* the interface always:

- always start with ! and end with ;
- are in upper case,
- numeric parameters may have leading zeros,
- have variable length text parameters,
- have `<CR>` and `<LF>` appended after each message.

This makes parsing a line at a time possible, using simple uppercase comparisons.

## 3.2 Message acknowledgements

For every message sent to the interface, the interface always acknowledges the message with

- `!OK` message if it is a recognised command or query

  or

- `!BAD` message if it is an unrecognised command or query

The general format of an acknowledge message is:

```
      [<echochars>]!OK[,<msgId>,<parameter>,<parameter>,…];
```

or

```
      [<echochars>]!BAD;
```

Note: This acknowledgement only indicates that the syntax of the message is correct. **It does NOT indicate that the eDIN system has performed the command or query successfully**. – see Message Errors section for further details.

It is possible to configure the interface to format the generated acknowledge messages to contain more or less information:

- The characters of the message sent to the interface can be echoed back in the message acknowledge, except unrecognised characters are echoed back as the period character ‘.’
- The command or query msgId and parameters can be included in the `!OK` (long) acknowledgement message

The echo feature is to help troubleshoot `!BAD` messages, and is normally turned off. Use `$DBGECHO` command to turn this feature on & off. The feature is always off on a new connection and off by default for AJAX requests.

The long acknowledge message includes the details of what it is acknowledging and is normally turned on. It can be useful to the end-user for synchronisation, to ensure the acknowledge message relates to the correct message sent to the interface. It can also be used to confirm the gateway interface has seen the command/query and parameters that you have sent, and has not interpreted them differently. Use `$DBGACK` command to turn this feature on & off. The feature is always on for a new connection and on by default for AJAX requests.

## 3.3 Message errors

One unusual aspect of the GATEWAY interface is that it only partially detects and reports problems with a command or query – only those related to command syntax via the message acknowledgement mechanism as described previously. It does not report other problems with the message.

Specifically, the GATEWAY interface **does NOT check the parameter values** of the command or query, that these values makes sense or are appropriate for this specific eDIN+ system. **This type of error is only indicated by an *absence* of a response** (beyond the message acknowledgement).

For example, if you query a scene using a scene number that does not existing in the in-built controller, the message will be successfully acknowledged but no further response will be generated. Similarly if you recall a scene that does not exist, the scene cannot be recalled and events will not be generated.

The reason for this behaviour of the GATEWAY is that its protocols have been designed to be consistent across all types of eDIN+ systems, no matter what sort of controller they have or if a controller exists at all.

It is useful to remember the following:

- The `!OK` response to a command or query *only* indicates the message has been *received correctly*, **not that it has been performed successfully**.
- If you have already checked that your connection is good, but *the system is not responding* to your commands or queries, it is **likely your parameter values are wrong**. Check them carefully against the configuration currently loaded in to the NPU.

For further details on connecting and trouble shooting. See the specific chapter on this topic.

## 3.4 Gateway version numbers

The GATEWAY interface will evolve and change over time. The GATEWAY interface has an associated version number. Each change to the interface increases the version number, so that these changes can be tracked.

The version number is a decimal value, with a major number before the decimal point and an minor number after the decimal point. Changes that are minor and backwards compatible with earlier versions only increase the minor number. Major changes and changes that are not backwards compatible increase the major number.

To be resilient to changes

- if you receive (event or query response) unknown messages or parameters from the GATEWAY connection, quietly ignore or log them, rather than generating and reporting this as an error. They are likely to be newer messages from a later version of the GATEWAY.
- Note which major version number you are coding to. If the major version of your connection is not right, add this error to your log file, as there is a good chance your code will be incompatible with the connection and not work properly.
- Note which minor version number you are coding to. If the minor version of your connection is smaller, add this warning to your log file, as there may be some messages that the connection does not support.

# 4 Connecting to and Troubleshooting the Gateway Interface

You can connect to an eDIN lighting system using the Gateway Interface from any NPU on that system as long as the NPU has:

- Firmware version 2.0.0.0 or higher
- A complete configuration of the system loaded on to it

## 4.1 Connections

There are four ways to connect to an eDIN system. Currently, all must be through an NPU module.

1. Ethernet / raw TCP-IP connection with the NPU
2. The NPU local RS232 port
3. A remote RS232 connection into an EVO-INT232 module
4. HTTP / AJAX-like web service requests on the NPU

It is possible to open more than one connection (also called session) on each NPU simultaneously. Currently each NPU can handle 1 RS232 session plus 4 Raw IP sessions plus 2 EVO-INT232 sessions, but this may change over time. Note: HTTP requests open short lived sessions that only last for that request and each (short-lived) HTTP request is handled independently from the next. As such the only limit is how many instantaneous HTTP request (including from web browsers) the NPU can handle and is not normally an issue.

To check that the connection is OK, send the null command (`$OK;`). If the connection is OK, the system will respond `!OK;<CR><LF>`. See below on Troubleshooting for further details.

### 4.1.1 Raw TCP/IP

The NPU must be configured to accept Raw IP GATEWAY connections. This is done from the web page: Settings->Network services

- Ensure 'Enable gateway control' is ticked
- Adjust the 'Use port' value as required or leave as the default value (port 26)

Connections can now be made.

When a new connection is made the interface sends out 2 messages:

```
!GATRDY;<CR><LF>
!VERSION,<version>;<CR><LF>
```

The gateway interface *automatically* closes raw TCP/IP connections that are idle for 1 hour. The gateway interface is idle if it does not receive any commands or queries or has not sent any events.

**To keep a raw TCP/IP connection open, send the null command (`$OK;`) at least once an hour.**

### 4.1.2 RS232 port

The NPU must configure its RS232 port for Gateway operation. This is done on the web page Settings->RS232

- Ensure under ‘RS232 port service->Port used by’ that ‘Gateway Control’ is selected
- Adjust ‘RS232 port settings’ (e.g. Baudrate, Data bits …) as required

When the RS232 port is ready for Gateway commands it sends out 2 messages:

```
        !GATRDY;<CR><LF>
        !VERSION,<version>;<CR><LF>
```

This is does this when

i. the RS232 port is newly configured for Gateway Control, and
ii. the NPU powers up and is (has previously been) configured for Gateway Control

The RS232 serial connection is left open unless the serial port is configured to be used by another service.

### 4.1.3 EVO INT-232

The INT232 module must be connected to the system’s MBus and have a unique MBus address compatible with all other modules, but the INT232 module should NOT be entered in the system’s configuration file.

The system auto detects the module and sets it up for GATEWAY operation using the fixed RS232 settings: 9600 baud, 8 data bits, no parity, 1 stop bit. It is not possible to change these settings.

When the system detects the INT232 and is ready to receive Gateway commands it sends out 2 messages:

```
        !GATRDY;<CR><LF>
        !VERSION,<version>;<CR><LF>
```

This is does this when

i. every time the NPU powers up and detects the INT232 is present on the MBus

The connection is left open while the system is alive.

A system can currently handle a maximum of 2 x INT232 modules as remote Gateway connections.

### 4.1.4 HTTP

The NPU must have its Web Server enabled which is the default setting.

The http connection uses plain text AJAX-like HTTP POST requests to the URL `http://<npu_ip_address>/gateway?` (Note the terminating ? is required) using the `Content-Type: text/plain`.

One or more command and/or query are assembled in to a plain text request block of the AJAX POST operation. The responses to the commands and queries are returned in the plain text response data block.

Example in JQuery:

```
$.ajax({
      type: "POST",
      url: 'http://'+ipaddr+'/gateway?',
      contentType: "text/plain;",
      dataType: "text",
      data: "$scnRecall,1;",
    })
```

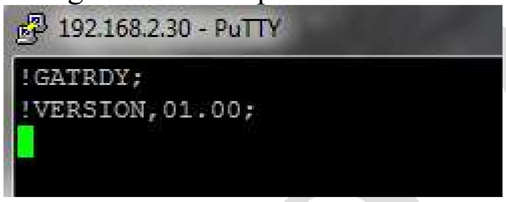
The http connection does not support long term sessions. Each http request/response stands by itself. As a consequence no event data is published on the http connections. Only commands and queries can be used.

## 4.2 Troubleshooting

### 4.2.1 Troubleshooting the Raw IP & RS232 connections

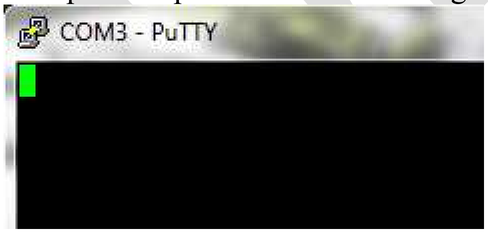
To check that you can connect to the NPU as expected, and that it is responding correctly, try connecting using a terminal emulation app such as HyperTerm or PuTTy. This app note will use PuTTy as an example.

For IP connections, open PuTTy and in the session configuration dialog type in the NPU's IP address and port number (default 26) as configured on the web page (see section above), and set the 'Connection type:' as 'Raw', then press 'Open'. A session dialog box should open



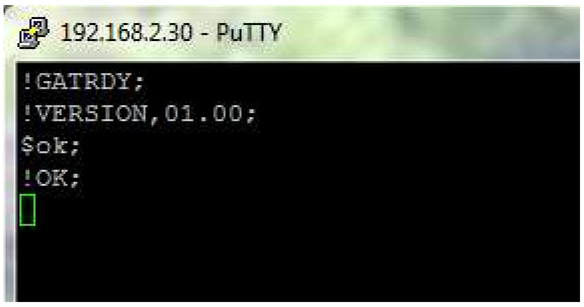
```
!GATRDY;
!VERSION,01.00;
```

For RS232 connections, open PuTTy and in the session configuration dialog set the 'Connection type:' as 'Serial', in the Connection->Serial dialog set up the COM port appropriately (default 9600,8,1,no parity, no flow control) as configured on the web page (see section above), and in the Termial dialog set 'Local Echo:' to 'Force On', then press 'Open'. A session dialog box should open



NOTE: the automatic `GATRDY` and `VERSION` messages do not occur unless the RS232 session is open when the NPU powers up.

To check that the connection responds to commands, type `$OK;<ENTER>`. You should see something like



```
!GATRDY;
!VERSION,01.00;
$ok;
!OK;
```

NOTE: some apps, like PuTTy, do not send any characters until `<CR><LF>` or `<ENTER>` is pressed, allowing you to edit the line before sending.

### 4.2.2 Troubleshooting the HTTP connection

If you are having problems using AJAX calls from your own application, the first thing to check is HTTP access to the NPU. Can you see/access the normal NPU web pages via a browser from the same computer that is making the AJAX call? If you cannot access even the web pages successfully then there is a bigger problem with the physical connection to the NPU. Correct this problem before continuing. If you can see the web pages then your problem is most likely with the application.

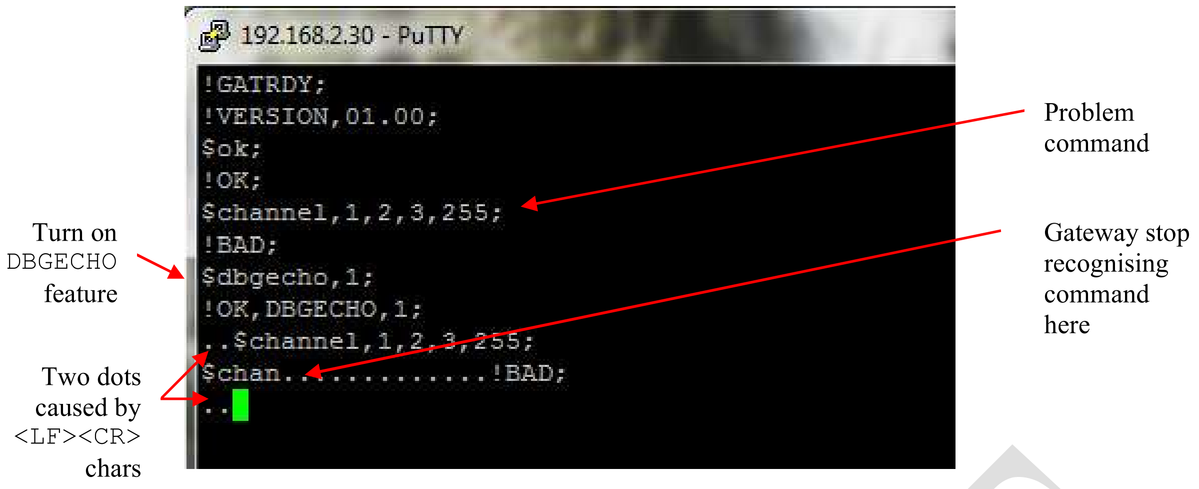
Within your application, here is a checklist to try:

- Is the URL being used is correct? Example: http://192.168.0.2/gateway?
  - Is the NPU IP address the same the one that works in the browser?
  - Have you included the terminating question mark?
- Are you setting the content-type to text/plain?
- Send the null command `$OK;` in the request data of the AJAX call.
  - Do you get any response data? A good connection will respond with `!OK;`

### 4.2.3 Troubleshooting a !BAD command or query

If the connection is working but a command or query is producing a `!BAD` acknowledgement there is a problem with the syntax of the message, i.e. there is something wrong with the text string being sent.

It is possible to turn on the `DBGECHO` feature on to help identify the problem. As an example open an interactive session via PuTTy and check the connection is OK. Then type `$DBGECHO,1;<ENTER>` to turn on the `DBGECHO` feature. Now type the problem command or query. The Gateway should echo good text but replace bad text with a period (full-stop) character. In this example the bad command is `$CHANNEL`; the correct command is `$CHANFADE`.


192.168.2.30 - PuTTY
Problem command
Gateway stop recognising command here
Turn on DBGECHO feature
Two dots caused by <LF><CR> chars

```
!GATRDY;
!VERSION,01.00;
$ok;
!OK;
$channel,1,2,3,255;
!BAD;
$dbgecho,1;
!OK,DBGECHO,1;
..$channel,1,2,3,255;
$chan...........!BAD;
..
```

Note: with HTTP connections you need to include the `$DBGECHO,1;` command at the start of ***every*** AJAX call, as HTTP does not preserve session state between calls.

Note: with Raw IP connections you need to include the `$DBGECHO,1;` command at the start of ***every*** session as soon as the connection is made. Every new session and connection resets back to default settings.

Note: with RS232 connections the settings are only reset back to defaults when the NPU reboots or powers up, or when RS232 port settings are changed.

### 4.2.4 Troubleshooting a command or query with the wrong or no response

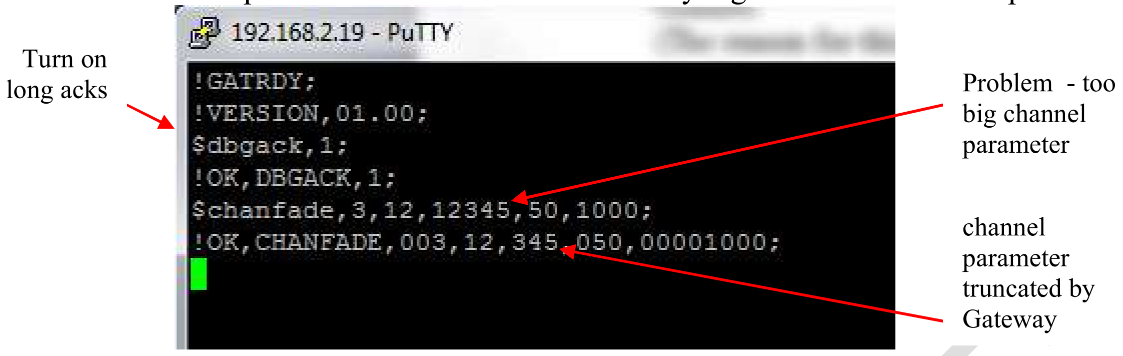
Even if a command or query has been acknowledged with `!OK;` it may not produce the desired response. The most likely problem is that it produces no response at all.

The first thing to note is that the `!OK;` acknowledgment is ***only*** saying that ***the syntax*** of the command or query ***is correct***, i.e. the text string is a recognised command or query. But ***it does not say if the command or query makes sense*** on your actual system. For example, it does not check that the (output) channel that you are trying to set to a level is in fact an output channel and not an input; it does not check that the channel you are querying actually exists.

The second thing to note is that if you ask the system to do something it cannot do, ***it will*** simply ignore the command or request and ***do nothing***. It will ***NOT*** send you message telling you that it cannot perform the requested operation. For example, if you send a query to a channel that does not exist, the Gateway will acknowledge that it has received the query, but no query response will be generated by the non-existing channel.
(The reason for this behaviour is that the Gateway interface itself does not know anything about your system so it cannot check if the channel exists or is of the correct type. This behaviour is applied consistently across all our different types of system.)

To troubleshoot this type of problem, the first thing to check is that the Gateway Interface has interpreted your text string correctly. To do this ensure that long acknowledgement are being generated and inspect the contents of the acknowledge message.

As an example open an interactive session via PuTTy and check the connection is OK. Then type `$DBGACK,1;<ENTER>` to turn on long acknowledgement (on by default). Now type the problem command or query and inspect the ack message. In this example the bad command has too many digits in the <channel> parameter.


192.168.2.19 - PuTTY
Turn on long acks
Problem - too big channel parameter
channel parameter truncated by Gateway

```
!GATRDY;
!VERSION,01.00;
$dbgack,1;
!OK,DBGACK,1;
$chanfade,3,12,12345,50,1000;
!OK,CHANFADE,003,12,345,050,00001000;
```

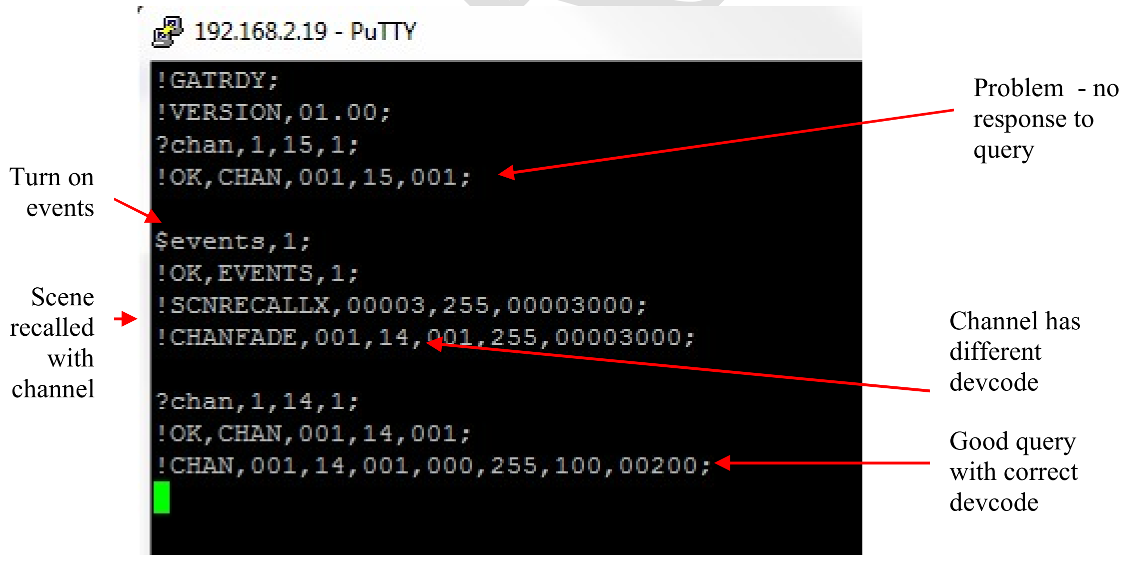
Note: with HTTP connections long acknowledgements are used by default, so you should not need to include the `$DBGACK,1;` command.

The next thing to check is that the parameters are in the correct order. Refer to the Quick Reference or Technical Reference for the format of the command.

If you have checked that you are sending the correct values in the correct order, then check that the channel actually exists *AND* that it is publicly accessible. You will need to look at the systems configuration for the channel/scene accessibility.

### 4.2.5 Troubleshooting using Gateway Events

If your connection is OK and you are sending the command or query with the correct syntax and format, but you are still having problems, then monitoring the live system using Gateway Events could help discover what the problem is.


192.168.2.19 - PuTTY
Turn on events
Scene recalled with channel
Problem - no response to query
Channel has different devcode
Good query with correct devcode

```
!GATRDY;
!VERSION,01.00;
?chan,1,15,1;
!OK,CHAN,001,15,001;

$events,1;
!OK,EVENTS,1;
!SCNRECALLX,00003,255,00003000;
!CHANFADE,001,14,001,255,00003000;

?chan,1,14,1;
!OK,CHAN,001,14,001;
!CHAN,001,14,001,000,255,100,00200;
```

Note: events are not published on HTTP connections. Open a separate rawIP/RS232 session to monitor the live system events, while you send the HTTP AJAX calls.

For example, you want to query a channel but the channel does not return a query response. However, you know from the configuration that the channel is included in a particular scene (or you can create a new scene in the configuration with that channel in it). By opening a new Gateway session, turning on Gateway events, and then recalling the scene (either via the Gateway session or the web page or any other means), you should see an event for the channel of interest and you can compare the event with what you expect for that channel.

# 5 Interface API

This section is used to adjust aspects of the connection. There are 3 standard groups of commands, queries and events messages

a) General purpose
b) Message debugging
   Determines how the connection responds to what is send. Their use has been illustrated in the chapter on Troubleshooting.
c) Event filtering
   Determines which events are sent on the connection

## 5.1 General Purpose Interface API

### 5.1.1 Null Command `$OK;`

The null command can be used at any time to test the connection or keep the connection open. Its response is always a simple `!OK;<CR><LF>`

*Command*: `$OK;`
*Long-ack Format*: `!OK;`
*From: v01.00*
e.g.

```
$ok;
!OK<CR><LF>
```

### 5.1.2 Version Query

The version query can be used at any time to find out the version of the GATEWAY interface that you are connected to. See section on Gateway version numbers.

*Query*: `?VERSION;`
*Long-ack Format*: `!OK,VERSION;`
*Reply: 'Report gateway version'*
*Format*: `!VERSION,<version-text>;`
*From: v01.00*
e.g.

```
?version;
!OK,VERSION;<CR><LF>
!VERSION,01.00;<CR><LF>
```

### 5.1.3 Sign on response

This event is sent when a connection opens to notify you the connection is ready for use. As well as this event, the gateway version is also reported when a connection opens.

*Event: 'gateway ready'*
*Format*: `!GATRDY;`
*From: v01.00*
e.g.

```
!GATRDY;
!VERSION,<version-text>;
```

## 5.2 Message Debugging Interface API

It is possible to adjust the message acknowledge mode and character echo mode of the connection. This is primarily to help with debugging problems on a connection. For details of this see the chapter on Connections and Troubleshooting.

All command and query messages successfully sent to the GATEWAY interface are acknowledged, either with a simple `!OK` or a longer text string that includes the details of the sent message as interpreted by the interface `!OK,…` The longer acknowledge text can be helpful to make sure the interface is correctly interpreting your parameter values. If you are having problems with the eDIN+ not performing your command or query as expected, then check parameter values in the long-ack text are as they should be.

Any command or query message that is acknowledged as `!BAD` instead of `!OK` means that the GATEWAY interface does not recognise part or all of the message. To find out which part of the message it does not like, turn on `DGBECHO` mode.

Normally the characters in the messages sent on a connection or not echoed back. In `DGBECHO` mode, every character sent to the connection is echoed back in some way. The character is sent back unaffected unless it is not liked by the connection, then it is a period character . is sent back. In this way it is possible to see which parts of your message need attention.

### 5.2.1 Long Acknowledge Mode

Query or set Long Acknowledge Mode on or off

*Command:* `$DBGACK,<on-off>;`
*Long-ack Format*: `!OK,DBGACK,<on-off>;`
*From: v01.00*
`on-off` = zero to turn mode off, non-zero to turn mode on
e.g.

```
$dbgAck,1; $dbgAck,0;
!OK,DBGACK,1;<CR><LF>
!OK;<CR><LF>
```

*Query:* `?DBGACK;`
*Long-ack Format*: `!OK,DBGACK;`
*From: v01.00*
*Reply: Report long acknowledge mode*
`!DBGACK,<on-off>;<CR><LF>`
e.g.

```
?dbgAck;
!OK,DBGACK;<CR><LF>
!DBGACK,1;<CR><LF>

?dbgAck;
!OK;<CR><LF>
!DBGACK,0;<CR><LF>
```

### 5.2.2 Debug Echo Mode

Query or set Debug Echo Mode on or off

*Command:* `$DBGECHO,<on-off>;`
*Long-ack Format*: `!OK,DBGECHO,<on-off>;`
*From: v01.00*
`on-off` = zero to echo off, non-zero to turn echo on
e.g.

```
$dbgEcho,1;$dbgEcjl,0;$dbgEcho,0;
!OK,DBGECHO,1;<CR><LF>
$dbgEc....!BAD;<CR><LF>
$dbgEcho,0!OK,DBGECHO,0;<CR><LF>
```

*Query:* `?DBGACK;`
*Long-ack Format*: `!OK,DBGACK;`
*From: v01.00*
*Reply: Report debug echo mode*
`!DBGECHO,<on-off>;<CR><LF>`
e.g.

```
?dbgEcho;
!OK,DBGECHO;<CR><LF>
!DBGECHO,0;<CR><LF>
```

## 5.3 Event filtering Interface API

GATEWAY events and event filtering to monitor an eDIN+ system is an Advanced topic and is included in Volume 2. However, seeing events can be very helpful when debugging a connection. This section includes a command to turn event messages on and off.

No events are sent on a session connection until the connection requests them. This avoids the session being flooded with unwanted event messages. To turn all event messages on use the 'Set Event Reporting for All events' command.

### 5.3.1 Set Event Reporting for All events

Turn reporting of all events either on or off.
*Command:* `$EVENTS,<on-off>;`
*Long-ack Format*: `!OK,EVENTS,<on-off>;`
*From: v01.00*
`on-off` = zero to turn all event flags off, non-zero to turn all event flags on
e.g.

```
$Events,1;
!OK,EVENTS,1;<CR><LF>
```

# 6 Scene and System API

This section is used to interact with the in-build controller of the eDIN+ system. If there is no active in-built controller these messages will not work and you will need to use the Channel API. There are 4 standard groups of commands, queries and events messages

a) Scene Control
   To control and query scene state
b) Live Scene Setting
   To perform simple scene setting
c) System Health
   Determines health and error status of system and its components
d) System Discovery
   To obtain information about systems components including areas and scenes.

## 6.1 Scene Control

The GATEWAY interface provides a full set of scene commands to match those available in the configuration. However to turn scenes on and off and adjust their levels you are unlikely to need to use the full set unless you are trying to mimic e.g. the behaviour of a wall plate button in the configuration.

This volume describes the most useful commands. Volume 2 describes the full set.

A lighting scene is used to control a number of lighting channels in a single operation. It has a *fade time* that defines how fast to control the channels (this can be overridden), and an *absolute* scene level that controls how bright to make each channel. It has a collection of lighting channels to control, each lighting channel having a *relative* level at which it is set at; different channels can be set to different levels.

Changing or adjusting the relative channel levels is called ‘scene setting’. All other actions on scenes do not affect the relative channel levels, but do affect the absolute scene level.

Every scene is identified through the GATEWAY by its *scene number*, which is used for the `<scn-num>` parameter in all scene messages.

Every scene also has its own status – it is either active (on) or inactive (off). Both current scene status and level that can be monitored and queried.

The rules that determine when a scene is active and inactive are involved and are configurable. However, in its default state, a scene is active when it has been recalled, and inactive when another *overlapping* scene is recalled – scenes are *overlapping* when the same (or more than one) channel is in both scenes. More information about this can be found in Volume 2.

The most useful command is `$SCNRECALLX`. This allows you to set the scene level to any value using any fade time. To turn a scene on set level to max (`255`), to turn a scene off set level to zero (`0`), or to any intermediate level via e.g. a slider control. This command allows you to override the scenes *natural* behaviour set by the configuration and behaves as you want it.

If you want to work with the configuration then the `$SCNRECALL` and `$SCNOFF` will recall the scene on and off that matches how e.g. wall plates operate.

If you want to toggle a scene on and off from a single button and not be concerned about the actual value of scene state, then the `$SCNONOFF` command can be used.

### 6.1.1 Scene Recall Commands

Recall scene action.
Make the scene active at maximum level and set the channels to the levels defined in the scene, using the scene's fade time defined in the configuration.

*Command:* `$SCNRECALL,<scn-num>;`
*Long-ack Format*: `!OK,SCNRECALL,<scn-num>;`
*From: v01.00*
e.g.
```
$scnrecall,8;
!OK,SCNRECALL,00008;<CR><LF>
```

- Recall scene to off (level = 0) action.
  This makes the scene inactive, sets the scene level to zero and set the channels defined in the scene to level = 0, using the scene's fade time defined in the configuration.

  *Command:* `$SCNOFF,<scn-num>;`
  *Long-ack Format*: `!OK,SCNOFF,<scn-num>;`
  *From: v01.00*
  e.g.
  ```
  $scnOff,8;
  !OK,SCNOFF,00008;<CR><LF>
  ```

- Recall scene action with explicit values.
  Perform a scene recall action using the explicit parameters, rather than the default values in the configuration.
  Note: recalling the scene to level = 0 is the same as turning the scene off and making it inactive.
  Note: this command specifies the scene level not channel levels. If a channel is defined in the scene at say 70%, and you recall the scene at 50% (level=128), then the channel will be set to 70% x 50% = 35%.

  *Command:* `$SCNRECALLX,<scn-num>,<scn-level>,<fadetime(ms)>;`
  *Long-ack Format*: `!OK,SCNRECALLX,<scn-num>,<scn-level>,<fadetime>;`
  *From: v01.00*
  `scn-level: 0-255. 0` = off, `255` = max (100%).
  `fadetime: 0-8388607` milliseconds
  e.g.
  ```
  $scnrecallx,8,255,60000;
  !OK,SCNRECALLX,00008,255,00060000;<CR><LF>
  ```

### 6.1.2 Scene Toggle Commands

Toggle commands toggle the scenes state from on to off or vice versa. Each is effectively 2 recall commands in one – one to recall the scene on and the other to recall the scene off. It automatically uses the recall action that will always change or toggle the current scene state.

These commands are useful if you want to toggle a scene on and off from a single button and not be concerned about the actual value of scene state.

There are 2 varieties. Both turn off the scene in the same way but differ in how they turn the scene on.

- The first `$SCNONOFF` is the most common and always uses the normal maximum scene level to recall the scene back on.
- The second `$SCNTOGGLE` uses a 'remembered snap-shot' of scene and channel levels when turning the scene back on. Although this behaviour can be useful if used on its own, it can get confusing if used with other toggle and recall commands. This is described in Volume 2.

▪ Recall On/Off scene action.
  (Toggle between `$SCNRECALL` and `$SCNOFF` actions).
  This command toggles a scene on and off. The scene is set off if the scene is currently active, or is recalled to max if the scene is currently inactive, both using the scene' default fade time.
  Note: If used in conjunction with scene commands that adjust the scene level, it resets the levels effectively making any adjustments temporary.

  *Command:* `$SCNONOFF,<scn-num>;`
  *Long-ack Format*: `!OK,SCNONOFF,<scn-num>;`
  *From: v01.00*
  e.g.

```
$scnOnOff,8;
!OK,SCNONOFF,00008;<CR><LF>
```

## 6.2 Scene Status

Each scene has a *scene state* that says whether a scene is *in use*; the state can be *active* or *inactive*. The eDIN+ system displays this state on plate buttons and elsewhere. The scene has a scene level that can be reported.

There is no agreed standard of when a scene is *in use* and *active* and when it is no longer in use and *inactive*. There is general agreement that a scene becomes *active* immediately it is recalled, but when does it become *not in use* and *inactive*? As different people have different expectations about what should happen in specific situations, the eDIN+ system can be configured to meet these expectations.

There are a few occasions when you would like to know for a particular scene how *in use* is calculated and what *active* means, but for the most part it is unimportant. When you request the scene status, each reply message returns the scene's mode and flag configuration properties that indicates what *active* means, but **it is always safe to**

**ignore the mode and flags parameters**. See Volume 2 for information on this and these properties.

Summary
`scn-num` = scene number used to identify the scene, as a decimal value.
`scn-state` = `0` (inactive) or `1` (active)
`scn-level` = `0` (off) or `1-255` (on)
`mode, flags` = fixed value indicating *in use* meaning - see Volume 2.

### 6.2.1 Scene Status

- Request scene status
  This is used to find out the current active/inactive status and level of one or more scenes.

  You can query all scenes, scenes in a particular area or a single scene.

  *Query:* `?SCNS;`
  *Query:* `?SCNS,<area-num>;`
  *Query:* `?SCN,<scn-num>;`
  *Long-ack Format*: `!OK,SCNS;` or `!OK,SCNS,<area-num>;` or `!OK,SCN,<scn-num>;`
  *From: v02.00 (v01.x replies do not include `<scn-level>` parameter)*
  *Reply: zero or more 'Report scene status'*
  `!SCN,<scn-num>,<mode>,<flags>,<scn-state>,<scn-level>;`
  e.g.

```
?scns,1;
!OK,SCNS,00001;
!SCN,00001,01,02,001,255;<CR><LF>
!SCN,00002,02,02,000,000;<CR><LF>
```

## 6.3 System Health

### 6.3.1 System Error List

The in-built controller maintains a list of all current channel errors. If the list contains any entries, there is a problem with the system and it will not work properly until the problem is fixed.

Note that this is not a log of any error that has occurred; it only holds channel errors that are current or live. If an error is intermittent, it will appear and disappear from the list as the error itself comes and goes. Also the list only holds current channel *errors*; it is not a list of every channel and their status; it does not include channels with no errors, i.e. channels that have the status code `0 Status Ok`.

- There are a number of flavours of Channel Status Error message, for the different types of channel and their `CHAN`, `DALI`, `DMX`, `BTN,` and `INP` identifiers, otherwise the messages are the same and report the error as a `status-code`. See Appendix  request input channel status
  *Query:* `?INP,<addr>,<devcode>,<chan-num>`
  *Long-ack Format*: `!OK,INP,000,000,000;`

*From: v02.00*
*Reply: 'Report input channel health' and one of 'Report input channel state*
```
!INPERR,<addr>,<devcode>,<chan-num>,<status-code>;
!INPSTATE,<addr>,<devcode>,<chan-num>,<status-code>,<state>;
!INPPIR,<addr>,<devcode>,<chan-num>,<status-code>,<state>;
!INPLEVEL,<addr>,<devcode>,<chan-num>,<status-code>,<state>;
```
e.g.
```
?inp,15,21,1;
!OK,INP,015,021,001;<CR><LF>
!INPERR,015,021,001,000;<CR><LF>
!INPPIR,015,021,001,000;<CR><LF>
```

Enumerations for a full list of values and their meaning.

In addition to the channel error messages, and error can be reported on a module if the problem occurs on all channels of a module. For example, a dimmer module has an AC mains input to power the module's output channels. If the module detects that there is no AC mains power coming in to the module, it will be reported with a single `MODULEERR` message, rather than being reported as 4 or 8 individual channel errors.

Also, errors can be reported on DALI fixtures – see Volume 2 Advanced DALI Fixture Health for details.

- Request system error list
  *Query:* `?ERRORS;`
  *Long-ack Format*: `!OK,ERRORS;`
  *From: v02.01*
  *Reply: list of errors*
  *zero or more Report Channel Health messages*
```
!MODULEERR,<addr>,<devcode>,<status-code>;
!CHANERR,<addr>,<devcode>,<chan-num>,<status-code>;
!DALIERR,<addr>,<devcode>,<dali-num>,<status-code>;
!DALIERR,<addr>,<devcode>,<dali-fixture>,<status-code>;
!DMXERR,<addr>,<devcode>,<zone-num>,<status-code>;
!BTNERR,<addr>,<devcode>,<btn-num>,<status-code>;
!INPERR,<addr>,<devcode>,<chan-num>,<status-code>;
```
e.g.
```
?errors;
!OK,ERRORS;<CR><LF>
!MODULEERR,003,015,002;<CR><LF>
!DALIERR,001,017,001,025;<CR><LF>
!DALIERR,001,017,F03,025;<CR><LF>
```

### 6.3.2 Polling for Channel Status
Instead of using the System Error List to detect problems, it is possible to determine the health and status-code of individual channels by requesting a channel's status using Channel API query messages – see Section 7 Channel API . As these Channel Status queries return more than just the health of a channel, using this method may be more convenient.

Note that channel status-codes reflect module errors automatically as is natural. For example if a module has a 'No AC Mains' error then all channels have this problem and will report the status code for ‘No AC Mains’ error. So polling individual channel status will not miss module errors.

## 6.4 System Discovery

There is some limited discovery queries in the GATEWAY interface. In general, it is assumed that any third party applications that use the Gateway Interface know about the specific eDIN+ system it is trying to control and monitor, and know what scenes and channels exist and how to access them. The available discovery queries provide extra information on the *known* scenes and wall plates.

However, the discovery queries can also tell you what scenes and wall plates are available in the system, so can be used for limited discovery.

If further information about what is in the configuration is required, it is also possible to obtain fuller information in text files following the Comma-Separated-Variable format. These CSV files can be obtained manually via the eDIN+ online or offline editors and these may be useful when developing a system for a specific site.

These CSV files can also be obtained via a different ‘Info’ NPU web service, that allows a system to obtain an up-to-date version of the files for a known or unknown system. Further details of this topic are in Volume 3.

### 6.4.1 Configuration Discovery Queries

A limited amount of information can be found about the configuration through the discovery queries. However, these queries only allow you to view certain things in the configuration, not change them, and they do not provide full access to the configuration.

The main discovery queries are `?SYSTEMID`, `?AREANAMES`, `?SCNNAMES` and `?PLATENAMES`. Information about individual channels is available with the `?BTNSTYLE` and `?SCNCHANNAMES` queries.

Discovery is centred around the *areas* defined in the configuration. In eDIN+ systems *areas* are useful to help provide order to the configuration, and group together scenes and channels. First obtain a list of available areas, together with their area number and name using the query `?AREANAMES`.

However *areas* are not essential to the operation of an eDIN+ system. You do not need to know about areas to use the GATEWAY interface - areas can be ignored. The `?SCNNAMES` and `?PLATENAMES` queries can be used with or without areas.

The `?SCNNAMES` query returns a list of either all available scenes or available scenes in a particular area. For each scene, the scene number and scene name are returned.

The `?PLATENAMES` query returns a list of either all available wall-plates or available wall-plates in a particular area. For each plate, its address and device code are returned together with its name and what type of wall-plate it is. You can find out about the buttons on a wall-plated using the query `?BTNSTYLE`. These 2 queries allow you to mimic the actual look of wall-plates if desired.

The information from these discovery queries is static and unchanging unless the configuration is changed, so these queries can be performed once or only when the configuration is changed.

The query `?SYSTEMID` can be used to check if the configuration has changed, allowing you to re-discover the information as necessary. Note: if a configuration has user accounts, as each account provides a different *window* on to the configuration and can change the available scenes and plates without changing the *system ids* – see Volume 2 for details.

The `?SCNCHANNAMES` query is used with live scene setting and described in that section.

Many of the query responses have an `access` value, relating to its accessibility. Accessibility is an advanced topic and is covered in Volume 2 if you would like further information on this. **In general you can safely ignore this `access` value**.

<u>Summary</u>

`scn-num` = scene number used to identify the scene, as a decimal value.
`area-num` = area number used to identify the area, as a decimal value.
`addr,devcode` = address and device-code of wall plate used to identify it.
`scn-name` = descriptive name for scene in UTF-8 encoded text.
`area-name` = descriptive name for area in UTF-8 encoded text.
`plate-name` = descriptive name for plate in UTF-8 encoded text
`area-content` = `1` (has channels) + `2` (has scenes) + `4` (has plates).
`access` = bit-field flags `1` (viewable) + `2` (controllable) + `4` (editable).
`plate-style` = `1` SGP style, `2` Coolbrium style, `3` ICON style, `4` Geneva style,
`5` to `8` rotary plate with 1 to 4 buttons respectively
`btn-icon` = reference number for button Icon

- Request scene areas
  This returns a list of available areas in the configuration. Each area has an `area-number` that the GATEWAY interface uses to identify the area, and a descriptive `area-name` that may be blank if it has not been set up in the configuration.

  As part of the configuration, every scene, wall-plate and channel is put in to an area. The `content` value summarises if any channels, scenes or plates have been assigned to this area, that may help with subsequent discovery queries but can safely be ignored.

  *Query:* `?AREANAMES;`
  *Long-ack Format*: `!OK,AREANAMES;`
  *From: v01.00*
  *Reply: set of area names*
  *zero or more 'Report Area Name'*
  `!AREANAME,<area-num>,<access>,<area-content>,<area-name>;`

(There is no end-of-list message).
e.g.
```
?areaNames;
!OK,AREANAMES;<CR><LF>
!AREANAME,00000,07,001,;<CR><LF>
!AREANAME,00001,07,007,Main Hall;<CR><LF>
!AREANAME,00002,07,003,Outside porch;<CR><LF>
```

- Request scene names
  This returns a list of scenes. You can query for all available scenes, or just scenes in a particular area.

  Each scene has an `scn-number` that the GATEWAY interface uses to identify the scene, their area (`area-num`) it has been assigned to and a descriptive `scn-name` that may be blank if it has not been set up in the configuration.

  If the area number parameter is not given in the query then all scenes are reported.

  *Query:* `?SCNNAMES;`
  *Query:* `?SCNNAMES,<area-num>;`
  *Long-ack Format*: `!OK,SCNNAMES;` or `!OK,SCNNAMES,00000;`
  *From: v01.00*
  *Reply: set of scene names*
  *zero or more 'Report Scene Name'*
  `!SCNNAME,<scn-num>,<access>,<area-num>,<scn-name>;`
  (There is no end-of-list message).
  e.g.
```
?SCNNAMES,1;
!OK,SCNNAMES,00001;
!SCNNAME,00003,07,00001,Off;<CR><LF>
!SCNNAME,00004,07,00001,On;<CR><LF>
!SCNNAME,00005,07,00001,Kitchen;<CR><LF>
```

- Request plate names
  This returns a list of wall plates, the type of plate they are, the area they are in and their name. You can query for all available plates, or just plates in a particular area.

*Query:* `?PLATENAMES;`
*Query:* `?PLATENAMES,<area-num>;`
*Long-ack Format*: `!OK,PLATENAMES;` or `!OK,PLATENAMES,<area-num>;`
*From: v01.00*
*Reply: set of plate names*
*zero or more 'Report Plate Name'*
`!PLATENAME,<addr>,<devcode>,<plate-style>,<access>,<area-num>,<plate-name>;`
If the area number parameter is not given in the query then all plates are reported. (There is no end-of-list message).
e.g.

```
?PLATENAMES,1;
!OK,PLATENAMES,00001;
!PLATENAME,001,002,03,07,00001,Main;<CR><LF>
!PLATENAME,002,002,01,07,00001,Bedside;<CR><LF>
```

- Request button styles
  This returns a list of buttons on a particular wall plate. Each button specifies the type of button they are and their name.

  *Query:* `?BTNSTYLE,<addr>,<devcode>,ALL;`
  *Query:* `?BTNSTYLE,<addr>,<devcode>,<btn-num>;`
  *Long-ack Format*: `!OK,BTNSTYLE,<addr>,<devcode>,<ALL || btn-num>;`
  *From: v01.00*
  *Reply: zero or more 'Report Button Style'*
  `!BTNSTYLE,<addr>,<devcode>,<btn-num>,<btn-icon>,<style>,<btn-name>;`
  `style` = not used
  e.g.

```
?BTNSTYLE,8,2,1;
!OK,BTNSTYLE,008,002,001;
!BTNSTYLE,008,002,001,001,000,;<CR><LF>
```

- Request system ID
  This is used to check if the configuration has changed since that last time you checked. If either stamp value has changed since you last checked, then the configuration has been altered in some way and you may need to re-discover.

  The edit-stamp is changed whenever the structure of the configuration has changed, for example if a new scene is added, and when any static property value changes, such as a scene name. These can only be done outside the GATEWAY.

  The adjust-stamp is changed when minor adjustments to the configuration are made by a user, primarily scene setting operations, either via the GATEWAY or outside of the GATEWAY. These changes are generally not important for successful control via the GATEWAY, but may be important if you are locally saving/remembering scene setting information.

  *Query:* `?SYSTEMID;`
  *Long-ack Format*: `!OK,SYSTEMID;`
  *From: v01.00*
  *Reply: ID stamps that uniquely define a configuration state*
  `!SYSTEMID,<serial-num>,<edit-stamp>,<adjust-stamp>;`

`serial-num` = serial number of NPU interface, as an 8-digit hexadecimal value.
`edit-stamp` = text string representing when the last edit was saved
`adjust-stamp` = text string representing when the last adjustment was saved.
e.g.

```
?systemid;
!OK,SYSTEMID;<CR><LF>
!SYSTEMID,0002016D,7027-55441,7027-55441;<CR><LF>
```

## 6.5 Live Scene Setting

The GATEWAY provides 2 very different ways to permanently change the channel levels defined in a scene: the *live* method using `SCNSAVE` and the advanced *offline* method using `SCNSET`. This section describes the *live* method. Information on the advanced *offline* method can be found in Volume 3.

The live scene setting method is very simple – use the `$SCNSAVE` command at any time to capture the current *live* state/level/colour of all the channels in the selected scene. All subsequent operations on the scene will now use these new values.

If a scene is too bright or dim, use the `$SCNRECALLX` command to set the scene to the correct level, then use `$SCNSAVE` to save that setting.

If one or more channel in the scene is the wrong setting, rather than the whole scene, then use the appropriate Channel API command to set the level or colour of the desired channels, then use `$SCNSAVE` to save the new value in the scene. It is recommended to recall the (complete) scene before changing channel levels – see things to be aware of below

There are 2 queries that can be used to discover what channels are in a scene. Both provide a list of channel entries in the selected scene. `?SCNCHANSTATES` returns the current level or state of each channel entry, including current *live* level or colour. `?SCNCHANNAMES` returns the name (and area) of each channel entry. Colour and tuneable white channels have separate entries in a scene to control level and colour.

Scene can include the following channel entries (`entry-id` in messages). These match the types of control in the Channel API.

- `CHAN,DALI,DMX` – level/brightness control
- `DMXRGBCOLR,CHANRGBCOLR,DMXRGBPLAY` – colour control
- `DMXTWCOLR,CHANTWCOLR` - tuneable white/colour temperature control

Things to be aware of

- Using `$SCNSAVE` is an *adjustment* and will change the `adjust-stamp` in `?SYSTEMID` query.
- It is not possible to exclude any entries/channels from the `$SCNSAVE` operation. All scene entries will be set to its current live value. It is recommended to set all channels to the current scene values by recalling the scene before you begin adjusting the individual channels. Then only the channels you have adjusted will change value.

- It is not possible to include new channels or entries with a `$SCNSAVE` operation. You can only adjust the existing entries in a scene.
- Scenes do *not* have to include *both* level and colour entries for colour or tuneable white channels. The `$SCNSAVE` operation will only save the level or/and colour of these channels if it has an entries for them. The other values will be ignored.
- This method does not allow you to:
    - define new scenes.
    - change which channels or entries are in the scene.
    - change the scene's default fade time.
- This method only changes the scene definition if the scene allows editing (access level edit flag set) – see Volume 2 about scene and channel accessibility.
  - Only editable scenes will report their channels.
  - Only controllable channels are reported.

### 6.5.1 Save Scene Levels

- Save scene levels action – permanently reprogram the channel levels of a scene to the current channel levels

  *Command:* `$SCNSAVE,<scn-num>;`
  *Long-ack Format*: `!OK,SCNSAVE,<scn-num>;`
  *From: v01.00*
  e.g.

```
$scnSave,8;
!OK,SCNSAVE,00008;<CR><LF>
```

### 6.5.2 Channel discovery for scenes

- Request channel names for scene
  This query returns a list of channel entries, and gives the name and area of each entry.

  *Query:* `?SCNCHANNAMES,<scn-num>;`
  *Long-ack Format*: `!OK,SCNCHANNAMES,00000;`
  *From: v02.01*
  *Reply: set of channel names in the specified scene*
  *zero or more 'Report Channel Name'*

```
!<entry-id>NAME,<addr>,<devcode>,<chan-num>,<access>,<area-
num>,<chan-name>;
```

  `entry-id` = one of the following:
  - `CHAN,DALI,DMX` - levels
  - `DMXRGBCOLR,CHANRGBCOLR,DMXRGBPLAY` - colour
  - `DMXTWCOLR,CHANTWCOLR` - tuneable white

  e.g.

```
?SCNCHANNAMES,4;
!OK,SCNCHANNAMES,00004;
!CHANNAME,002,012,001,03,00001,Main Ceiling;<CR><LF>
!DMXNAME,003,015,002,03,00001,Features;<CR><LF>
!DMXRGBCOLRNAME,003,015,002,03,00001,Features;<CR><LF>
!CHANNAME,004,017,001,03,00001,Side lights;<CR><LF>
!DALINAME,004,017,012,03,00001,Downlights;<CR><LF>
!CHANTWCOLRNAME,005,018,001,03,00001,Ambient;<CR><LF>
```

- Request channel status for scene
  This query returns a list of channel entries and their current live status/level/colour. The 'Report item status' response messages are the same as the replies from channel status queries. See Channel API for more details of querying channel status.

  *Query:* `?SCNCHANSTATES,<scn-num>;`
  *Long-ack Format*: `!OK,SCNCHANSTATES,<scn-num>;`
  *From: v02.01*
  *Reply: set of item statuses in scene*
  *zero or more 'Report (item) state'* – see Channel API for details
  e.g.

```
?SCNCHANS,4;
!OK,SCNCHANS,00004;<CR><LF>
!CHANLEVEL,002,012,001,000,100,00100;<CR><LF>
!DMXLEVEL,003,015,002,000,033,00100;<CR><LF>
!DMXRGBCOLR,003,015,002,000,#FF7E00;<CR><LF>
!CHANLEVEL,004,017,001,000,100,00100;<CR><LF>
!DALILEVEL,004,017,012,000,100,02500;<CR><LF>
!CHANTWCOLR,005,018,001,048,#2200K;<CR><LF>
```

# 7 Channel API

This section is used to interact with the actual hardware of the eDIN+ system. Care should be taken using this API if the eDIN+ system has an active in-built controller, as the controller will be also controlling the hardware.

In most cases, channel control and monitoring is simple and straight-forward. Most of the hardware in an eDIN+ system will be either an output lighting channel that can be set to a simple dimmable level, or a contact input channel or plate button that will trigger events and actions.

However, the real world does not always fit neatly in to simple categories. This makes the details of the Channel API more complex than the Scene & System API, but keep in mind that the Channel API messages are variations on a theme – they are setting the brightness (or colour) of an output channel, or querying the state of an input channel.

Before you use the Channel API, it would help if you are familiar with building a simple eDIN+ configuration, so that you are familiar with the different channel types, the way they are identified, and what you can do with them.

Volume 2 has information on Channel API messages that allow for more advanced control and monitoring of channels.

If you want to be the main controller and inhibit in-built controller Volume 3 has information about this.

## 7.1 Overview

### 7.1.1 Different types of Channels

Here are some things that do not fit neatly relating to the eDIN+ system and GATEWAY interface.

- Most outputs are identified using the `CHAN` notation. There are 2 exceptions
  i) `DALI` notation.
  Lighting channels on the DALI universe of UBC modules (often called *addressable DALI*) use the `DALI` notation instead of `CHAN`. This is because there are both ordinary `CHANs` and `DALI` outputs on the same module, and we need a way to identify when we are referring to the ordinary channels and when we are referring to the DALI outputs.
  ii) `DMX` notation
  The 8 channel IO module has both ordinary channels and DMX zones. Again we need a way to separate these so the DMX zones use the `DMX` notation instead of `CHAN`.

  Use `CHAN` unless the configuration indicates otherwise. Take your lead from the configuration – ***do not assume*** the `DALI` notation ***relates to all DALI fixtures***. Particularly, outputs on the DBM module are always identified with the `CHAN` notation, even if they are configurated to be DALI broadcast channels.

The consequence of the above means that most channel commands, queries and events come in 3 flavours – they are identical in all respects except they use `CHAN`, `DALI` or `DMX` as appropriate.

- Plate buttons are both inputs and outputs. They consist of a momentary contact switch input, and have colour LEDs and (for LCD plates) text that are outputs that can be controlled.

  This means that `BTN`s have both control messages and input events.

- Outputs that drive colour or tuneable white fixtures have 2 controls – one for the level or master brightness, and one for the colour. To identify between the 2 output controls, the level is controlled using the `CHAN` or `DMX` notation as though it was a normal output. The colour control adds a suffix `RGB` or `TW` to the channel notation, e.g. `CHANTW` or `DMXRGB`.

- Input channels come in different varieties. Sometimes you do not need to know which variety a channel is, when the `INP` notation is used. At other times the you need to know in which case the follow notations are used.
  - `INPSTATE` contact-input
  - `INPPIR` pir-input
  - `INPLEVEL` analogue-input

- Modules are needed to identify a particular channel but are less important for control. As a result there are very few messages that refer to a module as a whole, rather than the individual channels within the module.

### 7.1.2 Channel Identification

All *channel*s are identified with 3 items: first by the module's *MBus address* then the module's Mode *device code* and then the *channel number* within the module, shown as `<addr>,<devcode>,<chan-num>` in message descriptions below.

Each module and plate is set with a configurable *MBus address* in the range `1 – 511`, depending upon system configuration. Use the relevant address for the module.

Each module has a specific fixed *device code* that identifies the type of module. You *must* use the correct device code for the module. A list can be found in the appendix.

Each channel on a module is identified by a number, identified on the module itself and in a configuration. There are a few exceptions.

- `DALI` channels (on an *addressable* DALI universe):
  The Mode *DALI channel number* is used - `<addr>,<devcode>,<dali-num>`. Lighting ballasts on a DALI universe are assigned to a Mode DALI channel number during configuration/commissioning. It is this Mode DALI channel number that is used by this interface.

  Some messages use `<addr>,<devcode>,<dali-id>`. For these messages think of them as `<dali-num>` parameters and use the Mode DALI channel number.

`<dali-id>` parameters can also use advanced DALI Broadcast and Group Id values. See Volume 2 on Advanced DALI Addressing for further details.

Remember for modules that have DALI Broadcast output channels, you MUST use the standard `CHAN` commands. The `DALI` commands are only for UBC modules with addressable DALI outputs.

- `DMX` zones:
  The DMX zone number is used instead of a channel number -
  `<addr>,<devcode>,<zone-num>`.

  When using the DMX commands and queries the module's DMX universe and specified DMX zone both must be enabled. This is done during configuration/commissioning and cannot be controlled through the GATEWAY interface.

- plate `BTN` buttons:
  The button number is used instead of a channel number -
  `<addr>,<devcode>,<btn-num>.`
  It is also possible to specify the `<btn-num>` as `ALL`, to set or query all buttons on the plate.

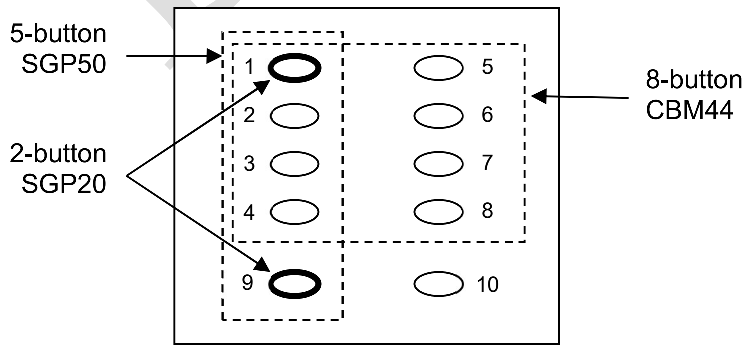
  Beware: button numbering is not obvious – see the section below.

  Note: Plates also have contact input channels. Use `INP` and *channel numbers* to identify the external contact inputs on the back of the plate.

### 7.1.3 Plate Button Identification and Numbering

All plates use the same `EVO-SGP-xx` *device code* '`2`', except for the LCD plate that uses `EVO-LCD-55` *device code* '`1`', and the RD-00-04 rotary switch plate that uses `EVO_RD_00_04` *device code* '21'

All button numbering is based on the SGP55 10-button plate numbering. So 2- and 5-button plates use 'Button 9' for their last button. Coolbrium 8-button plates use button 1-8. See diagram below.



For the Rotary plate, button numbering is from 1 across to a maximum of 4, usually 1 being the left-most knob, but it depends on how the plate assemblies are physically mounted.

## 7.2 Output Control

### 7.2.1 Set Output Level

Set the channel's output gradually to the specified final level fading over the specified time.

Levels are in the range 0 (minimum) to 255 (maximum), instead of percentages. This matches the 8-bit internal value and gives you maximum control of the level value. Specify a level of zero to turn the channel off, any other value to turn the channel on.

Fade times are specified in one-thousandths of a second or milli-seconds. This allows you to specify fractions of a second if required. Use a fade time of zero to snap the output to its new level.

Use this command for relay channels and other non-dimmable channels even through their outputs can be only full on or full off.

- To make them snap on or off as soon as the command is sent, use a fade time of zero.
- To make them turn on & off at the same time as other dimmable channels, use the same fade time as the other channels.
  - When fading to off the channel will remain on until the end of the fade time, when it will be switched off.
  - When fading on then channel will turn on instantly.

This behaviour makes it easy to drive a relay channel that is wired to an e.g. Analogue 1-10V dimmable channel. Simply send the same command to both channels.
(Note: not all relay channels support fade operations, in which case they will ignore the fade time and always snap.)

- Set channel to level with fade
  *Command:* `$CHANFADE,<addr>,<devcode>,<chan-num>,<level>,<fadetime(ms)>;`
  *Long-ack Format*: `!OK,CHANFADE,…;`
  *From: v01.00*
  e.g.
  ```
  $ChanFade,1,12,2,30,3000;
  !OK,CHANFADE,001,012,002,030,00003000;<CR><LF>
  ```

- Set DALI channel to level with fade
  *Command:* `$DALIFADE,<addr>,<devcode>,<dali-num>,<level>,<fadetime(ms)>;`
  *Long-ack Format*: `!OK,DALIFADE,…`
  *From: v01.00*
  e.g.
  ```
  $dalifade,02,17,016,255,1000;
  !OK,DALIFADE,002,017,016,255,00001000;<CR><LF>
  ```

- Set DMX zone brightness to level with fade
  *Command:* `$DMXFADE,<addr>,<devcode>,<zone-num>,<level>,<fadetime(ms)>;`
  *Long-ack Format*: `!OK,DMXFADE,…`
  *From: v02.00*
  e.g.
  ```
  $dmxfade,02,15,03,255,1000;
  !OK,DMXFADE,002,015,003,255,00001000;<CR><LF>
  ```

### 7.2.2 Stop Output Fade
Stop a channel fading prematurely before it has reached its final level. If no fade is active or if the channel is already at its final level, no action is taken.

- Stop (output) channel fade
  *Command:* `$CHANSTOP,<addr>,<devcode>,<chan-num>;`
  *Long-ack Format*: `!OK,CHANSTOP,<addr>,<devcode>,<chan-num>;`
  *From: v01.00*
  e.g.
  ```
  $chanstop,26,16,4;
  !OK,CHANSTOP,026,016,004;<CR><LF>
  ```

- Stop DALI channel fade
  *Command:* `$DALISTOP,<addr>,<devcode>,<dali-num>;`
  *Long-ack Format*: `!OK,DALISTOP,<addr>,<devcode>,<dali-num>;`
  *From: v01.00*
  e.g.
  ```
  $DALIstop,4,17,12;
  !OK,DALISTOP,004,017,012;<CR><LF>
  ```

- Stop DMX zone brightness fade
  *Command:* `$DMXSTOP,<addr>,<devcode>,<zone-num>;`
  *Long-ack Format*: `!OK,DMXSTOP,<addr>,<devcode>,<zone-num>;`
  *From: v02.00*
  e.g.
  ```
  $DMXStop,4,15,2;
  !OK,DMXSTOP,004,015,002;<CR><LF>
  ```

## 7.3 Output status
Determine the state of output channels.

The basic query returns the health of the channel as a status code and its current level together with an estimate of the current output power.

If the channel is a colour or tuneable white channel the basic query also returns the current colour – see section on colour and tuneable white status for details.

### 7.3.1 Request Output Status
There are 3 flavours of Query messages for the three `CHAN`, `DALI` and `DMX` identifiers, otherwise the queries are the same.

A status code of `0` means the channel is OK. Any other values indicates there is some problem with the channel. See Emulation Appendix for values and their meaning.

The level is in the range `0` (minimum) to `255` (maximum), instead of percentages. This matches the values used in the Control commands. A level of zero means the channel is off, any other value means the channel is on. Relay channels and other non-dimmable channels return a level value of `0` or `255`.

An estimate of the channel's current instantaneous power usage is also given. The power is based on the output channel's level with various adjustments: it takes in to consideration the channel's dimming curve and cup & cap value. However, the **reported power is just an estimate**, not a measurement, of power usage. If you need *accurate* power usage, using an *external power meter* will give better results.

The channel's power is reported in 2 parts: a percentage of maximum output power and the Wattage used at maximum power. The Wattage is the value set in the configuration, fixed for each channel. The power percentage converts the current control level value in to power value.

On a small technical note, the level and power reported assumes the channel is not in an active fade (i.e. not actively changing its level) and always reports the last fade operation's final value. In other words it assumes all fade commands perform a snap. This works well when channel levels are mostly unchanging and change relatively quickly when they do change. If this is a problem because channels are continually changing over long fade times, see Volume 2 on ways to request actual current level.

Summary
`status-code` = `0` (OK). see Emulation Appendix for other values
`level` = level (`0-255`) or relay state (`0` or `255`)
`power` = percentage of maximum wattage (`0-100`)
`wattage` = wattage at maximum level/power in Watts

- request (output or relay) channel status
  *Query:* `?CHAN,<addr>,<devcode>,<chan-num>;`
  *Long-ack Format*: `!OK,CHAN,<addr>,<devcode>,<chan-num>;`
  *From: v02.00*
  *Reply: 'Report output health' and 'Report output level'*
  ```
  !CHANERR,<addr>,<devcode>,<chan-num>,<status-code>;
  !CHANLEVEL,<addr>,<devcode>,<chan-num>,<level>,<power>,<wattage>;
  ```
  e.g.
  ```
  ?chan,1,12,2;
  !OK,CHAN,001,012,002;<CR><LF>
  !CHANERR,001,012,002,000;<CR><LF>
  !CHANLEVEL,001,012,002,030,099,00100;<CR><LF>
  ```

- request DALI channel status
  *Query:* `?DALI,<addr>,<devcode>,<dali-num>;`
  *Long-ack Format*: `!OK,DALI,<addr>,<devcode>,<dali-num>;`
  *From: v02.00*
  *Reply: 'Report DALI channel health' and 'Report DALI channel level'*
  ```
  !DALIERR,<addr>,<devcode>,<dali-num>,<status-code>;
  !DALILEVEL,<addr>,<devcode>,<dali-num>,<level>,<power>,<wattage>;
  ```
  e.g.
  ```
  ?dali,1,17,2;
  !OK,DALI,001,017,002;<CR><LF>
  !DALIERR,001,017,002,000;<CR><LF>
  !DALILEVEL,001,017,002,254,099,02500;<CR><LF>
  ```

- request DMX zone status
  *Query:* `?DMX,<addr>,<devcode>,<zone-num>;`
  *Long-ack Format*: `!OK,DMX,<addr>,<devcode>,<zone-num>;`
  *From: v02.00*
  *Reply: 'Report DMX zone health' and 'Report DMX zone level'*
  ```
  !DMXERR,<addr>,<devcode>,<zone-num>,<status-code>;
  !DMXLEVEL,<addr>,<devcode>,<zone-num>,<level>,<power>,<wattage>;
  ```
  e.g.
  ```
  ?dmx,1,15,2;
  !OK,DMX,001,015,002;<CR><LF>
  !DMXERR,001,015,002,000;<CR><LF>
  !DMXLEVEL,001,015,002,255,033,02500;<CR><LF>
  ```

## 7.4 Plate Button Control and Status

As said earlier, plate buttons are both inputs and outputs. They consist of a momentary contact switch input, and have colour LEDs and (for LCD plates) text that are outputs that can be controlled.

Note that the contact switch input and output colour & text are completely independent of each other. The outputs do not change automatically when an input occurs – a controller has to explicitly do this. Similarly, the outputs can be changed at any time regardless of if the input is changing.

Some of these button commands and queries can be applied to a single button or all buttons on a wall plate. To apply to all buttons substitute `ALL` for the button number parameter.

The `$BTNCOLR` and `$BTNTEXT` commands control the button outputs, and behave as you would expect. The `$BTNSTATE` command is slightly different as this relates to the contact switch input. It is used to *mimic* the behaviour of an *actual* button or if you want to *mirror* an actual wall *plate* via the gateway.

### 7.4.1 Set Button Colour

All plates have the same fixed set (or palette) of colours that buttons can be set to, and each colour is assigned a different number. This palette is different from and not to be confused with any palette used by DMX zones or Colour channels. The full button palette can be found in the Enumeration appendix.

- Set button colour
  *Command:* `$BTNCOLR,<addr>,<devcode>,ALL,<palette-colour>;` *(From: v02.00)*
  *Command:* `$BTNCOLR,<addr>,<devcode>,<btn-num>,<palette-colour>;`
  *Long-ack Format*: `!OK,BTNCOLR,<addr>,<devcode>,<btn-num>,<palette-colour>;`
  *From: v01.00*
  e.g.
  ```
  $BtnColr,08,2,4,3;
  !OK,BTNCOLR,008,002,004,003;<CR><LF>
  ```

### 7.4.2 Set Button Text

Set the text of a button. Limit the text to letter or number characters and spaces.

Although all plate buttons have a text property, only LCD plates can physically show the text and can only display a limited set of characters.

- Set button text
  *Command:* `$BTNTEXT,<addr>,<devcode>,ALL,<text>;` *(From: v02.00)*
  *Command:* `$BTNTEXT,<addr>,<devcode>,<btn-num>,<text>;`
  *Long-ack Format*: `!OK,BTNTEXT,<addr>,<devcode>,<btn-num>,<text>;`
  *From: v01.00*
  e.g.
  ```
  $btnText,63,1,all,;
  !OK,BTNTEXT,063,001,ALL,;<CR><LF>
  ```

### 7.4.3 Inject Button Input Event

This forces a button input event to occur in the eDIN+ system, that will cause any rules and actions connected to that button and event to be performed as though it came from the actual button itself.

This is primarily used to mirror a wall plate and its behaviour from the GATEWAY – see Volume 2 for further information on this.

Internal eDIN+ rules are connected to a specific button event. Each rule has only one button and one event, but actual buttons can have more than one rule connected to it. There are 5 possible button events: `Release-off`, `Press-on`, `Hold-on`, `Short-press` and `Hold-off` – see Enumeration appendix for numeric values.

To fully mimic a button so that all rules connected to the button are trigger appropriately, you should inject one of 2 sequences, depending on how long the button is pressed.
For a short press inject the sequence: when pressed `Press-on` then when released `Short-press`, `Release-off`
For a long held press inject the sequence: when pressed `Press-on` then `Hold-on` then when released `Hold-off`, `Release-off`

To trigger a single known event and its rules, simply inject this single event. It is not necessary to inject a full sequence.

- Inject button event
  *Command:* `$BTNSTATE,<addr>,<devcode>,<btn-num>,<event>;`
  *Long-ack Format*: `!OK,BTNSTATE,<addr>,<devcode>,<btn-num>,<event>;`
  *From: v02.00*
  e.g.
  ```
  $btnState,63,1,9,1;
  !OK,BTNSTATE,063,001,009,001;<CR><LF>
  ```

### 7.4.4 Request Button Status and related queries

The basic button query returns the health of the button as a status code and its current contact switch input state, as well as the current button colour and text. It is possible to query just button colour or text, and to query for a single button or all buttons on a wall plate.

A status code of `0` means the button is OK. Any other values indicates there is some problem with the channel. See Emulation Appendix for values and their meaning.

The input switch state is reported as either `0` (`Release-off`) or `1` (`Press-on`).

The palette-colour and text match the values sent in the 'Set Button …' commands.

<u>Summary</u>
`status-code` = `0` (OK). see Emulation Appendix for values
`state` = `0` (Release-off) or `1` (Press-on).
`palette-colour` = `0-255`. See Emulation Appendix for values
`text` = zero or more letters, number digits or spaces.

- request button status
  *Query:* `?BTN,<addr>,<devcode>,ALL;`
  *Query:* `?BTN,<addr>,<devcode>,<btn-num>;`
  *Long-ack Format*: `!OK,BTN,<addr>,<devcode>,<btn-num || ALL>;`
  *From: v02.00*
  *Reply:*
  *one or more 'Report button health'*
  *one or more 'Report button state'*
  *one or more 'Report button colour'*
  *one or more 'Report button text'*
```
!BTNERR,<addr>,<devcode>,<btn-num>,<status-code>;
!BTNSTATE,<addr>,<devcode>,<btn-num>,<state>;
!BTNCOLR,<addr>,<devcode>,<btn-num>,<palette-colour>;
!BTNTEXT,<addr>,<devcode>,<btn-num>,<text>;
```
  e.g.
```
?btn,15,1,2;
!OK,BTN,015,001,002;<CR><LF>
!BTNERR,015,001,002,000;<CR><LF>
!BTNSTATE,015,001,002,000;<CR><LF>
!BTNCOLR,015,001,002,005;<CR><LF>
!BTNTEXT,015,001,002,All Off;<CR><LF>
```

- request button colour
  *Query:* `?BTNCOLR,<addr>,<devcode>,ALL`
  *Query:* `?BTNCOLR,<addr>,<devcode>,<btn-num>;`
  *Long-ack Format*: `!OK,BTNCOLR,<addr>,<devcode>,<btn-num || ALL>;`
  *From: v01.00*
  *Reply: one or more 'Report button colour'*
```
!BTNCOLR,<addr>,<devcode>,<btn-num>,<palette-colour>;
```
  e.g.
```
?btncolr,15,1,all;
!OK,BTNCOLR,015,001,ALL;<CR><LF>
!BTNCOLR,015,001,001,003;<CR><LF>
```

```
!BTNCOLR,015,001,002,005;<CR><LF>
!BTNCOLR,015,001,003,005;<CR><LF>
...
```

- request button text
  *Query:* `?BTNTEXT,<addr>,<devcode>,ALL;`
  *Query:* `?BTNTEXT,<addr>,<devcode>,<btn-num>;`
  *Long-ack Format*: `!OK,BTNTEXT,<addr>,<devcode>,<btn-num || ALL>;`
  *From: v01.00*
  *Reply: one or more 'Report button text'*
  `!BTNTEXT,<addr>,<devcode>,<btn-num>,<text>;`
  e.g.
  ```
  ?btntext,15,1,2;
  !OK,BTNTEXT,015,001,002;<CR><LF>
  !BTNTEXT,015,001,002,All On;<CR><LF>
  ```

## 7.5 Colour and Tuneable White Control and Status

An output channel is determined by the configuration as a normal output, colour channel or a tuneable white channel. All output channels have a level or master brightness control, and this is controlled as described in the section on Output Control. Colour and Tuneable white channels have an additional control for the colour. These are described in this section.

Both `DMX` zone and DALI Broadcast `CHAN` channels can either be normal (White), Colour or Tuneable white channels, determined by the fixture type configured for the zone or channel.

Both Colour and Tuneable White channels can be set to static colours, and have a set of pre-set values that can be used to control the channel, each pre-set value has assigned its own number. This makes it easy to set a channel to a pre-set value that is e.g. a nominal red or blue without having to know the exact shade of red or blue.

Alternatively, there are situations where you want to adjust the colour of a channel to any possible value without being limited to the set of pre-defined pre-set values, e.g. adjusting the colour 'live' via a colour-wheel control. This is also possible:

- Colour channels are controlled with an `<rgb>` or `<wrgb>` (white-red-green-blue) value, using the same `#rrggbb` 6-hexdigit colour notation used in HTML/CSS, extending this to `#wwrrggbb` 8-hexdigit notation if a 4th white channel is used.
- Tuneable white channels are controlled with colour temperature values in Kelvin.

Colour channels can also be set to automatically play a predetermined colour sequence. Each sequence has its own number. Simply set the control to play this value to begin the sequence, or set the control to a static colour to stop the sequence playing.

There are 2 flavours of each command and query for the `CHAN` and `DMX` identifiers, otherwise the messages are the same.

See Enumeration Appendix for all pre-set and sequence values.

See Volume 2 for details on advanced control options and adjusting pre-set values via the GATEWAY.

### 7.5.1 Colour Control

Colour control commands have a crossfade time parameter. This specifies how fast the colour changes. For static colours this parameter defines how long it will take to change from its existing colour to the new colour. For sequences, this parameter specifies how long each step in the sequence takes. Crossfade times are specified in one-thousandths of a second or milli-seconds. This allows you to specify fractions of a second if required. Use a crossfade time of zero to snap the output to its new static colour. Colour sequences have a built-in limit on how fast they can change. A channel will default to this limit if you try and Play a sequence a crossfade time that is too small.

The colour or sequence is specified with a predefined number, or by a direct value.as described above. If you specify a nonsense value of a value the channel does not support no action will occur.

<u>Summary</u>
`preset` - static colours are `1-15`, `1-15` are determined by configuration.
`wrgb` - direct colour specified as a 6- or 8-hexdigit colour, e.g. red = `#FF0000`
`seq` - sequence number `64-68` (or `96-100`) – see Enumeration Appendix for details
`fadetime` – Crossfade time in milli-seconds.

- Set channel to pre-set static colour
  *Command:* `$DMXRGBCOLRFADE,<addr>,<devcode>,<zone-num>,<preset>,<fadetime(ms)>;`
  *Command:* `$CHANRGBCOLRFADE,<addr>,<devcode>,<chan-num>,<preset>,<fadetime(ms)>;`
  *Long-ack Format*: `!OK,DMXRGBCOLRFADE,…; or !OK,CHANRGBCOLRFADE,…;`
  *From: v02.01*
  e.g.
  ```
  $dmxRgbColrFade,02,15,03,2,1000;
  !OK,DMXRGBCOLRFADE,002,015,003,002,00001000;<CR><LF>
  ```

- Set channel to direct static colour
  This is the same command as 'Set channel to preset' except the direct wrgb value is substituted for the pre-set colour value.
  *Command:* `$DMXRGBCOLRFADE,<addr>,<devcode>,<zone-num>,#<wrgb>,<fadetime(ms)>;`
  *Command:* `$CHANRGBCOLRFADE,<addr>,<devcode>,<chan-num>,#<wrgb>,<fadetime(ms)>;`
  *Long-ack Format*: `!OK,DMXRGBCOLRFADE,…; or !OK,CHANRGBCOLRFADE,…;`
  *From: v02.01*
  e.g.
  ```
  $dmxRgbColrFade,02,15,03,#ff0000,1000;
  !OK,DMXRGBCOLRFADE,002,015,003,#FF0000,00001000;<CR><LF>
  ```

- Set channel to play sequence
  *Command:* `$DMXRGBPLAYFADE,<addr>,<devcode>,<zone-num>,<seq>,<fadetime(ms)>;`
  *Command:* `$CHANRGBPLAYFADE,<addr>,<devcode>,<chan-num>,<seq>,<fadetime(ms)>;`
  *Long-ack Format*: `!OK,DMXRGBPLAYFADE,…; or !OK,CHANRGBPLAYFADE,…;`
  *From: v02.01*
  e.g.

```
$dmxRgbPlayFade,02,15,3,64,1000;
!OK,DMXRGBPLAYFADE,002,015,003,064,00001000;<CR><LF>
```

### 7.5.2 Tuneable White Control

Tuneable white control commands have a crossfade time parameter. This specifies how it will take to change from its existing colour temperature to the new temperature. Crossfade times are specified in one-thousandths of a second or milli-seconds. This allows you to specify fractions of a second if required. Use a crossfade time of zero to snap the output to its new temperature.

The colour temperature is specified with a predefined number, or by a direct Kelvin value as described above. If you specify a nonsense value of a value the channel does not support no action will occur.

Summary
`preset` – pre-set colour temperatures are `48-63` and are defined by configuration.
`kelvin` - direct kelvin colour temperature typically `1800K` to `7000K`
`fadetime` – Crossfade time in milli-seconds.

- Set channel to pre-set colour temperature
  *Command:* `$DMXTWCOLRFADE,<addr>,<devcode>,<zone-num>,<preset>,<fadetime(ms)>;`
  *Command:* `$CHANTWCOLRFADE,<addr>,<devcode>,<chan-num>,<preset>,<fadetime(ms)>;`
  *Long-ack Format*: `!OK,DMXTWCOLRFADE,…; or !OK,CHANTWCOLRFADE,…;`
  *From: v02.01*
  e.g.
  ```
  $chanTWColrFade,02,18,03,50,3000;
  !OK,CHANTWCOLRFADE,002,018,003,050,00003000;<CR><LF>
  ```

- Set channel to direct kelvin colour temperature
  This is the same command as 'Set channel to preset' except the direct kelvin value is substituted for the pre-set value.
  *Command:* `$DMXTWCOLRFADE,<addr>,<devcode>,<zone-num>,#<kelvin>K,<fadetime(ms)>;`
  *Command:* `$CHANTWCOLRFADE,<addr>,<devcode>,<chan-num>,#<kelvin>K,<fadetime(ms)>;`
  *Long-ack Format*: `!OK,DMXTWCOLRFADE,…; or !OK,CHANTWCOLRFADE,…;`
  *From: v02.01*
  e.g.
  ```
  $chanTWColrFade,02,18,03,#2700K,3000;
  !OK,CHANTWCOLRFADE,002,018,003,#2700K,00003000;<CR><LF>
  ```

### 7.5.3 Request Colour Status

Determine the current colour of a colour channel.

The first method to determine the current colour of a colour channel is use the normal 'request Output status' query (as described in section on Output Status). As well as reporting the health and brightness level of the channel it will report the colour as well as a separate message.

It is also possible to request just the colour info with the 'Request Colour Status' query.

<u>Summary</u>
`preset` - static colours are `0-15`. `0` is black, `1-15` are determined by configuration.
`wrgb` - direct colour specified as a 6- or 8-hexdigit colour, e.g. red = `#FF0000`
`seq` - sequence number `64-68` (or `96-100`) -see Enumeration Appendix for details

- request output channel status
  *Query:* `?CHAN,<addr>,<devcode>,<chan-num>;`
  *Query:* `?DMX,<addr>,<devcode>,<zone-num>;`
  *Long-ack Format*: `!OK,CHAN,…` or `!OK,DMX,…`
  *From: v02.00*
  *Reply: 'Report of output status' followed by 'Report of colour status'*
  e.g.
```
?chan,1,18,2;
!OK,CHAN,001,018,002;<CR><LF>
!CHANERR,001,018,008,000;<CR><LF>
!CHANLEVEL,001,018,008,255,099,00100;<CR><LF>
!CHANRGBCOLR,001,018,002,000,#FF0000;<CR><LF>
```

- request colour status
  There are a number of flavours for the reply message. These are essentially the same message with a number value for pre-set and sequence settings.
  *Query:* `?CHANRGB,<addr>,<devcode>,<chan-num>;`
  *Query:* `?DMXRGB,<addr>,<devcode>,<zone-num>;`
  *Long-ack Format*: `!OK,CHANRGB,…` or `!OK,DMXRGB,…`
  *From: v02.00*
  *Reply: one of 'Report of colour status'*
```
!CHANRGBCOLR,<addr>,<devcode>,<chan-num>,<preset>,#<wrgb>;<CR><LF>
!CHANRGBPLAY,<addr>,<devcode>,<chan-num>,<seq>,#<wrgb>;<CR><LF>
!DMXRGBCOLR,<addr>,<devcode>,<zone-num>,<preset>,#<wrgb>;<CR><LF>
!DMXRGBPLAY,<addr>,<devcode>,<zone-num>,<seq>,#<wrgb>;<CR><LF>
```

  e.g.
```
?dmxRgb,2,15,3;
!OK,DMXRGB,002,015,003;<CR><LF>
!DMXRGBCOLR,002,015,003,005,#8054A2;<CR><LF>
```

### 7.5.4 Request Tuneable White Status
Determine the current colour temperature of a tuneable white channel.

The first method to determine the current temperature of a tuneable white channel is use the normal 'request Output status' query (as described in section on Output Status). As well as reporting the health and brightness level of the channel it will report the colour temperature as well as a separate message.

It is also possible to request just the temperature info with the 'Request colour temperature status' query.

<u>Summary</u>
`preset` – pre-set colour temperatures are `48-63` and are defined by configuration.
`kelvin` - direct kelvin colour temperature typically `1800K` to `7000K`

- request output channel status
  *Query:* `?CHAN,<addr>,<devcode>,<chan-num>;`
  *Query:* `?DMX,<addr>,<devcode>,<zone-num>;`
  *Long-ack Format*: `!OK,CHAN,…` or `!OK,DMX,…`
  *From: v02.01*
  *Reply: 'Report of output status' followed by 'Report of colour temperature status'*
  e.g.

```
?chan,1,18,2;
!OK,CHAN,001,018,002;<CR><LF>
!CHANERR,001,018,002,000;<CR><LF>
!CHANLEVEL,001,018,002,255,099,00100;<CR><LF>
!CHANTWCOLR,001,018,002,048,#3000K;<CR><LF>
```

- request colour temperature status
  There are a number of flavours for the reply message. These are essentially the same message depending on channel type.
  *Query:* `?CHANTW,<addr>,<devcode>,<chan-num>;`
  *Query:* `?DMXTW,<addr>,<devcode>,<zone-num>;`
  *Long-ack Format*: `!OK,CHANTW,…` or `!OK,DMXTW,…`
  *From: v02.01*
  *Reply: one of 'Report of tuneable white status'*

```
!CHANTWCOLR,<addr>,<devcode>,<chan-num>,<preset>,#<kelvin>K;<CR><LF>
!DMXTWCOLR,<addr>,<devcode>,<zone-num>,<preset>,#<kelvin>K;<CR><LF>
```

  e.g.

```
?dmxTW,2,15,3;
!OK,DMXTW,002,015,003;<CR><LF>
!DMXTWCOLR,002,015,003,051,#5000K;<CR><LF>
```

### 7.5.5 Channel Colour Discovery

There is no specific query to determine if a channel is a colour channel. However, the normal 'Request Output Status' query reports the colour or colour temperature status if the channel is a colour or tuneable white channel. This could be used to discover colour channels.

## 7.6 Input State

The main task on input channels is to determine and monitor the input state of the channel, and this is documented in this section.

It is also possible via the GATEWAY to control some types of input channel, and *inject* input channel events to get the eDIN+ system to mimic actual changes in input channels – however these are advanced topics documented in Volume 2.

### 7.6.1 Request Input Status

This query returns the health of the channel as a status code and its current state.

A status code of `0` means the channel is OK. Any other values indicates there is some problem with the channel. See Emulation Appendix for values and their meaning.

As noted earlier in the section of Different types of Channels, there are a number of different type of input channels, and the value of current state means different things for each of the different input types. To be able to know what type of input a channel is and what the returned state value means, there are different flavours of message for the different input types.

- `INPSTATE` contact-input
  `state` = `0` when Inactive (the 'Normally Closed' or 'Normally Open' state) or `1` when active.
- `INPPIR` pir-input
  `state` = `0` when sensor is reporting an Unoccupied or Empty state or `1` when sensor is reporting an Occupied state.
- `INPLEVEL` analogue-input
  `state` = calibrated input level, between its minimum and maximum calibrated values. For an uncalibrated channel this range is `0-255`.

It is not important to know the type of input a channel is to perform the query, as the query message uses the generic `INP` channel notation. The response message indicates the channel type if unknown.

▪ request input channel status
  *Query:* `?INP,<addr>,<devcode>,<chan-num>`
  *Long-ack Format*: `!OK,INP,000,000,000;`
  *From: v02.00*
  *Reply: 'Report input channel health' and one of 'Report input channel state*

```
!INPERR,<addr>,<devcode>,<chan-num>,<status-code>;
!INPSTATE,<addr>,<devcode>,<chan-num>,<status-code>,<state>;
!INPPIR,<addr>,<devcode>,<chan-num>,<status-code>,<state>;
!INPLEVEL,<addr>,<devcode>,<chan-num>,<status-code>,<state>;
```

  e.g.

```
?inp,15,21,1;
!OK,INP,015,021,001;<CR><LF>
!INPERR,015,021,001,000;<CR><LF>
!INPPIR,015,021,001,000;<CR><LF>
```

# A Enumerations

This section defines the values used by various commands and queries.

- Device Codes

| devCode | Product Code | Products |
|---|---|---|
| 01 | EVO-LCD-55 | LCD Wall Plate |
| 02 | EVO-SGP-xx | 2, 5 and 10 button Wall Plates,<br>Coolbrium & Icon plates |
| 04 | EVO-RP-03-02 | Evo 2-channel Relay Module |
| 08 | EVS-xxx | All Legacy Evo Slave Packs |
| 09 | EVO-INT-CI-xx | Evo 4 & 8 channel Contact Input modules |
| 12 | DIN-02-08 | eDIN 2A 8 channel leading edge dimmer module |
| 13 | DIN-03-04-TE | eDIN 3A 4 channel trailing edge dimmer module |
| 14 | DIN-03-04 | eDIN 3A 4 channel leading edge dimmer module |
| 15 | DIN-INT-00-08 | eDIN 8 channel IO module |
| 16 | DIN-RP-05-04 | eDIN 5A 4 channel relay module |
| 17 | DIN-UBC-01-05 | eDIN Universal Ballast Control module |
| 18 | DIN-DBM-00-08 | eDIN 8 channel Configurable Output module |
| 19 | DIN-DCM-xxx | All eDIN Dimmer Packs |
| 21 | DIN-RP-00-xx | All eDIN Rotary switch wall plates |
| 24 | ECO_MULTISENSOR | eDIN Multi-sensor (both Mk1 and Mk2) |
| 30 | MBUS-SPLIT | MBus splitter module |
| 144 | DIN-RP-05-04 | eDIN 5A 4 channel mains sync relay module |
| 145 | DIN-UBC-01-05 | eDIN Universal Ballast Control 2 module |

- Channel Status Codes

| | value | Error | Description |
|---|---|---|---|
| | 0 | Status Ok | No Errors |
| | 2 | Device missing | Device or Module is not responding to MBus messages. |
| | 3 | Channel Errors | Module has errors on at least one specific channel – see the individual channel for details of the error |
| | 4 | Bad Device Firmware | System is configured to use features that are not present in current module firmware. |
| | 5 | No AC | Module uses mains AC and it does not detect any main AC power |
| | 6 | Too Hot | The module has detected that its internal temperature is above its maximum rated operating temperature. |
| | 7 | Override Active | The channel has been manually set to override mode and is no longer controlled by the system |
| | 8 | Internal Failure | The channel or module has detected some sort of hardware failure internally |
| | 9 | DALI Fixture Errors | There are fixtures on this DALI *channel* that are reporting errors – see the individual DALI fixtures for details of the error |
| | 10 | Channel Load Failure | The module has detected there is a problem with the external load a channel is driving |
| | 20 | No DALI PSU | The module has detected that there is no PSU on its DALI bus. |
| | 21 | No DALI Commissioning Data | The DALI universe on this module does not contain any commissioning data. |
| | 22 | DALI Commissioning problem | The module has detected that the actual DALI fixtures detected do not match with the commissioning data |
| | 25 | DALI Lamp failure | A DALI fixture on this channel is indicating a lamp failure condition |
| | 26 | DALI missing ballast | A DALI fixture that is in the commissioning data is not present (is not responding). |

- eDIN Button Palette Colours

| | value | Name |
|---|---|---|
| | 0 | Black |
| | 1 | White |
| | 2 | Red |
| | 3 | Green |
| | 4 | Blue |
| | 5 | Orange |
| | 6 | Cyan |
| | 7 | Magenta |
| | 8 | Yellow |
| | 9 | DimWhite |
| | 10 | DimRed |
| | 11 | DimGreen |
| | 12 | DimBlue |
| | 13 | DimOrange |
| | 14 | DimCyan |

| | | |
|---|---|---|
| | 15 | DimMagenta |
| | 16 | DimYellow |

- eDIN Colour Palette and Modes

| | value | Mode |
|---|---|---|
| | 1-15 | Static Colour |
| | Palette code | Static wash using palette colour |
| | 64-68 | Solid Colour Sequence |
| | 64 | Long Rainbow Solid |
| | 65 | Short Rainbow Solid |
| | 66 | Hot Colours Solid |
| | 67 | Cold Colours Solid |
| | 68 | User Colours Solid |
| | 96-100 | Ripple Colour Sequence |
| | 96 | Long Rainbow Ripple |
| | 97 | Short Rainbow Ripple |
| | 98 | Hot Colours Ripple |
| | 99 | Cold Colours Ripple |
| | 100 | User Colours Ripple |

| | Palette code | Default Name |
|---|---|---|
| | 1 | Red |
| | 2 | Orange |
| | 3 | Yellow |
| | 4 | LawnGreen |
| | 5 | Green |
| | 6 | Mint |
| | 7 | Cyan |
| | 8 | DeepSkyBlue |
| | 9 | Blue |
| | 10 | Purple |
| | 11 | Magenta |
| | 12 | DeepPink |
| | 13 | User1 |
| | 14 | User2 |
| | 15 | User3 |

- eDIN Tuneable White Pre-sets

| | Preset value | Default Name | Default Temperature |
|---|---|---|---|
| | 48 | Candlelight | 1800K |
| | 49 | SoftWhite | 2700K |
| | 50 | WarmWhite | 3000K |
| | 51 | Whitelight | 3500K |
| | 52 | CoolWhite | 4000K |
| | 53 | BrightWhite | 5000K |
| | 54 | Daylight | 6000K |
| | 55 | BlueSky | 7000K |

- Button Events

| | value | state |
|---|---|---|
| | 0 | Release-off |
| | 1 | Press-on |
| | 2 | Hold-on |
| | 5 | Short-press |
| | 6 | Hold-off |