Title: Gateway Interface Volume 2 - Advanced

Version: 2.0.3

Author: Jeff Heine

Date: 02 March 2022

History

Date	Ver	Initials	Changes
24-May-19	2.0.0	Jph	Initial draft
6-Sep-19	2.0.1	Jph	Document: Admin rights for `$EVTADV` Clarify `wrgb` notation
14-Dec-21	2.02	Jph	Modification for Gateway Ver 2.01
28-Feb-22	2.03	Jph	Modification for Gateway Ver 2.02 Add DALI fixture health query Improve Advanced DALI addressing section Move back control section to volume 3

Table of Contents

1 Introduction

1.1 Purpose and scope

This document contains details for using advanced features of 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 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).

2 Overview of Advanced Features

This Volume 2 document describes advanced features of the eDIN+ GATEWAY interface. It builds on Volume 1 that describes the standard features, and assumes the reader is familiar with these standard features.

This document takes the different advanced features topic by topic, and gives a chapter to each. These chapters describe in detail the advanced commands & queries and events relevant to each topic. An overview of the topics is given in this chapter.

Those familiar with Volume 1 know that the GATEWAY interface is also organised in to 3 APIs: Interface, Channel and Scene & System. This chapter also gives a summary of the extensions to each API that are described in detail topic-by-topic.

2.1 Advanced Topics

2.1.1 Events

The eDIN+ system can generate event messages on a GATEWAY connection. This topic describes the event messages and discusses how to use them to monitor an eDIN+ system.

2.1.2 Users & Access

It is possible to set up user accounts in an eDIN+ system, that limit access to particular areas within the configuration. This topic describes the different types of access and how to access the eDIN+ system via the GATEWAY interface using different user accounts.

2.1.3 Advanced Scene Control & Status

Scenes are central to an eDIN+ system and the standard commands and queries described in Volume 1 provide full control and monitoring of the scenes. However there are some advanced features of scenes that can be accessed from the GATEWAY. This topic describes the full set of Scene control commands that mirror the operations available to the internal configuration. It also discusses some of the intricacies of when a scene state is active and inactive and explains more about the scene `mode` and `flags` status parameters.

2.1.4 Advanced Channel Control & Status

Channels are central to an eDIN+ system but come in a large variety of types. The eDIN+ system provides a set of standard commands and queries common to all channel types, but also has advanced features to access the differences. This topic covers some of this complexity.
It is possible to use the eDIN Relay module to control blinds and curtains, but requires specialised commands to do this. This topic explains how to do this via the GATEWAY.

The eDIN+ system works hard to integrate DALI fixtures in to a common channel framework, and hide the complexity of the DALI Bus operation. However, on sites

that are familiar with the DALI Standard and its operation, and use UBC modules, it is possible to access DALI fixtures on a UBC using the native DALI Broadcast, Group and Fixture addressing. This *native DALI addressing* is described in this topic.

The standard output status queries ignore the time taken in fading a channel. This usual is not a problem as fade times are usually short and effectively look instantaneous to an external system. However, in some situation very long fade times are used, and it is important to know the actual current channel level mid-fade. This topic describes the query to use.

Input channels are generally only used internally by the internal eDIN+ controller. However, there are times when it is desirable to either override the internal controller or provide the control function externally. This topic describes how to override or control the different types of input channels.

2.1.5 System Adjustments

Most of a system's features are set in the configuration, are fixed and cannot be changed by a normal user or via the GATEWAY. However, there are a few features that are allowed to be adjusted by the user, and these can also be done via the GATEWAY. Volume 1 describes simple live scene setting and Volume 3 describes advanced offline scene setting.

This topic describes the adjustment of pre-set values for colour and tuneable white channels. It also describes adjusting the value of different time settings for the system.

2.2 Advanced API extensions

This section lists the advanced messages and topics described in this volume by API, and cross references with the topics where they are detailed more fully.

2.2.1 Interface API

- Event management – see Chapter 3 Events
  `$EVENTS`, `$EVTSCN`, `$EVTERR`, `$EVTOUT`, `$EVTDIS`, `$EVTCIN`, `$EVTAIN`, `$EVTADV`
  `?EVENTS`, `?EVTSCN`, `?EVTERR`, `?EVTOUT`, `?EVTDIS`, `?EVTCIN`, `?EVTAIN`, `?EVTADV`
- Change User account – see Chapter 4 Users & Access
  `$USER`, `?USER`
- Change User password – see Chapter 4 Users & Access
  `$USERPSSWD`.
- `PUBLIC` and `ADMINISTRATOR` accounts – see Chapter 4 Users & Access

2.2.2 Scene API

- Advanced scene control – see Chapter 5 Advanced Scene Control & Status
  `$SCNFAST`, `$SCNTOGGLE`, `$SCNBACKON`, `$SCNRAISE`, `$SCNLOWER`, `$SCNSTOP`, `$SCNRAMP`, `$SCNNUDGEUP`, `$SCNNUDGEDN`
- Scene Events – see Chapter 3 Events
  `!SCNSTATE`,
  `!SCNRECALL`, `!SCNRECALLX`, `!SCNOFF`, `!SCNFAST`, `!SCNBACKON`, `!SCNRAISE`, `!SCNLOWER`, `!SCNRAMP`, `!SCNSTOP`, `!SCNNUDGEUP`, `!SCNNUDGEDN`, `!SCNONOFF`, `!SCNTOGGLE`, `!SCNSAVE`
- Scene status, mode and flags – see Chapter 5 Advanced Scene Control & Status

2.2.3 System API

- Adjusting timelines – see Chapter 7 System Adjustments
  tbc
- Controller inhibit – see Chapter 3 Events
  tbc

2.2.4 Channel API

- Advanced relay control – see Chapter 6 Advanced Channel Control & Status
  `$CHANPULSE`.
- Advanced DALI addressing – see Chapter 6 Advanced Channel Control & Status
  `BST`, `Gxx`, `Fxx`
- Live channel levels – see Chapter 6 Advanced Channel Control & Status
  `?CHANLEVELX`, `?DALILEVELX`, `?DMXLEVELX`
- Controlling input channels – see Chapter 6 Advanced Channel Control & Status
  `$INPSTATE`, `$INPPIR`, `$INPLEVEL`
- Adjusting colour pre-sets – see Chapter 7 System Adjustments
  `$RGBPRESET`, `$TWPRESET`
- Channel access – see Chapter 4 Users & Access
  `?CHANACCESS`, `?DALIACCESS`, `?DMXACCESS`, `?BTNACCESS`, `?INPACCESS`
- Channel Events – see Chapter 3 Events
  `!CHANFADE`, `!DALIFADE`, `!DMXFADE`, `!CHANSTOP`, `!DALISTOP`, `!DMXSTOP`,
  `!DMXRGBCOLR`, `!DMXRGBPLAY`, `!CHANRGBCOLR`, `!CHANRGBPLAY`,
  `!DMXTWCOLR`, `!CHANTWCOLR`,
  `!BTNCOLR`, `!BTNTEXT`, `!CHANPULSE`,
  `!BTNSTATE`, `!INPSTATE`, `!INPPIR`, `!INPLEVEL`,
  `!MODULEERR`, `!CHANERR`, `!DALIERR`, `!DMXERR`, `!BTNERR`, `!INPERR`

3 Events

Events are essentially messages that tell you when a change has occurred in the system, be it on a scene, input channel, output channel or wall plate button. They can also be in response to an explicit query for information.

The event messages broadly fit in to 4 categories:

a) Input channel events that can trigger actions
b) Scene and Channel operations that change the state of a scene or channel
c) Changes in scene and channel status, either caused by operations or because the health of a channel changes.
d) Result of a query.

Events may be used for:

a) Input events
   - used by your own controller to trigger your own rules.
b) & c) changes in scene and channel states
   - used to monitor and track changes in the eDIN+ system.
d) Result of a query.
   - both query responses and event messages are sent on the same connection and can be indistinguishable from one another, so the responses to queries must also be handled by your event handler.

Because you may or may not be interested in all messages, the Interface API allows you to configure a connection to send only events messages of certain classes, so that you do not have to handle and process a flood of unwanted event messages. However, response messages that are the result of a query are always sent and never filtered out.

GATEWAY events are split in to following classes. The classes are:

Flag	Name	Description
EVTSCN	Scene events	Scene commands and changes to scene level and status
EVTERR	Health Status events	Changes in channel & module health status
EVTOUT	Output events	Changes in output channels, including DALI channels and DMX zones
EVTDIS	Plate Display events	Changes to wall plate LED colour and text
EVTCIN	Contact Input events	All input events except Analogue inputs, including Contact inputs, PIR inputs and plate buttons.
EVTAIN	Analogue Input events	Change in Analogue Input channel level
EVTADV	Advanced events	Events associated with Advanced features

The EVTCIN and EVTAIN event classes are events on input channels and button switch inputs, and are most useful if you are providing your own controller. Changes in analogue inputs have been separated out from all other input events due to sheer number and frequency of this type of event. Analogue inputs are often used for light sensors so that lighting levels can be adjusted depending upon how much natural light

exists, in a technique often called *daylight harvesting*. As natural light can change continually, a lot of analogue input events can be generated.

The `EVTSCN` events are for change of scene state either level or activeness or both, and are most useful for monitoring the system state.

The `EVTOUT` and `EVTDIS` events are for change of state in both outputs including channels, DMX and buttons, and are most useful for monitoring the system state. Note: `EVTDIS` only includes events for button outputs; use `EVTCIN` for button input events.

The `EVTERR` events indicate changes of health on all hardware, and are useful if you want to monitor the health of the system.

The `EVTADV` events cover events generated by advanced techniques, and are only required if you are using these techniques.

The last section in this chapter describes how to override the in-built controller of an eDIN system but allow it to kick in if your controller fails.

3.1 Using Events

Volume 1 describes a number of ways to interact with the GATEWAY interface and an eDIN+ system including

iii) Control with asynchronous event-driven feedback

but the details of this method is left for this Volume 2.

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.

Control with asynchronous event-driven feedback allows the most integration between your system and the eDIN system, but your system must be able to be handle event-driven programming. This technique allows your system to respond instantly to changes in the eDIN+ system, and only respond and react when changes occur. If your system is naturally an event driven system, then events may be the more natural way to link with the eDIN+ system. However, this style of coding is harder to debug when problems arise.

You can only use events 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.*

No events are sent on a connection until the connection requests them. This avoids the connection being flooded with unwanted event messages. With this technique, it is necessary to send event filtering commands (see section below) to *turn events on every time you make a new connection*. It is possible to request all events or request only events in certain classes (see below).

In addition, as events are in general *changes* in state, it usually requires additional code to (synchronise to) determine the initial state of the eDIN+ system and maintain synchronisation when connections are lost and re-established. This means sending explicit queries to the GATEWAY interface to return the current state of items of interest when a connection is established.

3.2 Event filtering Commands and Queries

No events are sent on a connection until the connection requests them. This avoids the connection being flooded with unwanted event messages. It is necessary to send one or more of the event filtering commands when a new connection is established if events are wanted on a connection.

These commands and queries define or report the error filtering on a specific connection. Each connection has its own setting.

Events are split in to a number of classes as described above. It is possible to request all events or request only a certain class.

These commands and queries are part of the Interface API and can be sent at any time by all users to configure their connection.

3.2.1 Set Event Reporting for All events

This is a convenience message to turn reporting of events in all classes on a connection 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>
```

3.2.2 Query Event Reporting for All events

This is a convenience message to find out the current state of all event flags for a connection.

Query: `?EVENTS;`
Long-ack Format: `!OK,EVENTS;`
From: v01.00
Reply: set of Event flags
a list of all 'Report Event Flag' (same as individual requests)
`!EVTxxx,<on-off>;<CR><LF>`
e.g.

```
?eVents;
!OK,EVENTS;<CR><LF>
!EVTOUT,1;<CR><LF>
!EVTDIS,1;<CR><LF>
!EVTERR,0;<CR><LF>
!EVTSCN,1;<CR><LF>
!EVTCIN,0;<CR><LF>
!EVTAIN,0;<CR><LF>
!EVTADV,0;<CR><LF>
```

3.2.3 Set Individual Event Reporting

This collection of commands turns reporting of events of a class on a connection either on or off. Refer to the later sections in this chapter and elsewhere to determine which event flag controls which events.

Note: You can only set the advanced events flag using `$EVTADV` on connections with *administrator privileges* (see chapter on Users for further details). Otherwise the command is ignored. You do *not* need administrator privileges to query the state of `?EVTADV` event flag.

Long-ack Format: `!OK,EVTxxx,<on-off>;`
From: v01.00

Command: `$EVTSCN,<on-off>;`
Command: `$EVTERR,<on-off>;`
Command: `$EVTOUT,<on-off>;`
Command: `$EVTDIS,<on-off>;`
Command: `$EVTCIN,<on-off>;`
Command: `$EVTAIN,<on-off>;`
Command: `$EVTADV,<on-off>;`
`on-off` = zero to turn event flag off, non-zero to turn event flag on
e.g.

```
$EvtScn,1;
!OK,EVTSCN,1;<CR><LF>
$EvtOut,0;
!OK,EVTOUT,0;<CR><LF>
```

3.2.4 Query Individual Event Reporting

This is a collection of query messages to find out the current state of individual event flags for a connection.
Long-ack Format: `!OK,EVTxxx;`
From: v01.00

Query: `?EVTSCN;`
Reply: 'Report Scene Event Flag'
`!EVTSCN,<on-off>;<CR><LF>`
Query: `?EVTERR;`
Reply: 'Report Error Status Event Flag'
`!EVTERR,<on-off>;<CR><LF>`
Query: `?EVTOUT;`
Reply: 'Report Output Event Flag'
`!EVTOUT,<on-off>;<CR><LF>`
Query: `?EVTDIS;`
Reply: 'Report Display Event Flag'
`!EVTDIS,<on-off>;<CR><LF>`
Query: `?EVTCIN;`
Reply: 'Report Contact Input Event Flag'
`!EVTCIN,<on-off>;<CR><LF>`
Query: `?EVTAIN;`
Reply: 'Report Analogue Input Event Flag'
`!EVTAIN,<on-off>;<CR><LF>`
Query: `?EVTADV;`
Reply: 'Report Advanced Event Flag'
`!EVTADV,<on-off>;<CR><LF>`
e.g.
```
?EvtOut;
!OK,EVTOUT;<CR><LF>
!EVTOUT,1;<CR><LF>
```

3.3 Channel API events

Only channels and modules with View access flag set (see chapter on Users & Access) will have these events reported on the Gateway interface, and only if the connection has been asked for events.

For button events, the button's plate *module* must have its View access flag.

3.3.1 General Output Events

These are the result of commands to change the level or colour of output channels.

DALI events come in 2 flavours. One flavour uses the standard Mode DALI channel number notation. The other uses the advanced DALI addressing if advanced DALI operations are used – refer to the Advanced DALI section.

- *Event: 'Change in (output) channel level with fade'*
  Class: `EVTOUT`
  From: v01.00 (DMX from v02.00)

```
!CHANFADE,<addr>,<devcode>,<chan-num>,<level>,<fadetime(ms)>;
!DALIFADE,<addr>,<devcode>,<dali-id>,<level>,<fadetime(ms)>;
!DMXFADE,<addr>,<devcode>,<zone-num>,<level>,<fadetime(ms)>;
```

e.g.

```
!CHANFADE,002,021,006,255,00003000;<CR><LF>
!DALIFADE,002,017,016,255,00001000;<CR><LF>
!DALIFADE,002,017,016,BST,00001000;<CR><LF>
!DMXFADE,002,015,006,255,00005000;<CR><LF>
```

- *Event: 'Stop in (output) channel'*
  Class: `EVTOUT`
  From: v01.00 (DMX from v02.00)

```
!CHANSTOP,<addr>,<devcode>,<chan-num>;
!DALISTOP,<addr>,<devcode>,<dali-id>;
!DMXSTOP,<addr>,<devcode>,<zone-num>;
```

e.g.

```
!CHANSTOP,002,021,006;<CR><LF>
!DALISTOP,002,017,016;<CR><LF>
!DALISTOP,002,017,G01;<CR><LF>
!DMXSTOP,002,015,006;<CR><LF>
```

- *Event: 'Change in colour control'*

```
!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>
```

  Class: `EVTOUT`
  From: v02.01
  e.g.

```
!DMXRGBCOLR,002,015,006,001,#80FF00;<CR><LF>
!CHANRGBCOLR,002,015,006,000,#00FF00;<CR><LF>
```

- *Event: 'Change in tuneable white control'*

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

  Class: `EVTOUT`
  From: v02.01
  e.g.

```
!DMXTWCOLR,002,015,006,048,#4350K;<CR><LF>
!CHANTWCOLR,002,015,006,050,#3000K;<CR><LF>
```

- *Event: 'Pulse in (relay) channel state'*
  Class: `EVTOUT`
  From: v02.00

```
!CHANPULSE,<addr>,<devcode>,<chan-num>,<action>,<pulsetime(ms)>;
```

  e.g.

```
!CHANPULSE,001,012,002,001,001000;<CR><LF>
```

3.3.2 Button Output Events

These are the result of commands to change the level or text of a wall plate button.

- *Event: 'Change in Button colour'* and *'Change in Button text'*
  Class: `EVTDIS`
  From: v01.00

```
!BTNCOLR,<addr>,<devcode>,<btn-num>,<palette-colour>;
!BTNTEXT,<addr>,<devcode>,<btn-num>,<text>;
```
e.g.
```
!BTNCOLR,063,001,008,015;<CR><LF>
!BTNTEXT,063,001,008,All On;<CR><LF>
```

3.3.3 General Input Events

These are the result of natural changes of the level or state of input. There also the result of injecting inputs from the GATEWAY (see chapter on Advanced Input channel control)

- *Event: 'Change in Button switch state'*
  Class: `EVTCIN`
  From: v02.00
  ```
  !BTNSTATE,<addr>,<devcode>,<btn-num>,<new-state>;
  ```
  e.g.
  ```
  !BTNSTATE,063,001,008,000;<CR><LF>
  ```

- *Event: 'Change in PIR input state'*
  Class: `EVTCIN`
  From: v02.00
  ```
  !INPPIR,<addr>,<devcode>,<chan-num>,<new-state>;
  ```
  e.g.
  ```
  !INPPIR,005,021,008,000;<CR><LF>
  ```

- *Event: 'Change in switched input'*
  Class: `EVTCIN`
  From: v02.00
  ```
  !INPSTATE,<addr>,<devcode>,<chan-num>,<new-state>;
  ```
  e.g.
  ```
  !INPSTATE,005,021,008,000;<CR><LF>
  ```

3.3.4 Analogue Input Events

These are the result of natural changes of the level or state of input. There also the result of injecting inputs from the GATEWAY (see chapter on Advanced Input channel control)

- *Event: 'Change in analogue input'*
  Class: `EVTAIN`
  From: v02.00
  ```
  !INPLEVEL,<addr>,<devcode>,<chan-num>,<new-level>;
  ```
  e.g.
  ```
  !INPLEVEL,005,021,008,255;<CR><LF>
  ```

3.3.5 Channel Health Events

These are the result of natural changes of the status of channels or modules.

If configured to do so, a UBC can report problems with individual DALI fixtures. If there is a problem with a DALI fixture this is reported on both the Mode DALI channel the fixture is associated with and also on the fixture itself. A DALI fixture is identified

by an `F` followed by its DALI short address `0-63`, e.g. `F01` for fixture at DALI short address 1.

See volume 1 for a list of status codes.

Notes:

- If you are monitoring channel health through these event messages you need to track both relevant `!MODULEERR` messages as well as `!CHANERR` events and derivatives. This is because module errors are only reflected and reported in `!MODULEERR` event messages instead of `!CHANERR` events. This is to avoid a slew of `!CHANERR` and derivative messages being generated.
  This is different to the behaviour when polling individual channels using `?CHAN` and derivatives that do report module level error if they exist on that channel.
- When a module error occurs and generates a `!MODULEERR` event message, any current channel errors on that module are temporarily cleared down and the appropriate `!CHANERR` event is generated. This is to aid error tracking so that only 1 *live* error is reported for a channel either at the module level or channel level. When the module error disappears and reported with a `!MODULEERR` with status code 0, the channel error will be re-reported with a `!CHANERR` event message or derivative if the channel error is still present.

- *Event: 'Change in Module status'*
  Class: `EVTERR`
  From: v02.00
  `!MODULEERR,<addr>,<devcode>,<new status-code>;`
  e.g.
  `!MODUERR,002,021,001;<CR><LF>`

- *Event: 'Change in (output) channel status'*
  Class: `EVTERR`
  From: v02.00
  `!CHANERR,<addr>,<devcode>,<chan-num>,<new status-code>;`
  `!DALIERR,<addr>,<devcode>,<dali-num>,<new status-code>;`
  `!DALIERR,<addr>,<devcode>,<dali-fixture>,<new status-code>;`
  `!DMXERR,<addr>,<devcode>,<zone-num>,<new status-code>;`
  e.g.
  `!CHANERR,002,021,006,001;<CR><LF>`
  `!DALIERR,002,017,016,025;<CR><LF>`
  `!DALIERR,002,017,F16,025;<CR><LF>`
  `!DMXERR,002,015,001,002;<CR><LF>`

- *Event: 'Change in button status'*
  Class: `EVTERR`
  From: v02.00
  `!BTNERR,<addr>,<devcode>,<btn-num>,<new status-code>;`
  e.g.
  `!BTNERR,063,001,008,000;<CR><LF>`

- *Event: 'Change in input channel status'*
  Class: `EVTERR`
  From: v02.00
  `!INPERR,<addr>,<devcode>,<chan-num>,<new-status>;`

e.g.
`!INPERR,005,021,008,000;<CR><LF>`

3.4 Scene API events

Only scenes with View access flag set (see chapter on Users & Access) will have these events reported on the Gateway interface, and only if the connection has been asked for events.

3.4.1 Scene Status Events

The 'Change in scene state' event message is generated whenever there is a change in scene state, either due to a scene command directly on the scene, or due to other events occurring in the system.

Changes in `scn-state` are always instantaneous. Changes in `scn-level` occur as *fades* over a period of time. The event message reports how long it will take to reach its new level. If you are unconcerned about scene level you can ignore this value.

- *Event: Change in Scene State*
  Class: `EVTSCN`
  From: v02.00
  `!SCNSTATE,<scn-num>,<scn-state>,<scn-level>,<fadetime(ms)>;`
  `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)
  `fadetime` = time (in milliseconds) it will take to reach `scn-level`
  e.g.
  `!SCNSTATE,00001,1,255,00001000;<CR><LF>`

3.4.2 Scene Action Events

There are a number of event messages that report that a scene action or operation has occurred. These mirror the scene commands – refer to Scene Control sections in Volume 1 and 2. An event is generated whenever a scene action occurs either by a GATEWAY command or when the in-built controller responds to internal events.

- Report of Scene Action
  Class: `EVTSCN`
  All From: v01.00

Event: `!SCNRECALL,<scn-num>;`
Event: `!SCNRECALLX,<scn-num>,<level>,<fadetime(ms)>;`
Event: `!SCNOFF,<scn-num>;`
Event: `!SCNFAST,<scn-num>;`
Event: `!SCNBACKON,<scn-num>;`
Event: `!SCNRAISE,<scn-num>;`
Event: `!SCNLOWER,<scn-num>;`
Event: `!SCNRAMP,<scn-num>;`
Event: `!SCNSTOP,<scn-num>;`
Event: `!SCNNUDGEUP,<scn-num>;`
Event: `!SCNNUDGEDN,<scn-num>;`
Event: `!SCNONOFF,<scn-num>;`
Event: `!SCNTOGGLE,<scn-num>;`
Event: `!SCNSAVE,<scn-num>;`
e.g.

```
!SCNRECALL,00008;<CR><LF>
!SCNRECALLX,00008,255,00001000;<CR><LF>
!SCNOFF,00008;<CR><LF>
!SCNFAST,00008;<CR><LF>
!SCNBACKON,00008;<CR><LF>
!SCNRAISE,00008;<CR><LF>
!SCNLOWER,00008;<CR><LF>
!SCNRAMP,00008;<CR><LF>
!SCNSTOP,00008;<CR><LF>
!SCNNUDGEUP,00008;<CR><LF>
!SCNNUDGEDN,00008;<CR><LF>
!SCNONOFF,00008;<CR><LF>
!SCNTOGGLE,00008;<CR><LF>
!SCNSAVE,00008;<CR><LF>
```

3.5 Advanced events

Class: `EVTADV`
This class allows events associated with the following advanced features. Refer to the relevant volume and section for event details.

- Advanced DALI repair
  See Volume 3.

4 Users & Access

It is possible to set up user accounts in an eDIN+ system, that limit access to particular areas within the configuration. This chapter describes the different types of access and how to access the eDIN+ system via the GATEWAY interface using different user accounts.

Each scene, channel, DMX zones and button has inherent *access flags* that defines how they respond to gateway operations. These flags define if they can be queried, controlled and adjusted. These flags are set up in the configuration and *cannot* be changed via the GATEWAY interface. There are GATEWAY queries to return the access flags for an item.

User accounts are used to modify the access flags for each user. Scenes and channels can be put in an *area*, and then user accounts set up which *areas* they can access. A user may be given access to only certain areas. They may be allowed access to an area but not allowed to make adjustments in that area. In this way the access flags of channels and scenes are modified depending on the user.

Another way to think about user accounts is that each user account provides a different window on to the eDIN+ system. Some users may see the whole system, as if no user accounts where set up. Others will see a partial or reduced system.

It is important to realise that Access levels and User accounts involve minimal changes to the way the GATEWAY interface operates and is used. Each connection can be set to a particular user account. Otherwise operation is the same. All users use the same GATEWAY commands and queries to control and monitor the system. The only difference is that a particular scene or channel may exist for one user, but not another.

These topics are discussed further in the sections of this chapter.

4.1 Scene and Channel Access flags

For most of the time you do not need to be concerned about access levels and flags. A scene or channel will either be:

i) Private and fully hidden from the user. It effectively does not exist.

or

ii) Public – it exists and has no restriction.

A third case is

iii) Access-only – it exists and can be fully monitored and controlled, with standard queries and control commands.
But it cannot be adjusted and will not respond to these adjustment commands such as `$SCNSAVE`

This third case is only likely when operating with user accounts. If you do not already know from the config, you may want to inspect its access level only if you are thinking of adjusting it.

Each scene, channel, DMX zones and button has *access flags* that defines how they respond to gateway operations, and these are combined in to a single *access level* value. This access level is called the *inherent* access level.

There are 3 access flags: *View*, *Control* and *Edit*, and these flags are combined as a set of bit-field flags in to a number of *access level* values, as shown in the tables below.

Access Flag	Bit	Numeric Value	Description
`View`	0	1	If set the scene or channel can be queried.
`Control`	1	2	If set the scene or channel can be controlled with commands.
`Edit`	2	4	If set the scene can be edited / adjusted with scene setting operations, or aspects of the channel or module can be edited / adjusted, e.g. colour pre-set.

Access Level	Value	Description
`Private`	`0`	No access possible. It effectively does not exist.
`Monitor-only`	`1`	Can be monitored and queried but not controlled or adjusted.
`Access-only`	`3`	Can be monitored and controlled, but not adjusted. Will respond to standard queries and control commands.
`Public`	`7`	Fully accessible and adjustable. No restrictions.

For systems with an NPU the access level is set by the configuration and cannot be changes through the Gateway. For all other systems, access flags cannot be changed; all channels and scenes (that exist) are permanently fully accessible, i.e. everything has `public` access level.

4.1.1 Channel Access Queries

These queries can be used to establish if a channel will respond to a command or query.

Channels that do not exist, and channels that have a `Private` access level, will not reply to these queries, otherwise all channels will reply even if they are not set to reply to standard queries. Normally a channel will not respond to a query if the *View* flag is not set.

All the queries are the same except for the flavour of channel being queried. Also,

- Buttons reflect the access level of their Plate module. You can use `ALL` instead of `btn-num` to get the access levels for all buttons on a plate. You can determine the access level of a plate using the System discovery query `?PLATENAMES`.
- All flavours of Input channels use the same generic `INP` query.
- You can also use Advanced DALI addressing with the DALI query.

`access` = bit-field flags `1` (viewable) + `2` (controllable) + `4` (editable).

- request channel access
  All the queries are the same except for the flavour of channel being queries.
  From: v01.00
  Long-ack Format: `!OK,<xxx>ACCESS,…;`

  Query: `?CHANACCESS,<addr>,<devcode>,<chan-num>;`
  Reply: 'Report channel access'
  `!CHANACCESS,<addr>,<devcode>,<chan-num>,<access>;`

  Query: `?DALIACCESS,<addr>,<devcode>,<dali-id>;`
  Reply: 'Report DALI channel access'
  `!DALIACCESS,<addr>,<devcode>,<dali-id>,<access>;`

  Query: `?DMXACCESS,<addr>,<devcode>,<zone-num>;`
  Reply: 'Report DMX zone access'
  `!DMXACCESS,<addr>,<devcode>,<zone-num>,<access>;`

  Query: `?BTNACCESS,<addr>,<devcode>,<btn-num>;`
  Query: `?BTNACCESS,<addr>,<devcode>,ALL;`
  Reply: zero or more 'Report button access'
  `!BTNACCESS,<addr>,<devcode>,<btn-num>,<access>;`

  Query: `?INPACCESS,<addr>,<devcode>,<chan-num>;`
  Reply: 'Report channel access'
  `!INPACCESS,<addr>,<devcode>,<chan-num>,<access>;`

  e.g.
  `?chanAccess,1,12,2;`
  `!OK,CHANACCESS,001,012,002;<CR><LF>`
  `!CHANACCESS,001,012,002,07;<CR><LF>`

4.1.2 Scene Access Queries

There are no separate queries to request a scene's access level. The access level is returned from a normal `?SCN` or `?SCNS` scene status query and in Scene status events.

4.2 User accounts and Area Access

As said earlier, each user account provides a different window on to the eDIN+ system. Some users may see the whole system, as if no user accounts were set up. Others will see a partial or reduced system.

User accounts are based around *areas*. Scenes and channels are put in to areas, and user accounts define if they can access the scenes and channels in that area, via *area access levels*.

These area access levels modify the *inherent* access level of scenes and channels resulting in a *user access level*. It is this user access level that the GATEWAY uses when providing access to a scene or channel command or query.

There are some special user accounts. There is a default `Public` *guest* account that provides a minimum level of access for all users. Some accounts can be made *administrator* accounts as a shorth-hand for giving unrestricted `public` access to *all* areas.

4.2.1 Area access levels

For each account, an *area access level* is configurated *for every area* in the eDIN+ system. The area is either:

- `Private` - inaccessible and hidden,
- `Access-only` - allowing only monitoring and control, or
- `Public` - unrestricted allow monitoring, control and adjustments.

The area access level for every area is returned in the responses to the System Discovery query `?AREANAMES`.

User accounts and area access levels can only be created and set during configuration and cannot be created or modified via the GATEWAY interface, (except for passwords).

4.2.2 User Access levels

In addition to area access levels and user accounts, every scene and channel is assigned to an area and also has an inherent access level. When accessed with a user account, a scene or channel's *inherent access level* is combined with its *area access level* to produce a final *user access level*, that determines how the scene or channel will respond to the command or query. (see section below for details of this).

All scenes and channels in an eDIN+ system are in an *area*, and only one area. There is always one area in a system called the Default area (area number 0). Other areas can be created. Scenes and channels are always put in the Default area when created and can be configured in to other areas as desired.

So how are scene and channel access flags modified by user accounts? The details are a bit involved but the concepts are quite simple.

- Firstly accounts can only restrict *inherent* access – they cannot increase or override a scene or channels *inherent* access level. If a scene or channel has been made invisible via the configuration, no user account can make it visible (including Administrator accounts).
- Secondly, all accounts implicitly inherit the access of the Public account. The Public account provides a default minimum level of access that all other accounts build on. In other words, if you have access to a scene or channel from the Public account you also have access to it no matter which account you are using and no matter what area access levels have been set in that account.

The detailed calculation involves bit-wise AND and OR operators on the *inherent* access level and appropriate *area* access level.

```
user-access = (inherent-access && public-area-access)
            ||(inherent-access && account-area-access);
```

In layman's language, a channel or scene is *viewable*, if the channel or scene is *inherently* viewable *AND* it is in an *area* that is viewable. This is determined for both the Public account and the actual user account. The end result is that the channel or scene is viewable if it is viewable from either Public account *OR* user account. This is done for each access flag: *view*, *control* and *edit*.

The area of a scene or channel or its inherent access level cannot be changed via the GATEWAY interface.

4.2.3 Special Public account

There is always a special user account called `Public`. This is the guest account and has no password or credentials. You cannot create or delete this account.

It is used
- as the default account when a connection has not been set to use any other user account.
- to log out a connection of another account
- when you try to log a connection in to a different user account but fail

If an eDIN+ has user accounts then the area access levels in the Public account can be configured - by default all areas are hidden, (i.e. no access to anything). With no user accounts, the Public account automatic has full access to everything.

4.2.4 Administrator Privileges and Special Administrator account

Normal user accounts can be made an *administrator* account that gives the account *administrator privileges*. The *administrator privileges* make all areas public and completely unrestricted. Not only is this a short-hand way to define this unrestricted access, but also some eDIN+ features and operations are only available to *administrator* accounts.

For convenience, when user accounts are set up in an eDIN+ system an *administrator* account is always set up, called `Administrator`, so there is always at least one account with administrator privileges with unrestricted access to everything. This account can be used as a normal account, but can also be used when access is needed by an engineer to resolve or repair a system.

4.3 Using User accounts with Connections

As said earlier, it is important to realise that *access levels* and *user accounts* involve minimal changes to the way the GATEWAY interface operates and is used. Other than setting a user for each connection, all other operations is the same. All users use the same GATEWAY commands and queries to control and monitor the system. The only difference is that a particular scene or channel may exist for one user, but not another.

It is also possible to use the GATEWAY interface without worrying about users. There is a special `Public` *guest* account that is automatically used on connections until another user is set. For eDIN+ systems without user accounts, the `Public` user always has unrestricted public access to everything, so no (other) user is needed. For eDIN+

systems with user accounts, the `Public` account access is configurable, so it is up to each system to decide how much default access to provide.

User accounts are used on connections. You can set the user on every connection independently and each connection has its own window on the eDIN+ system depending on its current user.

A user is set on a connection using the user's *credentials*. There are different mechanisms for doing this for the different types of connection. All connections can change a user with commands. In addition, HTTP connections can use the HTTP Basic Authentication mechanism, via HTTP headers.

As user accounts present different windows of an eDIN+ system, the available scenes and channels may change when you change user. When you change users, you should always invalidate any data held for the current user and refresh any data you hold for the new user.

4.3.1 The Public guest account

As said earlier, the special `Public` *guest* account that is automatically used on connections until another user is set. It has no password or credentials so does not need any special action or authentication to be used.

In fact, the `Public` account is automatically used and when no other user is set. For example:

- is the default account when a new connection is made and no credentials have been presented.
- is the account when a connection presents incorrect credentials, i.e. tries to change a user but fails.
- is the account when a connection explicitly cancels a user.
- is the account when a connection explicitly changes to the `Public` user.

4.3.2 User Account commands and queries

To change a user on a connection you must send the user's *credentials* using the `$USER` command.

A user's credentials consist of a `user-name` and `password`. These are set up in the configuration. It is also possible to change a password via the GATEWAY. Both `user-name` and `password` are plain text and are case-sensitive. The credentials must match exactly those set up to be successful.

A user can change its own password using the `$USERPSSWD` command. The connection must already be successfully set for the user using *current credentials*. The connection must present both current and new passwords. The current user remains unchanged, but if successful, then all future connections will use the new password for *user credentials*. It is not possible to query the password.

These commands and queries are part of the Interface API and can be used on all types of connection. As HTTP connections are short-lived you have to present user

credentials with every access you make for. You can use the `$USER` command, but it may be more convenient to use the HTTP basic authorization mechanism instead.

As user accounts present different windows of an eDIN+ system, the available scenes and channels may change when you change user. When you change users, and if you have any local (cached) data any available scenes, areas or channels, then you should always delete this data and refresh it for the new user. Although you can change the user at any time, it is usual to only do this when you make a new connection - when you change the user, close down the existing connection and open a new one. (HTTP connections work differently – see below.)

Summary

`user-name` – plain text, case sensitive name of user account
`password` – unencrypted plain text, case sensitive password for account
`status-code` – `0` = successful operation, `1` = operation failed

- Change to a particular user account
  Command: `$USER,<user-name>,<password>;`
  Long-ack Format: `!OK,USER,<user-name>,<password>;`
  From: 02.00
  Reply: 'Report Change of User'
  `!USERERR,<user-name>,<status-code>;<CR><LF>`
  e.g.
  ```
  $User,William,Greensleeves;
  !OK,USER,William,Greensleeves;<CR><LF>
  !USERERR,William,0;<CR><LF>
  ```

- Cancel current user account; log off
  This command changes the user to `Public`
  Command: `$USER;`
  Long-ack Format: `!OK,USER;`
  From: 02.00
  Reply: 'Report Change of User'
  `!USERERR,<user-name>,<status-code>;<CR><LF>`
  e.g.
  ```
  $user;
  !OK,USER;<CR><LF>
  !USERERR,Public,0;<CR><LF>
  ```

- Change current user password
  Command: `$USERPSSWD,<user-name>,<old-password>,<new-password>;`
  Long-ack Format: `!OK,USERPSSWD,<user-name>,<old-password>,<new-password>;`
  From: 02.00
  Reply: 'Report Change of Password'
  `!USERPSSWDERR,<user-name>,<status-code>;<CR><LF>`
  e.g.
  ```
  $userPsswd,William,Greensleeves,Shakespear;
  !OK,USERPSSWD,William,Greensleeves,Shakespear;<CR><LF>
  !USERPSSWDERR,William,0;<CR><LF>
  ```

- Query current user account
  Query: `?USER;`
  Long-ack Format: `!OK,USER,<user-name>;<CR><LF>`
  From: 01.00
  Reply: 'Report of User'
  `!USER,<user-name>;<CR><LF>$user;`
  `!OK,USER;<CR><LF>`
  `!USER,Public;<CR><LF>`

4.3.3 Using HTTP Basic Access Authentication with User accounts

HTTP access does not support long-lived connections. Each access is effectively a new connection that closes at the end of the access. This means that user credentials need to be presented with each access. It is possible to include the `$USER` command as the first command in each access but the HTTP standard provides another mechanism that may be more convenient.

The HTTP standard has a way to present user credentials as part of any HTTP transaction via a mechanism called Basic Access Authentication.

This may already be supported in your client-side. For example, the jQuery AJAX methods allow you to provide username and password parameters directly.
Example in jQuery:

```
$.ajax({
      type: "POST",
      url: 'http://'+ipaddr+'/gateway?',
      username: "William",
      password: "Greensleeves",
      ...
    })
```

If your client-side does not directly support Basic Access Authentication, you can simply add an authentication header to your HTTP request. In basic HTTP authentication, a request contains a header field of the form `Authorization: Basic <credentials>`, where `credentials` is the base64 encoding of username and password joined by a single colon (:)
(Reference: //en.wikipedia.org/wiki/Basic_access_authentication)
Example in jQuery:

```
$.ajax({
      type: "POST",
      url: 'http://'+ipaddr+'/gateway?',
      headers: {
        "Authorization": "Basic "+btoa("William:Greensleeves")
      },
      ...
    })
```

5 Advanced Scene Control & Status

Scenes are central to an eDIN+ system and the standard commands and queries described in Volume 1 provide full control and monitoring of the scenes. However there are some advanced features of scenes that can be accessed from the GATEWAY.

This chapter explains more about scene status. It describes some of the intricacies of when a scene state is active and inactive and explains more about the scene `mode` and `flags` status parameters.

This chapter describes the full set of Scene control commands. Volume 1 describes the standard commands that provide complete scene control. You only need to use the advanced scene control commands if you are fully conversant with the scene operations available in an eDIN+ configuration and you want to mimic that exact behaviour from your controller.

5.1 Scene status explained

Each scene in an eDIN+ system has a *scene state* that says whether it is *in use* or not; the state can be *active* or *inactive*. The eDIN+ system displays this state on plate buttons and elsewhere.

There is no agreed standard of when a scene is *in use* and *active* and when it is no longer in use and *inactive*. Often it is straight-forward if all scenes *completely overlap*, i.e. every scene contains exactly the same channels. If all scenes in an area control the same channels but just set them to different brightness's, then a scene is in use and becomes *active* when you recall that scene and becomes *inactive* when you recall another scene.

It is straightforward if scenes do not *overlap* at all. If there are 2 scenes with completely different channels and no channel is in both scenes, then the scenes are completely independent of one another and do not affect each other's state.

However, things are more difficult if scenes *partially overlap*, i.e. there are a few, but not all, channels in both scenes. If you recall one scene, then recall the second so that some of the channels change brightness, but others are left alone, is the original scene still in use and active? Often it depends on what the channels are, how many are overlapping and what the intention of both scenes are. Often it depends on the end-user – different people expect different things in the same specific situation.

Suffice to say getting scene state to behave exactly as you want can get very difficult and complicated. It is possible to adjust the algorithm the eDIN+ system uses to calculate scene state. It can use *scene mode* or *channel mode*, and it can use *strict* or non-strict rules. This is explained below.

5.1.1 Scene Mode and Flags values

The 'Report scene status' message includes a `mode` and a `flags` value.
`mode` = `1` - *scene* monitor mode, `2` - *channel* monitor mode
`flags` = bit-field flags `1` (is-an-off-scene) + `2` (strict rule applied)

5.1.2 The Is-An-Off-Scene flag

Sometimes it is useful to know if a scene is considered an *off scene* by the system, as these are treated slightly differently by the system. An *off scene* is a scene that turns all the channels in the scene off, i.e. to a level of zero.

The *state* and *level* of an *off scene* behaves in exactly the same way as normal scenes, but the control of its channels is opposite:

- The scene *state* is *active* when the scene is recalled e.g. `$SCNRECALL`, `$SCNBACKON`, and *inactive* when recalled to off, e.g. `$SCNOFF`.
- The scene *level* is `0` when *inactive*, and non-zero (usually `255`) when *active*.
- But the *channel levels* are turned *off* when the scene is recalled and *on* when recalled to off.

If you are monitoring both scenes and levels, *off scenes* can be a little disconcerting as scene state and channel levels behave 'out of sync'
- channels are switched *on* with `!SCNOFF` events and switch *off* with `!SCNBACKON`.
- channels are *on* when scene level is `0`, and *off* when scene level is `255`.

If you are using toggle commands, then remember these work on *scene* state *not channel* levels. That is why the `$SCNBACKON` command turns the *off scene* back on, but turns the *channel* levels off.

5.1.3 Scene monitor mode

The default scene behaviour is *scene monitor mode* with *strict* rules. In *scene monitor mode* scene operations on a scene will make the scene active. Operations on other *overlapping* scenes may make the scene inactive. This works in most situations, but it is possible to configure the *strict* rule to off that changes when the scene becomes inactive. Here are a few situations when you may want to do this.

- If you have a main scene with a number of channels, but one channel controls a lighting feature that also has its own individual scene control as well. If you want the main scene to ignore the state of the feature channel, then turn *off* the *strict* rule.
- If you have a large hall or space that has a main area but also lots of smaller areas within it, and each smaller area has its own scenes that control the lights in their area. You have an 'All on' scene that turns the lights on all areas on, including the lights in both the main area and the smaller areas, and you want the 'All on' scene to remain active as long as the main area lights are on, even if the lights in the smaller areas have changed. Turn *off* the *strict* rule for the 'All on' scene to ignore the state of the smaller areas.

5.1.4 Channel monitor mode

The system calculates the scene state with channel on/off states no matter which scene or scenes got them to that state.

Channel monitor mode is most useful for *off scenes* and scenes that control feature channels. It does not work particularly well for normal scenes.

The *strict* rule flag says that *all* channel *states* must *match* the those defined in the scene for the scene to be *active*. Turning the strict rule *off* says that *any* channels must

match. For normal scenes this means that the channel must be *on* to match. If the channel level is defined as *off* in the scene, then the channel must be *off* to match. Note: the *matching* works on whether a channel is simply *on* or *off*. It does *not* try to match the exact level of the channel with that defined in the scene. This is why it works well for off scenes and feature channel scenes.

5.2 Advanced Scene Control

There are a number of scene commands that mirror the commands available in an eDIN+ configuration. The standard commands described Volume 1 provide complete scene control. You only need to use the advanced scene control commands if you are fully conversant with the scene operations available in an eDIN+ configuration and you want to mimic that exact behaviour from your controller.

5.2.1 Summary of Standard Scene Commands

The standard commands allow you to turn a scene on or off with:

- `$SCNRECALL` and `$SCNOFF`

The standard commands allow you to toggle a scene on and off with:

- `$SCNONOFF`

The standard commands allow you to fully control the scene level, including turning the scene on & off, and with a controllable fade time with:

- `$SCNRECALLX`

5.2.2 Advanced Scene Recall Commands

- Recall scene with *fast* recall time action.
  This ignores the scene's fade time and recalls the scene at maximum level using the *fast* fade time. The *fast* fade time is defined in the configuration as a global setting. This command is used to immediately turn a scene on that has a very long fade time. For example, a scene may be set to fade slowly over half an hour, but occasionally you need the lights to come on straight away, i.e. fade to max over a second or so.

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

5.2.3 Advanced Scene Toggle Commands

Toggle commands toggle the scenes state from on to off or vice versa. The standard toggle command is `$SCNONOFF` that is simple and straightforward to use. There is an advanced *scene toggle* action that has a number of pitfalls that makes it less useful.

The advanced `$SCNTOGGLE` command toggles a scene on and off, but uses a *remembered snap-shot* of scene and channel levels when turning the scene back on, instead of always recalling the scene back on to maximum.

This command is useful internal to an eDIN+ system when needing to provide full on, off and level up & down control of e.g. a channel from a single button, and can put

restrictions and limits around this operation. It is less useful when level control can be provided via e.g. a slider that is linked directly with `$SCNRECALLX` operations.

The problem with this command is that it can *remember* very dim levels, levels that are officially on, but look off because they are so dim. It then looks like this command is not doing anything as it toggles between off and almost (very dim) off. The only way to recover is to use another command to restore levels to something brighter.

There is a partner command that provides half of the toggle action, to get a scene back on.

- Recall scene with toggle action.
  (`$SCNTOGGLE` toggles between `$SCNBACKON` and `$SCNOFF` actions).
  This command is similar to `$ScnOnOff` but saves and restores scene and channel levels. There are a number of things to be aware of:
  - There is only one snap-shot per scene, held by the scene. This snap shot is held temporarily. It *does not* change the levels defined in the scene, i.e. it is *not* a scene-setting action. It is forgotten when the controller is powered down or reset.
  - Any/all `$SCNOFF` actions saves a snap-shot of the current scene and channel levels before they are turned off, whether implicit or explicit in other commands, not just this one.
  - It does not matter how the scene and channels have got to their current level. What is important is their current level at the time the `$SCNOFF` action occurs.
  - Scene and channel levels are only saved if they are currently non-zero, i.e. you cannot save an 'off' level.
  - However, levels will be saved even if at very low levels that physically look like they are off, resulting in a confusing behaviour of toggling between actual off and 'look-a-like-off' that looks like nothing is happening.

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

- Recall scene with previous (toggle) level.
  This is a scene recall action, but recalls the scene and channels to their previously saved level. It is half of the `$SCNTOGGLE` action that turns a scene on. This will recall the scene & levels to their saved levels even if the scene is already active. Refer to `$SCNTOGGLE` for further information.

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

5.2.4 Advanced Scene Level Adjustment Commands

These commands adjust the scene level and the values of the channels and other objects defined in the scene. For the most part, these values are simple channel levels, but a scene may contain other objects that are invisible to the GATEWAY interface, but are still affected by these commands. In other words these commands may have unintended side effects. This will depend on how the configuration is built.

Note, also, that the effect that these commands have on scene state (whether it is active or inactive) is also complicated and also depends on how the scene is configured.

If you simply want to change channel & scene levels, then use the `SCNRECALLX` command. If you want to mimic the action of a plate button, then these commands may be useful.

- Raise scene action – start dimming up.
  This starts a dim-up operation:
  - Channels defined in the scene will start to increase their channel levels to their maximum.
  - Tuneable-white channels will start to increase their colour-temperature to their maximum.
  - Colour channels will start to move their colour forward through their colour palette in a circular un-ending sequence.
  - Channels that are already off (level=0) are unaffected.
  - The rate that the levels increase is set by a global configuration setting.

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

- Lower scene action– start dimming down
  This starts a dim-down operation:
  - Channels defined in the scene will start to decrease their channel levels, to a minimum level that is set by a global configuration setting.
  - Tuneable-white channels will start to decrease their colour-temperature to their minimum.
  - Colour channels will start to move their colour backwards through their colour palette in a circular un-ending sequence.
  - Channels that are off (level=0) are unaffected.
  - The rate that the levels decrease is set by the global configuration setting that determines the SCNRAISE rate.

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

- Stop scene dimming - stop raise, lower and ramp actions
  This stops any dimming action on all channels defined in the scene.

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

- Ramp scene action – start dimming. Toggle between Raise and Lower actions
  This command will alternate between performing a `SCNLOWER` action and a `SCNRAISE` action, depending on the current scene state.

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

- Nudge scene up action - raise scene level by small amount
  This is similar to `$SCNRAISE` but dims-up by a fixed step size and stops:
  - Channels defined in the scene will increase their channel levels by the step size to their maximum.
  - Tuneable-white channels will move their colour-temperature forward to the next preset value.
  - Colour channels will move their colour forward to the next palette entry.
  - Channels that are already off (level=0) are unaffected.
  - The step size is defined by a global configuration setting
  - The rate that the levels increase is set by a global configuration setting.

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

- Nudge scene down action - lower scene level by small amount
  This is similar to `$SCNLOWER` but dims-down by a fixed step size and stops:
  - Channels defined in the scene will decrease their channel levels by the step size, but are cupped to a minimum level that is set by the global configuration setting that sets the `SCNLOWER` minimum.
  - Tuneable-white channels will move their colour-temperature backwards to the next preset value.
  - Colour channels will move their colour backwards to the previous palette entry.
  - Channels that are already off (level=0) are unaffected.
  - The step size and rate of change uses the same global configuration settings as `SCNNUDGEUP`.

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

6 Advanced Channel Control & Status

6.1 Advance Relay Module Control

To turn a channel on and off you generally use the `$CHANFADE` command, for both dimmable and relay channels, and this works well for mains-voltage relay control.

However, the eDIN+ modules `EVO-RP-03-02` and `DIN-RP-05-04` are able to be used for low-voltage control such as blind control. Often this control requires quick pulses to be sent that are difficult to achieve using a sequence of `$CHANFADE` commands. So instead of sending channel levels to the relay channel, it is possible to send relay commands using the `$CHANPULSE` command.

Possible relay channel actions are shown in the table below. Note although the terms `Open` and `Close` are used and are correct for *normally open (NO)* relay contacts, it is possible some relays channels to be set to work with *normally closed (NC)* relays, when the terms `activate` and `deactivate` are more accurate.

value	name	action
1	`Pulse Close`	Momentarily activate relay contact
2	`Pulse Open`	Momentarily deactivate relay contact
3	`Pulse Toggle`	Momentarily toggle relay contact

6.1.1 Advanced Output State Commands

- Send relay channel action
  Command: `$CHANPULSE,<addr>,<devcode>,<chan-num>,<action>;`
  Long-ack Format: `!OK,CHANSTATE,<addr>,<devcode>,<chan-num>,<action>,<pulsetime(ms)>;`
  From: v02.00
  e.g.
  ```
  $ChanPulse,1,12,2,1,1000;
  !OK,CHANPULSE,001,012,002,001,001000;<CR><LF>
  ```

6.2 Live Channel Levels

There are two techniques for determining the level of a channel. You can either poll for the current level using a query, or you can enable channel events on a connection and monitor these events.

Polling is useful for obtaining the level on demand at irregular and infrequent intervals, and is the simpler technique to implement. But with polling you are only sampling the level at instances, not monitoring it continually, so the values returned are not 100% representative of a channel's level and this might cause problems for you. If you want to continually monitor and track a channel's level we recommend using events, as the event tells you the length of fade operations. (We also do not recommend *frequent* regular polling over the GATEWAY, i.e. polling quicker that 5 minute intervals).

When polling for a channel's level, there are 2 sets of queries available. Although both return the level of the channel, there are querying slightly different things.

The normal channel query ?CHAN and its derivatives report a channel's level but always 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 set of queries are described in Volume 1.

This set of normal queries works best when channel levels are mostly unchanging and change relatively quickly when they do change. For example, if your channels fade over 3 seconds, but you are querying every 5 minutes, you are never going to be able to track fade operations accurately. Each channel is going to be spending most of its time statically at the level of the last fade operation, so by ignoring on going fade operations, the value returned is the value that the channel will be at for most of its time until the next query, and usually what makes most sense.

The advanced channel query ?CHANLEVELX and its derivatives do not ignore on-going fade operations, and always returns the value of the level at the instance the query is made. This set of queries is described in this Volume 2 below. Use this set of queries if the channel has fade operations over long fade times. For example, if a channel is fading over 20 minutes and you are querying every 5 minutes, then returning the actual level mid-fade usually makes most sense.

However using the set of advanced ?CHANLEVELX queries with short fade times can give a mis-leading level if it coincides with a short fade. The level may not be near to this value for any length of time and so the value reported may be an inaccurate representation of a channel's level.

If a channel has both long and short fade operations, and it is important not to have inaccurate values, then it is recommended not to use polling but to tracking the level using events, as the event tells you the length of a fade operation.

To track channel levels with events, enable the channel events using the $EVTOUT or $EVENTS command, and monitor the !CHANFADE and !CHANSTOP events and their derivatives – these are described in the Events chapter in this Volume 2.

6.2.1 Advanced Request Channel Level

The advanced channel level requests follow the same format as the normal Request Channel Status queries, except they only return a message for the channel's level and the value is the instantaneous level even during active fade operations.

There are three flavours of Query messages for the three CHAN, DALI and DMX identifiers, otherwise the queries are the same.

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.

Summary
`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 instantaneous (output or relay) channel level
  Query: `?CHANLEVELX,<addr>,<devcode>,<chan-num>;`
  Long-ack Format: `!OK,CHANLEVELX,<addr>,<devcode>,<chan-num>;`
  From: v02.00
  Reply: 'Report instantaneous output level'
  `!CHANLEVELX,<addr>,<devcode>,<chan-num>,<level>,<power>,<wattage>;`
  e.g.
  ```
  ?chanLevelx,1,12,2;
  !OK,CHANLEVELX,001,012,002;<CR><LF>
  !CHANLEVELX,001,012,002,030,099,00100;<CR><LF>
  ```

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

- request instantaneous DMX zone level
  Query: `?DMXLEVELX,<addr>,<devcode>,<zone-num>;`
  Long-ack Format: `!OK,DMXLEVELX,<addr>,<devcode>,<zone-num>;`
  From: v02.00
  Reply: 'Report instantaneous DMX zone level'
  `!DMXLEVELX,<addr>,<devcode>,<zone-num>,<level>,<power>,<wattage>;`
  e.g.
  ```
  ?dmxLevelx,1,15,2;
  !OK,DMXLEVELX,001,015,002;<CR><LF>
  !DMXLEVELX,001,015,002,255,033,02500;<CR><LF>
  ```

6.3 Advanced DALI Virtual Channel Addressing

In eDIN+ systems, we work hard to make DALI lighting channels the same as other lighting channels, simplifying their control, so DALI buses are accessed via *Mode DALI channels*, rather than native DALI operations that use native DALI addressing.

But on occasions we need to use *DALI identifiers* to control the DALI *virtual channels*, but these are restricted in their use.

Mode DALI channels need to be mapped on to native DALI operations and native DALI addressing, which gives rise to three *UBC operating modes* that changes how the mapping is done:

- Broadcast,
- Standard (Group), or
- Advanced (Channel).

In some of these operating modes, native DALI Broadcast and Group addressing peep through and we create *DALI virtual channels* that enable us to perform native DALI Group and Broadcast operations. We access these virtual channels using *DALI identifiers.* However, these *virtual channels* only have limited functionality and (as they represent a collection of Mode channels and DALI fixtures) cannot have status or levels of their own, and in general *cannot be queried*.

To bypass the Mode DALI and virtual channel mechanism, and to access direct native DALI addressing, you must use the low level XDALI API – see Volume 3.

6.3.1 Advanced Virtual DALI Output Commands

It is possible to send DALI fade and stop commands to the *virtual channels* `BST` or `Gxx` (if they exist) using the following commands.

These commands follow the standard `$DALIFADE` and `$DALISTOP` syntax (see Gateway Interface Volume 1 - Standard), except the Mode DALI *channel number* parameter is replaced with the following advanced *DALI identifiers*

- `BST` – DALI Broadcast – all fixtures (*UBC operating modes* Standard & Advanced)
- `G00` to `G15` – DALI Group – commissioned DALI groups (*UBC operating mode* Advanced)

The behaviour of these commands are undefined if they are used on UBCs that are not in the correct *operating mode*.

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

- Stop virtual DALI channel fade
  Command: `$DALISTOP,<addr>,<devcode>,<dali-id>;`
  Long-ack Format: `!OK,DALISTOP,<addr>,<devcode>,<dali-nid>;`
  From: v01.00
  e.g.
  ```
  $DALIstop,4,17,g0;
  !OK,DALISTOP,004,017,G00;<CR><LF>
  ```

6.3.2 Advanced Virtual DALI Output Events

When the advanced DALI output commands are used the equivalent DALI event messages are generated. These are the same as the standard DALI events (see section 3.3.1 - General Output Events) except the `BST` or `Gxx` *DALI identifiers* are used.

For example,

- In UBC Standard mode and UBC Advanced (Channel) mode
  ```
  $dalifade,02,17,BST,255,1000;
  !OK,DALIFADE,002,17,BST,255,00001000;<CR><LF>
  !DALIFADE,002,17,BST,255,00001000;<CR><LF>
  ```
- In UBC Advanced (Channel) mode
  ```
  $dalifade,02,17,G00,255,1000;
  !OK,DALIFADE,002,17,G00,255,00001000;<CR><LF>
  !DALIFADE,002,17,G00,255,00001000;<CR><LF>
  ```

6.3.3 Advanced DALI Fixture Health Query

Volume 1 talks about System health and the `?ERRORS` query and Channel status/health but glosses over DALI channel and fixture health. This section gives more details on this topic.

If configured to do so, a UBC can report problems with individual DALI fixtures. It can report if a DALI fixture has a lamp problem (status code 25), if it is not reporting / missing (status code 26), or if it its settings do not match those saved during commissioning (status code 22).

It reports the problems both on the relevant *Mode DALI channel* with a `!DALIERR,<addr>,<devcode>,<chan-num>,<status-code>`, and on the exact fixture with a second `!DALIERR,<addr>,<devcode>,Fxx,<status-code>` event (Vol 2 Channel Health events).

These fixture errors will also be reported when requested with the `?ERRORS` query (Volume 1 System Health).

It is also possible to query the current status of an individual DALI Fixture with the *Request DALI fixture status* query. This query is the same as the *request DALI channel status* query, except that a fixture *DALI identifier* is used, and only the *Report DALI fixture health* message is returned.

The *fixture DALI identifier* is an `F` followed by its DALI short address `0-63`, e.g. `F01` for fixture at DALI short address 1.

- *Query:* `?DALI,<addr>,<devcode>,<dali-fixture>;`
  Long-ack Format: `!OK,DALI,<addr>,<devcode>,<dali-fixture>;`
  From: v02.00
  Reply: 'Report DALI fixture health'
  `!DALIERR,<addr>,<devcode>,<dali-fixture>,<status-code>;`
  e.g.
  ```
  ?dali,1,17,F1;
  !OK,DALI,001,017,F01;<CR><LF>
  !DALIERR,001,017,F01,000;<CR><LF>
  ```

A fixture **must be** a *known fixture* to be reported, i.e. fixtures that exist in the *expected fixture list* in eDIN+ system's DALI commissioning data. Just like with channels that do not exist, if the fixture is not known, it will not report a health message.

6.4 Controlling Input Channels and Injecting Input Events

There are 3 types of input channels: Contact inputs, PIR or Occupancy inputs, and Analogue inputs. Also, contact inputs can be used in one of 2 ways:

- as simple contact closure switches, e.g. room partition switches, where the state of the switch is important,
- as momentary or push-button switches, similar to eDIN+ button channels, where the timing of events is important and the state of the switch is not used.

It is possible to determine the type of input channels using the standard `?INP` 'Request input channel status' query. Each type uses a different response:

- `INPSTATE` for contact inputs
- `INPPIR` for PIR or occupancy inputs
- `INPLEVEL` for analogue inputs

In general it is not possible to control the state of an input channel, just monitor its state and react to changes in state. For example, how can you change a contact switch from open to closed except by physically moving the switch? So for most of the time you will be polling the state of input channels using the `?INP` query or listening for events that announce changes of input state.

However, there are a number of situations where you might want to mimic an input channel or force the eDIN+ controller to perform certain actions that you know are triggered on certain input change-of-state events. This is possible via the GATEWAY using `$BTNSTATE`, `$INPSTATE` and `$INPLEVEL` commands.

eDIN+ PIR input channels have advanced features for sophisticated performance and can have their channel state controlled using commands. It is possible to control the actual state of PIR channels via the GATEWAY using the `$INPPIR` command.

6.4.1 Simple Contact Closure Input Channels

When queried simple contact input channels always report their state as either:

- `state = 0` when inactive (the 'Normally Closed' or 'Normally Open' state)
- `state = 1` when active

Whenever the contact input switches change state they always generate a *'Change in switched input'* event whose new state is either:

- `Release-off (0)` - default inactive state of the input channel
- `Press-on (1)` - active state of the input channel

All other input events can be ignored.

It is possible to override the actual input state to `Release-off` or `Press-on` using the `$INPSTATE` command. This command will generate a suitable *'Change in switched*

input' event and this state will be reported when queried with `?INP`. This override state will persist until it is changed either with another `$INPSTATE` command or when that actual input channel reports a *'Change in switched input'* event.

Care should be taken when using the `$INPSTATE` command with simple contact input channels as it leaves the state of the eDIN+ system in a confused state, where the system thinks the channel is in one state when the reality may be different.

6.4.2 Button and Momentary Contact Input Channels

Button and momentary contact input channels have additional events that relate to how long the button or contact switch has been pressed, so that different actions can occur if a button is held for a sufficient time or not.

When queried buttons and momentary contact input channels always report their state as either:

- `state = 0` when inactive (not pressed)
- `state = 1` when active (being pressed)

However as they are only active `1` momentarily when the button or contact switch is pressed, their state is almost always `0` and querying the button or channel is not particularly useful. Querying does not report the held state. If this state is important, it has to be tracked via events.

Whenever the contact input switches change state they always generate a *'Change in switched input'* event whose state is either:

- `Release-off (0)` - immediately the button or input switch is released
- `Press-on (1)` - immediately the button or input switch is pressed

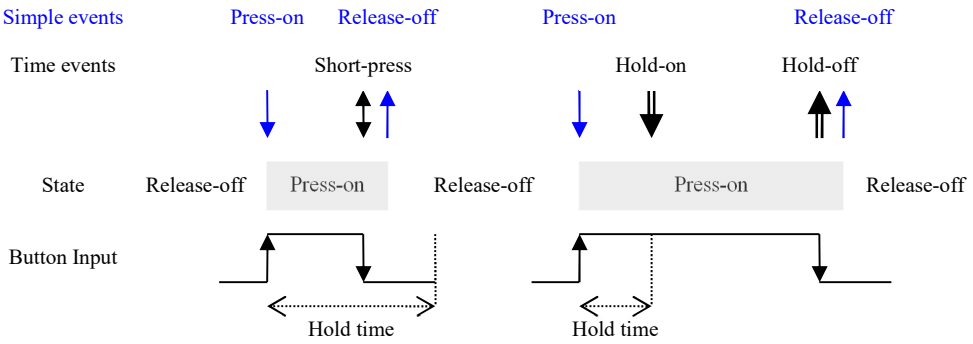
In addition to the `Press-on` and `Release-off` events there are 3 other events that are generated, depending on how long the button or switch is pressed.

- `Hold-on (2)` - when pressed and held for a sufficient time
- `Hold-off (6)` - when released after a `Hold-on`
- `Short-press (5)` - when released but no `Hold-on` event occurred

So button and (all) contact input channels generate 2 different sequences of events:

- For a short press inject the sequence:
  when pressed `Press-on` (when pressed) then `Short-press`, `Release-off` (when released)
- For a long held press inject the sequence:
  when pressed `Press-on` (when pressed) then `Hold-on` (when held for a sufficient time) then `Hold-off`, `Release-off` (when released)

The diagram below shows the two sequences of events, depending on how long the button is held for.



It is possible to override the actual button or input state and generated an input event for any of the 5 events using the `$INPSTATE` and `$BTNSTATE` commands. This command will generate a suitable *'Change in switched input'* event and trigger any rules connected to the button or input channel event. In this way you can *mimic* or *simulate* buttons or input switch channels via the GATEWAY.

It is almost always safe to use the `$INPSTATE` and `$BTNSTATE` commands with button and momentary contact input channels to generate single or sequences of input events. Although this command affects the internal state of the channel, similar to simple contact closure input channels, this state is rarely if ever used for momentary channels and so is unimportant.

If input events are injected via the GATEWAY at the same time the actual button or input channel is pressed/released, the events from the 2 sources will interleave, but usually no harm is done, and is no different from if 2 buttons are configured to perform the same actions and are pressed together. The result is usually 'last event wins'.

6.4.2.1 Button and Contact input channel timings explained

This section explains how events on contact input channels are timed, and what the related timing parameters do. You do not need to be concerned with this section unless you are getting un-reliable behaviour from your contact input.

Contact Input channels have a number of timing parameters that can be configured:

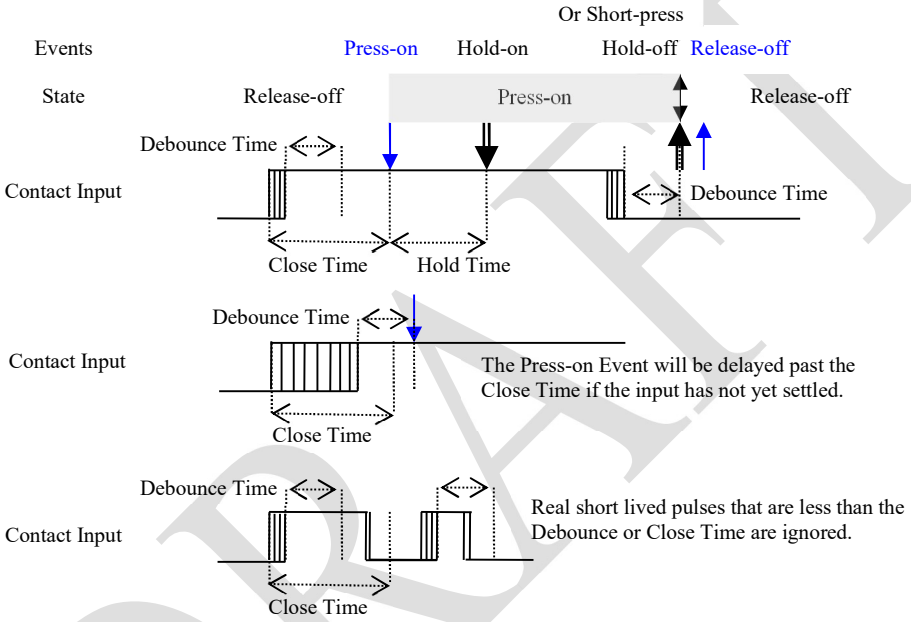
- Debounce Time
- Close Time
- Hold Time

Buttons have the same timing parameters, but they are fixed and cannot be configured.

The *Debounce Time* is used to time a stable input after all changes have settled. Mechanical switches do not usually have a single 'clean' switch between open and close states. Particularly with contact closures, the actual contact bounces for a small time generating a short burst of open/close events. The *Debounce Time* is used to detect a clean change of state and reject short lived noise. It is used on both open and closure events. This parameter does not normally need adjustment from its default value. It may need increasing if you have a particularly 'heavy' switch that has an unusually long bounce/settling time. If the *Debounce Time* is too short you may notice erroneous extra `Press-on` and `Release-off` *events*.

The *Close Time* is used to generate a consistent time for the `Press-on` event and is timed from the start or first *unclean* contact closure. As mechanical switches can vary how much bounce is generated every time it switches, if the `Press-on` event was timed from the end of the *Debounce* period it also would vary from when the contact is first switched. The *Close Time* ensures the `Press-on` event, and actions connected to it such as scene recalls, always occur with the same latency after button is initially pressed or switch closed. The *Close Time* also rejects short lived pulses that are less than the *Close Time*.

The following diagram illustrates the *Debounce* and *Close* times and how they relate to each other.



The *Hold Time* is used to generate `Hold-on` events. It is timed from the `Press-on` event, so that you must add the *Close Time* and the *Hold Time* together if you want the time of the `Hold-on` event from the initial switch closure.

The *Inactive Time* is no longer used and can be ignored.

6.4.3 PIR Input Channels

eDIN+ PIR input channels are used with PIR (Passive-Infra-Red) or other occupancy sensors to monitor occupancy of a physical area, and to automate the turning off and maybe on of lights. A channel generates a *presence* event when it detects the physical area has become newly occupied, and an *absence* event when the area is no longer occupied. For fully automated *presence* control, lights are turned both on & off automatically on *presence* & *absence* events. For partially automated *absence-only* control, lights are only turned off automatically using the *absence* event.

eDIN+ *PIR input* channels are *not*, and should not be confused with, *contact input* channels. PIR channels process the (often contact/relay closure) output from a PIR or

occupancy sensor, and convert this in to an *occupied* state. They also respond to commands sent by the controller, so that they work in sympathy with the controller.

There are a few issues when the automated control of either *presence* or *absence* control is mixed with manual control from e.g. a wall plate. This is why you may want to send the channel commands.

- When lights are turned on manually, we need a future *absence* event to turn the lights off automatically.
  We do this by sending a `SET` command to channel that sets the channel in to the `occupied` state (if it is not already set) but without generating a *presence* event that could override the manual turning on. A future *absence* event will be generated when the channel no longer detects occupancy.
- When lights are turned off manually because everyone is leaving the room, we want the channel to reflect this and become `unoccupied`, so that it will detect a new presence and turns the lights on when someone walks back in.
  We do this with the `CLEAR` command, that clears the channel to the `unoccupied` state and also inhibits new detections for a short time so that leaving the room does not re-trigger the sensor and turn the lights on again.
- Presence and absence *detection* is not perfect so sometimes we want the automated detection to be temporarily suspended. Typically this is for an *Audio Visual scene* when a meeting room has been darkened and is showing a film or video. Because viewers are sitting quietly, sometimes the sensors detect that the room is no longer occupied generating an *absence* event, then a viewer moves and a new occupancy is detected and a new *presence* event is generated. However, we do not want the lights to change or be controlled by these events. We suspend a channel from generating events using the `HOLD` command. This sets and holds the channel in to the `occupied` state indefinitely, but without causing *presence* or *absence* events, until the channel is sent another command that releases it.
- It may be that the controller wants to force presence or absence events to resynchronise the lights to the physical state of an area and channel. The `TRIGGER`, `TIMEOUT` and `SYNC` commands force the channel to send a *presence* or *absence* event.

When queried PIR channels always report their state as either:

- `state = 0` when empty or `unoccupied`
- `state = 1` when `occupied`

PIR channels generate *'Change in PIR input state'* events reporting the following event states:

- `Empty` (0) - sensor at rest, no occupancy, possible *hard* alternative to *absence* event
- `Triggered-On` (1) - *presence* event, active sensor detected
- `Timeout-Off` (2) - *absence* event, inactive sensor, timeout occurred
- `Set-On` (3) - manual *presence* override, used to track occupancy
- `Cleared-Off` (5) - manual *absence* override, used to track occupancy

PIR channels respond to the following commands. These commands are sent via the GATEWAY using the `$INPPIRSET` command.

- `TRIGGER` (`1`) - force an immediate *presence* event, timeout re-started
- `TIMEOUT` (`2`) - force an immediate *absence* event, timeout cleared
- `SYNC` (`0`) - force an immediate *presence* or *absence* event depending on current internal state of channel.
- `SET` (`3`) - manual *presence* override, timeout re-started
- `HOLD` (`4`) - manual *presence* override, timeout suspended
- `CLEAR` (`5`) - manual *absence* override, timeout cleared and inhibit period started

6.4.3.1 PIR channel timings explained

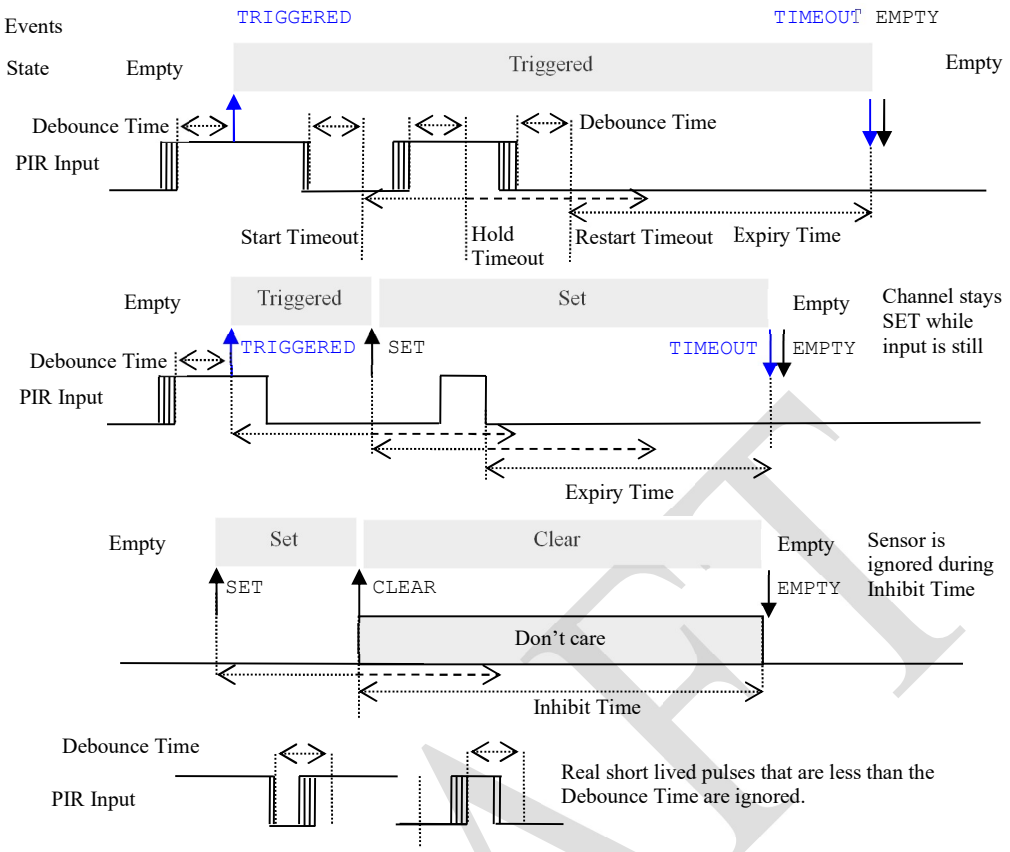
The PIR channel can work with a number of different sensor types.

- The channel can work with chatter-type sensors that generate a short contact closure at regular intervals while they detect presence.
- The channel can work with self-timed sensors that generate a single contact closure when presence is first detected, and remains closed until presence is no longer detected. **It is recommended that this type of sensor is configured to have its self-time period less then 10 seconds**, so that `TIMEOUT`, `CLEAR` and walk-test modes work correctly.
- The channel can be configured to work with Normally-Open or Normally-Closed type sensors.

PIR Input channels have a number of timing parameters that can be configured:

- Debounce Time - to clean up sensor inputs
- Expiry Time - the timeout period a channel remains occupied
- Inhibit Time - the inhibit time used in the `CLEAR` override state

The following diagram illustrates the timing of a PIR channel.



The *Debounce Time* is used in the same way as Contact Inputs – to ensure clean changes of state. The input must be settled for the *Debounce Time* for a change of state to be registered, to avoid extraneous events.

The *Expiry Time* defines the timeout period of the `occupied` state. The timeout period gets restarted every active sensor input and for every `SET` command sent. Although the channel is triggered by the leading contact closure event of the sensor, the timeout period does not actually begin until the sensor releases the input (see diagram above). This allows the channel to work correctly with the different types of sensor. This means that the sensor's self-timed period and the channel's expiry period are added together. **This makes it important that you set the sensor's period to less than 10 seconds and use the channel's expiry period to time the absence**.

The *Inhibit Time* is used in the `CLEAR` override state. The *Inhibit* period restarts for every `CLEAR` command sent. During this period the state of the sensor input is ignored. At the end of this period the channel moves to the `EMPTY` state and the sensor input re-enabled.
It is recommended to keep the inhibit period short – typically 20 seconds or less. This ensures that the sensors are not disabled for extended periods and channels can be retriggered quickly. If longer inhibit periods are required, e.g. a wall plate is at the other side of a large room, it is recommended that the wall plate button event starts a Delay Timer object, and the Delay Timer object issues the `CLEAR` command when it times out, rather than the wall plat button issuing the `CLEAR` command directly.

6.4.4 Analogue Input Channels

eDIN+ Analogue Input channels are used to monitor a control voltage from an external controller e.g. a slider control, or monitor the output voltage from an analogue sensor, e.g. a lux or temperature sensor. They have a reporting range, and report values and changes of value within this range. This reporting range defaults to `0-255`, but can be *scaled* so that it reports more meaningful values to the controller. Also, *scaling* allows differences between physical channels/sensors to be adjusted for and removed. For example, all lux sensor channels can be adjusted and *scaled* so that they all report the same nominal 300 Lux, even when the actual input level to achieve this 300 Lux is different.
Note: the *scaling* mechanism is for convenience and does not need to be performed.

There are 3 types of analogue input channels:

- Analogue input – most used
  Has a physical analogue voltage input range of 0 - 10V, and by default has a calibrated range of `0-255`.
- DSI input
  Is a digital serial inputs that receives values 0-255 that conform to the DSI serial format.
- Evo input
  Has a physical analogue voltage input range of 0 - 9.3V, and reporting range of `0-255`. Is used to support an imperfection in some or our old products, and it is not expected that you will normally use this type.

Analogue channels have 2 properties that control when the channel generates an event that reports its current analogue level. It is usual to use one property or the other but not both.

- level sensitivity – this sets a threshold on how much the channel's level has to change since it was last reported before the new level will be reported. Any changes in level smaller than this threshold will not generate an event and the new level will not be reported. This stops small changes, possibly noise, from being reported.
  The channel will only report its level based on significant changes in level.
  The special value of `0` turns this feature off.
- time period – this sets a reporting time period threshold. Every time the channel reports its level it resets a reporting timer. If the timer reaches the reporting time period, the channel will generate an event to report its level. This enables a channel to periodically report its level based on time, irrespective of its level.
  The special value of `0` turns this feature off.

Level sensitivity is primarily used for simple mirroring of control inputs, such as from a slider control, to one or more output lighting channels. Time periods are used primarily with sensors for more sophisticated control, to periodically update the internal controller.

Analogue input channels can have their current level and status queried with the `?INP` query.

It is possible to override the actual input level and generated an input event for any level value using the `$INPLEVELSET` command. This command will generate a suitable *'Change in analogue input'* event and trigger any rules connected to the input channel. The override level value will persist until an new event either from the actual channel or from another `$INPLEVELSET` command.

7 System Adjustments

There are a limited number of configuration properties that can be adjusted via the GATEWAY interface. They include:

- Channel levels defined in scenes
- Pre-set colour and tuneable white values used by modules
- Times defined in Timelines and Delay timers.

All these adjustments affect the `adjust-stamp` value in the `?SYSTEMID` query.

Simple live scene setting is described in Volume 1.
Adjusting pre-set colours and temperatures is describe in this chapter in Volume 2.
Adjusting pre-set times of Timelines and Delays is describe in this chapter in Volume 2.

7.1 Colour and Tuneable White Pre-set Adjustments

Commands that set the colour or colour temperature of a colour channel can use either a direct rgb or kelvin value, or a pre-defined or pre-set value. It is possible to set the direct rgb or kelvin value of these pre-sets via the GATEWAY.

Each module has its own set of colour and temperature pre-sets that **are shared by all channels or DMX zones on that module**. Because adjusting pre-set values affects more than one channel and may affect channels in more than one area, you **must have *administrator privileges* if you are using user accounts**.

As these commands permanently change the value of the pre-set, it is recommended that you first determine the new value of the pre-set indirectly by adjusting the colour or temperature of a channel, using the `$CHANRGBCOLR` or similar command.

Summary
`rgb-preset` - adjustable pre-set static colours are `1-15`.
`tw-preset` – adjustable pre-set colour temperatures are `48-55`.
`wrgb` - direct colour specified as a 6- or 8-hexdigit colour, e.g. red = `#FF0000`
`kelvin` - direct kelvin colour temperature typically `1800K` to `7000K`

- Set Pre-set Colour
  Command: `$RGBPRESET,<addr>,<devcode>,<rgb-preset>,#<wrgb>;`
  Long-ack Format: `!OK,RGBPRESET,<rgb-preset>,#<wrgb>;`
  From: v02.00
  e.g.
  ```
  $rgbPreset,02,15,03,#FF00FF;
  !OK,RGBPRESET,002,015,003,#FF00FF;<CR><LF>
  ```

- Set Pre-set Kelvin
  Command: `$TWPRESET,<addr>,<devcode>,<tw-preset>,#<kelvin>K;`
  Long-ack Format: `!OK,TWPRESET,<tw-preset>,#<kelvin>K;`
  From: v02.00
  e.g.

```
$twPreset,02,15,50,#2100k;
!OK,TWPRESET,002,015,050,#2100K;<CR><LF>
```

7.2 Timeline adjustments

To be completed