Mode_Lighting_Control_small 50px

Title:

Version:

Author:

Date: 14 February 2018

**History**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Ver | Initials | Changes |
| 8-May-14 | 0.1 | Jph | Initial attempt based on Technical Specification ver 0.6 |
| 16-Apr-15 | 0.2 | Jph | Update to gateway ver 1.03  Add HTTP connections  Minor corrections |
| 29-Sep-15 | 1.0.0 | Jph | Minor corrections  Add Contact Input timings  Add Area/Scn/Plate Discovery section |
| 29-Apr-16 | 1.1.0 | Jph | Update to Gateway ver 1.04  Revised PIR channel sections  Revised DMX Zone channels  Added Using DMX App Note |
| 9-May-16 | 1.1.1 | Jph | Add SCNBACKON cmd & event |
| 13-Sep-16 | 1.2.0 | Jph | Update to Gateway ver 1.05  Add DALI Repair App Note  Add DALI Emergency Testing App Note |
| 22-Sep-16 | 1.2.1 | Jph | Amend DALI repair App to add DALISCAN and DALIFIX queries. |
| 12-Dec-16 | 1.3.0 | Jph | Update to Gateway ver 1.06  Revise |
| 28-Mar-17 | 1.3.1 | Jph | Gateway ver 1.07  Correct advanced DALI ALL to BST  Correct DALI fixture status codes in !DALFIX messages  Add to DALI repair section, info on ?DALIERR query |
| 14-Feb-18 | 1.3.2 | Jph | Gateway ver 1.08  Revise DALI repair section |

**Table of Contents**

[1 Introduction 3](#_Toc506378824)

[1.1 Purpose and scope 3](#_Toc506378825)

[1.2 Terminology 3](#_Toc506378826)

[1.3 References 3](#_Toc506378827)

[2 Connecting to and Troubleshooting the Gateway Interface 4](#_Toc506378828)

[2.1 Connections 4](#_Toc506378829)

[2.1.1 Raw TCP/IP 4](#_Toc506378830)

[2.1.2 RS232 4](#_Toc506378831)

[2.1.3 HTTP 5](#_Toc506378832)

[2.2 Troubleshooting 5](#_Toc506378833)

[2.2.1 Troubleshooting the Raw IP & RS232 connections 5](#_Toc506378834)

[2.2.2 Troubleshooting the HTTP connection 6](#_Toc506378835)

[2.2.3 Troubleshooting a !BAD command or query 7](#_Toc506378836)

[2.2.4 Troubleshooting a command or query with the wrong or no response 7](#_Toc506378837)

[2.2.5 Troubleshooting using Gateway Events 8](#_Toc506378838)

[3 Understanding Channel Types 10](#_Toc506378839)

[3.1.1 Output Channels 10](#_Toc506378840)

[3.1.2 Button Channels 12](#_Toc506378841)

[3.1.3 Contact Input Channels 14](#_Toc506378842)

[3.1.4 PIR Input Channels 16](#_Toc506378843)

[3.1.5 Analogue Input Channels 18](#_Toc506378844)

[4 Using eDIN+ Multi-zone DMX 19](#_Toc506378845)

[4.1 Understanding eDIN+ DMX multi-zones 19](#_Toc506378846)

[4.2 Understanding eDIN DMX modes 19](#_Toc506378847)

[4.3 Understanding the eDIN DMX palette 21](#_Toc506378848)

[4.4 Bypassing eDIN DMX modes 21](#_Toc506378849)

[5 Using eDIN+ DALI Broadcast Channels 23](#_Toc506378850)

[5.1 DALI Broadcast Channel Types and Initialisation 23](#_Toc506378851)

[6 Advanced DALI Operation 25](#_Toc506378852)

[7 Advanced DALI Repair 26](#_Toc506378853)

[7.1 Identifying Faulty Fixtures 26](#_Toc506378854)

[7.2 Step 1 – Perform a scan 27](#_Toc506378855)

[7.3 Step 2 – Identifying the correct new fixture to repair with 28](#_Toc506378856)

[7.4 Step 3 – Repairing a missing fixture with the matching new fixture 29](#_Toc506378857)

[7.5 Step 4 – Tidying the Expected Data 29](#_Toc506378858)

[7.6 Step 5 – End to repair session 30](#_Toc506378859)

[7.7 Step 6 – Back up the configuration 30](#_Toc506378860)

[7.8 Example 30](#_Toc506378861)

[8 Advanced DALI Emergency Lighting Support 32](#_Toc506378862)

[9 Scene Setting 33](#_Toc506378863)

[9.1.1 Live Scene Setting 33](#_Toc506378864)

[9.1.2 Off-line Scene Setting 33](#_Toc506378865)

[10 System Discovery through the Gateway 35](#_Toc506378866)

# Introduction

## Purpose and scope

This document contains application notes for the eDIN Gateway Interface.

## Terminology

|  |  |
| --- | --- |
| Phrase | Description |
| DALI | Digital Addressable Lighting Interface. An industry standard physical interface/network for control of lighting equipment. It allows bi-directional data exchange, allowing its use with sensors, switches and emergency lighting as well as dimmable luminaries. |

## 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).

# Connecting to and Troubleshooting the Gateway Interface

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

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

## Connections

There are three ways to connect to the NPU

1. Ethernet / raw TCP-IP connection
2. NPU local RS232 port
3. HTTP connection

Note: It is ***NOT*** possible to use an EVO-INT232 module.

It is possible to open more than one connection (also called session) on each NPU simultaneously. Currently each NPU can handle 1 RS232 session plus 4 rawIP sessions, but this may change over time. Note: HTTP connections do not open sessions and each (short-lived) HTTP request is handled independently from the next. As such the only limit is how many instantaneous HTTP request (including from web browsers) the NPU can handle.

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

### Raw TCP/IP

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

* Ensure ‘Enable gateway control’ is ticked
* Adjust the ‘Use port’ value as required or leave as the default value (port 26)

Connections can now be made.

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

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

The gateway interface *automatically* closes raw TCP/IP connections that are idle for 1 hour. The gateway interface is idle if it does not receive any commands or queries or has not sent any events. To keep a connection open send the null command ($OK;) at least once an hour.

### RS232

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

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

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

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

This is does this when

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

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

### HTTP

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

The http connection uses AJAX http POST requests to the URL http://<npu\_ip\_address>/gateway? (Note the terminating ? is required) using the Content-Type: text/plain.

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

Example in JQuery:

$.ajax({

type: "POST",

url: 'http://'+ipaddr+'/gateway?',

contentType: "text/plain;",

dataType: "text",

data: "$scnRecall,1;",

})

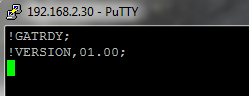
The http connection does not support long term sessions. Each http request/response stands by itself. As a consequence no event data is published on the http connections. Only commands and queries can be used.

## Troubleshooting

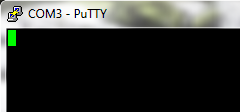
### Troubleshooting the Raw IP & RS232 connections

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

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

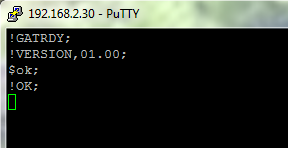


For RS232 connections, open PuTTy and in the session configuration dialog set the ‘Connection type:’ as ‘Serial’, in the Connection->Serial dialog set up the COM port appropriately (default 9600,8,1,no parity, no flow control) as configured on the web page (see section above), and in the Termial dialog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



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

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



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

### Troubleshooting the HTTP connection

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

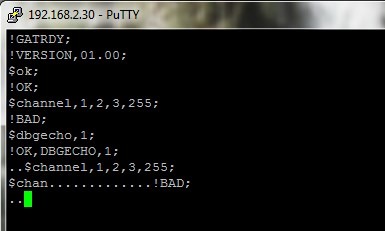
Within your application, here is a checklist to try:

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

### Troubleshooting a !BAD command or query

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

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



Gateway stop recognising command here

Two dots caused by <LF><CR> chars

Turn on DBGECHO feature

Problem command

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

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

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

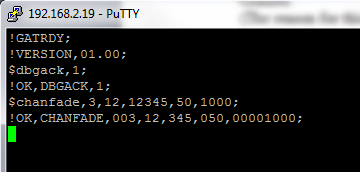
The first thing to note is that the !OK; acknowledgment is ***only*** saying that ***the syntax*** of the command or query ***is correct***, i.e. the text string is a recognised command or query. But ***it does not say if the command or query makes sense*** on your actual system. For example, it does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

The second thing to note is that if you ask the system to do something it cannot do, ***it will*** simply ignore the command or request and ***do nothing***. It will ***NOT*** send you message telling you that it cannot perform the requested operation. For example, if you send a query to a channel that does not exist, the Gateway will acknowledge that it has received the query, but no query response will be generated by the non-existing channel.

(The reason for this behaviour is that the Gateway interface itself does not know anything about your system so it cannot check if the channel exists or is of the correct type. This behaviour is applied consistently across all our different types of system.)

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

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



channel parameter truncated by Gateway

Problem - too big channel parameter

Turn on long acks

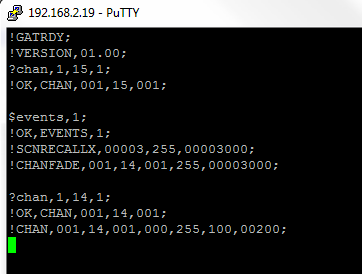
Note: with HTTP connections long acknowledgements are used by default, so you should not need to include the $DBGACK,1; command.

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

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

### Troubleshooting using Gateway Events

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



Scene recalled with channel

Channel has different devcode

Good query with correct devcode

Turn on events

Problem - no response to query

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

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

# Understanding Channel Types

The eDIN Lighting system has a number of different channel types. This App Note helps to understand the different types and how they are used with the Gateway interface.

### Output Channels

There are 2 different ways an output channel can operate:

* Full Analogue Output – most mains & low voltage channels
* Contact Output or Switched Analogue Output – mains dimmable and relay channels can operate in this mode
* An eDIN DMX zone looks like an output channel when controlling its brightness

There are 4 different commands to set the level of an output channel:

* Set channel to level over time ($CHANFADE)
* Set channel to state ($CHANSTATE)
* Set DALI channel to level over time ($DALIFADE)
* Set DMX zone brightness to level over time ($DMXLEVEL)

‘Set channel to state’ is primarily used with relay outputs on RP-03-2 and eDIN Relay modules to switch the channel immediately with no fade. However, ‘Set channel to state’ and ‘Set Channel to Level over 0ms’ have the same effect on analogue and output channels and the commands can be used interchangeably.

‘Set channel to level over time’ is used to produce a fade on both analogue and contact outputs. Contact outputs and switched analogue outputs will switch when their ‘internal’ level reaches the configured threshold in the fade. So if the configured threshold is at level ‘1’, the switched channel will turn on at the start of a ‘Fade Up From Level Zero’ operations and turn off at the end of a ‘Fade Down to Zero’ operation. In this way a mains channel can be use to switch the mains power to a low voltage fixture.

MODE DALI universe channels on the addressable DALI universes of eDIN UBC modules can only be controlled with the ‘Set DALI channel to level over time’ command. The command is ignored by all other channels.

(Note: Output channels configured as ‘DALI Broadcast’ outputs are controlled using the standard ‘Set channel … commands and queries and can NOT be controlled using ‘Set DALI channel …’ command and queries. The ‘Set DALI channel …’ command is only for true MODE DALI channels on an addressable DALI universe on a UBC module.)

EDIN DMX zones operate as full analogue channels and should be sent a ‘Set DMX zone to level over time’ fade command. The colour of the DMX zone is controlled with other commands. Refer the eDIN DMX Operation App Note section for further details.

Output channel queries report the current power of the channel as well as its current control level and status. The reported power is not an accurate measure of the real power being used. It is an indication of expected power based on the type of output channel. It is reported as a percentage of maximum power.

The maximum power of a channel is given in a second parameter, and is configured for each channel using the commissioning tools. It is reported in Watts.

The power is calculated from the control level using a dimming curve. The dimming curve is fixed for each channel type as follows:

* Mode DALI channels and Low voltage DALI Broadcast channels use the logarithmic dimming curve defined by the standard (ref [1]), unless it is set for linear control.
* Mains dimmed channels use the cosine or ‘S-shaped’ curve that matches the characteristics of a chopped main sine wave.
* Relay and switched channels report either zero (off) or maximum power (on). Note: if these channels are linked to (i.e. used to switch on & off) a low voltage channel, the relay/switched maximum channel power should be configured to 0 and only the low voltage channel configured to the actual maximum power. This will stop the power used by lighting control gear being reported twice.
* DMX zones combine the level of each rgb or rgbw component to estimate the power. So to report maximum power, all components must be at maximum power.  
  Also, the power is reported as an average over time, so for sequence modes the reported power is the average over a full sequence rather than the instantaneous power when the query is issued.  
  To determine the value to Maximum Wattage to configure for the zone, add together the power rating for each component in a fixture (to obtain the power rating of the fixture) then multiply this by the number of fixtures in the zone.  
  Examples:
  + For RGB fixtures, a static Red scene has the Red lamp on each fixture at maximum and the others at zero, so the reported power is 33%.
  + For RGBW fixtures, a static Red scene still has only the red lamp at maximum, but as there are 4 lamps per fixture the current power is reported as 25%.
  + Static patterned scenes assume the number of actual fixtures is a multiple of the scene pattern repeat, so the Short Rainbow pattern repeats every 6 fixtures and its reported current power assumes there are 6, 12 or 18 (etc) fixtures connected.
  + Dynamic ripple scenes also assume the number of actual fixtures matches the ripple pattern repeat when reporting current power, (compare with static pattern scenes). As each fixture takes on each colour in the pattern over time, the averaged reported power is correct for each fixture over a full ripple cycle.
* All other channels use a linear curve.

### Button Channels

Buttons have the following operations through the Gateway interface:

* Their current status, colour and text can be queried
* Their colour and text can be set with $BTNCOLR and $BTNTEXT commands
* Input events can be ‘injected’ with the $BTNSTATE command
* Button activity can be monitored by configuring the Gateway interface to send out Button events

Mode switch plates have a set of buttons that act as momentary contact input switches. They also have a tri-colour LED and (for LCD plates) text associated with each button. The colour and text of the button is independent of the contact state of the button, and is explicitly set as an independent output.

All plates use the same SGP\_XX device code ‘2’ , except for the LCD plate that uses device code ‘1’. All button numbering is based on the SGP55 10-button plate numbering. So 2- and 5- button plates use ‘Button 9’ for their last button. Coolbrium 8-button plates use button 1-8. See diagram below.

1

2

3

9

7

5

6

8

4

10

5-button SGP50

2-button SGP20

8-button CBM44

The LED colour is set with a colour from the system wide colour palette. The colour palette includes fixed standard colours (0-16) and user definable colours (128-143). The user colours are defined in the eDIN Web page configuration.

Button colour and text is set using the $BTNCOLR and $BTNTEXT commands.

WARNING: Button colour and text is usually controlled by the eDIN system, and overriding them through the Gateway as it is unlikely to be successful. However, you can monitor and query button states, colours and text successfully through the Gateway.

Buttons generate a sequence of input state events that is the combination of simple contact (on-off) events and time-dependent (Short-press and Hold) events.

A) Simple Contact Events

There are two simple contact events:

* Press-on (1) - start of active state
* Release-off (0) - start of default state

A Press-on event is generated as soon as the button is pressed. A Release-off event is generated when the button is released. These events do not depend on how long the button is pressed or held.

B) Time-dependent Short-press & Hold Events

There are 3 time-dependent contact events

* Short-press (5) - momentary short active pulse
* Hold-on (2) - start of a long active press
* Hold-off (6) - start of default state after held event

A Hold-on event is generated when the button is pressed and held for the button’s configured hold-time (default 1 second).

A Hold-off event is generated when a held button is released.

A Short-press event is generated if the button is pressed and then quickly released before the configured hold-time elapses, i.e. before a Hold-on event occurs.

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

Press-on

Release-off

Press-on

Release-off

Short-press

Hold-on

Hold-off

Hold time

Hold time

Simple events

Time events

State

Button Input

Press-on

Press-on

Release-off

Release-off

Release-off

Buttons operate on these momentary events. Actions, such as scene recalls, are tied these events. The simple Press-on event is used for simple actions such as scene recall. The time-dependent events are used for richer actions, such as impulse control.

It is possible to ‘inject’ button events via the gateway command $BTNSTATE. This allows you to mimic the action of a actual plate. You only need to inject the button event that triggers the action you require, although injecting the full sequence of events will ensure all actions tied to that button will be run as they are for the real button.

Due to the momentary nature of buttons, polling a button’s state should be avoided as it will give unreliable behaviour. Use event monitoring if you want to track the activity on buttons.

### Contact Input Channels

Contact input channels have the following operations through the Gateway interface:

* Their current status and state can be queried
* Input events can be sent out from the Gateway interface

These channel types include:

* External contact inputs on plates
* Contact inputs on CI0x, eDin 8CH IO and UBC modules

Contact Inputs are a little complicated as they are used to do a number of different things including:

* contact switches, e.g. room partition switches, where the state of the switch is important
* momentary contact switches as buttons, where the timing of events is important and the state of the switch is not used.

When contact inputs are used to monitor contact switch states, two input *events* are important:

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

The Release-off state is the natural or default state of a contact channel. The polarity of this state, i.e. ‘Normally Open’ or ‘Normally Closed’ can be configured for each channel. The Press-on state is the opposite and active state.

The input *state* is reported as Press-on or Release-off only. The Press-on *state* is set after a Press-on *event* has occurred and until the next Release-off *event* occurs. The Release-off *state* is reported at all other times, i.e. after a Release-off *event* until the next Press-on *event*.

Press-on or Release-off *events* are always generated when the state of the contact changes. You may query or poll the input *state* (and use Press-on and Release-off states for ‘require’/’AND’ rules in the configuration). You may use Press-on – Release-off *events* to trigger actions.

When used as a button, the same input event sequences are generated as they are for buttons, and you can use them interchangeably with button inputs. (See Button Channels section for further information.)

NOTE: it is not possible to ‘inject’ contact input events, as you can for buttons. This is because the system currently cannot determine if a contact input is being used as a button or as a contact switch, but it is not possible to override the physical state of a switch.

#### Contact Input Channel Timings

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

* Debounce Time
* Close Time
* Hold Time

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. 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.

Release-off

Hold-off

Press-on

Press-on

Hold-on

Close Time

Hold Time

Events

State

Contact Input

Release-off

Release-off

Debounce Time

Debounce Time

Or Short-press

Close Time

Contact Input

Debounce Time

The Press-on Event will be delayed past the Close Time if the input has not yet settled.

Contact Input

Debounce Time

Close Time

Real short lived pulses that are less than the Debounce or Close Time are ignored.

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.

### PIR Input Channels

PIR Input channels have the following operations through the Gateway interface:

* Their current status and state can be queried
* Their state can be manually overridden to synchronise their state with other system activities
* Their state can be monitored by polling or by configuring the Gateway interface to send out input PIR events

A PIR channel is used with a PIR occupancy sensor. It detects short lived pulses from the sensor eventually timing out when there is no longer activity from the sensor.

PIR channels have the following PIR events and states:

* EMPTY (0) - sensor at rest, no occupancy
* TRIGGERED (1) - active sensor detected, timeout re-started
* TIMEOUT (2) - inactive sensor, timeout occurred
* SET (3) - manual override, timeout re-started
* HOLD (4) - manual override, timeout suspended
* CLEAR (5) - manual override, timeout cleared and inhibit period started

*Presence* rules are tied to the TRIGGERED *event*. *Absence* rules are tied to the TIMEOUT *event*. (This means no rules are triggered by manual override operations.)

It is possible to manually override the state of a PIR input to SET, HOLD or CLEAR, using the $INPPIR command.

* The SET override is usually included in scenes that turn lights on, and is used to restart the PIR channel’s timeout. This ensures that even if the area is not occupied when the scene is recalled (and lights turned on), a TIMEOUT event will occur and *absence* rules triggered to turn lights off again.  
  Notes:  
  - There is no difference between TRIGGERED state and SET state other than the event generated and hence whether a *Presence* rule is run or not.  
  - If the sensor becomes active while the channel is in the SET state, the timeout will be retriggered but the channel will remain in the SET state and no new events are generated or *Presence* rules run.
* The CLEAR override is usually included in ‘off’ scenes that turn all lights off, and is used to clear the PIR channel so that it can be retriggered by the sensor. The CLEAR operation also includes an inhibit period that stops the channel being re-triggered by active sensor inputs. This is to allow time for a person to leave an area after they have turned the lights off from a wall plate without retriggered a *presence* event.
* The HOLD override is used to suspend the timeout, but keep the channel occupied. This is often used when Audio-Visual (AV) scenes are active, and people are still for extended periods of time, (not picked up by the sensor causing spurious *presence* and *absence* events).  
  Note: the HOLD state is held indefinitely, so this feature is normally used with another timed rule that SETs or CLEARs the channel to re-enable the sensor.

#### PIR Input Channel Timings

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.

Don’t care

CLEAR

SET

TIMEOUT

EMPTY

Triggered

TRIGGERED

Expiry Time

Events

State

PIR Input

Empty

Empty

Debounce Time

Debounce Time

Start Timeout

Restart Timeout

PIR Input

Debounce Time

Real short lived pulses that are less than the Debounce Time are ignored.

Triggered

Empty

Debounce Time

Channel stays SET while input is still active

PIR Input

Expiry Time

Set

SET

TRIGGERED

EMPTY

TIMEOUT

Empty

Hold Timeout

Set

Empty

Sensor is ignored during Inhibit Time

Inhibit Time

Clear

EMPTY

Empty

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 period 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 so. 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.

### Analogue Input Channels

Analogue channel types include:

* Analogue input
* DSI input
* Evo input

Analogue input channels can have their current level and status queried. Events can be sent out from the Gateway interface when the channel changes level or status.

Analogue input has an analogue voltage input range of 0 - 10V, and generates a value 0 – 255. This is the normal type of analogue input used.

Evo input has an analogue voltage input range of 0- 9.3V, and generates a value of 0 -255. This extra type was added to support an imperfection in some or our old products, and it is not expected that you will normally use this type.

DSI inputs are digital serial inputs that receives values 0-255 that conform to the DSI serial format.

# Using eDIN+ Multi-zone DMX

## Understanding eDIN+ DMX multi-zones

A DMX *universe* has 512 different addresses. Each address can be set to its own level (0-255). DMX colour *fixtures* use 3 or more addresses, one address for each colour component, for example an RGB fixture will use its first address for the Red colour level, the second address for Green colour level and the third address for Blue colour level. The actual DMX address of the first address of the DMX fixture can be configured on each fixture.

To make using DMX straightforward, the eDIN system splits a DMX *universe* in to a number of DMX *zones*. To define a *zone*, simply specify the first DMX address of the *zone* and the type of *fixture* used in the zone.  
Note:

- All *fixtures* within one *zone* must be of the same type, although *zones* on the same *universe* can have different types.

- eDIN DMX supports the following different DMX colour fixture types:

* RGB simple 3 colour fixture
* RGBW 4 colour fixture: red, green, blue and white
* RGBM 3 colour fixture with a Master Brightness control
* RGBWM 4 colour fixture with a Master Brightness control
* BGR 3 colour fixture with reverse addressing (blue is the first address)

The DMX zones are set up in the configuration and cannot be altered via the Gateway interface.

A DMX zone uses all the DMX addresses from its start address until it meets the next DMX zone. For example,

* If only 1 DMX zone is used, make its start address 1 and it will use all 512 DMX addresses.
* If there are 2 DMX zones starting at DMX address 1 and 256, then Zone 1 uses DMX address 1-255, and Zone 2 uses DMX address 256-512.

DMX zones work at the fixture level. Each zone assumes the maximum number of fixtures that will fit in to the zone’s DMX address range, and repeats its colour effect across all fixtures. This means that it is unnecessary to specify how many fixtures are in each zone – simply use as many as you require and you will be OK as long as you do not spill in to the next DMX zone’s address range.

NOTE: Often DMX fixtures are configured to use the same (set of) DMX address(es). This means that they look like a single fixture and work in unison. Most of eDIN+ DMX modes will work in this configuration, except for the Ripple modes. However, if you want more than one DMX zone, you need to make sure the fixtures in one zone are set to a DMX address that is different from fixtures in the other DMX zones.

## Understanding eDIN DMX modes

The eDIN lighting system has made using DMX straightforward. It has some pre-defined colour effects called DMX *modes*. To achieve the required effect simply set the DMX *zone* to the specific *mode* using the $DMXMODE command. The brightness of the DMX *zone* is controlled separately through brightness level commands.

Each DMX *universe* has its own colour palette of 15 colours, so all *zones* in the one *universe* (module) share a colour palette. The various colour modes use one or more of the palette colours for their effect. (See below for further details.)

There are 15 *static* modes – one for each entry in the palette. These simple display the chosen colour across the whole DMX zone.

There are 5 *solid* colour sequence modes. Each sequence uses its own pre-defined selection of colour entries from the palette. At any moment in time, the whole zone is set to the colour of that entry, but the zone continually cross-fades between the colours in its sequence. This provides a continually changing wash of colour.

There are 5 *ripple* colour sequence modes. Each sequence uses the same pre-defined selection of palette entries as the *solid* modes. However it differs from the *solid* modes in that at any moment in time the selected colours are spread across adjacent fixtures. If the sequence has 6 selected colours, then the first 6 fixtures of the zone will be set to these 6 colours. This colour effect is then repeated for the next 6 fixtures in the zone and so on to the end of the zone. In addition, each fixture is continually cross fading through the colour selection. The effect is a rainbow of colours that continually shifts to the left or right.

eDIN DMX Palette and Modes

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

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

The $DMXMODE command enables you to set the zone to a specific mode. You also have to specify a cross-fade time. For *static* modes, this cross-fade time is used once to fade the zone’s colour in to the new mode, and is usually a value of a few seconds. For the *solid* and *ripple* sequence modes, the cross-fade time sets the speed of the continuous cross-fade of the sequence. This is usually set to minutes.

Although the modes are pre-defined, there is some customisation available. Firstly, there is one *solid* mode and one *ripple* mode that uses a *user selection* of colours from the palette. The actual colours used in the *user selection* is set in the configuration and cannot be changed via the Gateway interface.

Secondly, the user can adjust the colours in the palette. This can be done using the $DMXPALETTE Gateway command. Although each entry in the colour palette has a default name and value, there is no restriction on what value these can be set to. (See below.)

## Understanding the eDIN DMX palette

EDIN DMX colour *modes* use one or more colours from a DMX colour *palette* to achieve their effect. Each DMX *universe* (module) has its own colour *palette* of 15 *entries* or colours, so all *zones* in the one *universe* (module) share a colour *palette*.

Each *entry* in the *palette* has a default name that corresponds to its default colour value. This is useful when using the sequence *modes* that cycle through a range of colours, or a limited number of preconfigured static colours of a precise shade are required.

However, it is possible to adjust the colour value of any or all of the individual *entries* in a DMX *palette*. This can be done using the $DMXPALETTE Gateway command. So instead of thinking on an *entry* as a particular shade of a specific colour, it may in some situation be better to nominally assigning *entries* to specific *zones* and label the *entry* as e.g. “Meeting Room 1 Colour”. Then this *entry* can be set to any colour as required.

There are 3 *entries* at the end of the DMX colour *palette* that are not used in the standard preconfigured sequence *modes*, with the default names USER1, USER2 and USER3. It is suggested that you up use these *entries* first, so that you can use the standard colour sequence *modes* unmodified. However, this restriction may be irrelevant to you, particularly if you do not intend to use the sequence *modes*.

Note: It is possible to redefine the name of each DMX palette entry. However this can not be done via the Gateway interface, and can only be done during configuration. In addition, the name of each entry is not used by the Gateway interface, so redefining the name is only useful as an *aid-memoir* during configuration.

## Bypassing eDIN DMX modes

Although the eDIN DMX *modes* provide useful colour effects, they all work with limited range of colours in the DMX palette. There is an alternative way to control the colour in a DMX *zone* from the Gateway interface, by setting the raw RGB(W) colour of the *zone* directly using the $DMXCOLR command. This overrides the current *mode* of the zone.

There are a number of limitation with the $DMXCOLR command.

* All *fixtures* in the *zone* are set to the same specified colour, in the same way *static* modes are. You cannot achieve *ripple-*type effects.
* There is no cross-fade with $DMXCOLR. The *zone* is updated to the specified colour instantly.
* The *zone* does not remember the override colour. If the eDIN module is restarted after a power cycle, the DMX zone will return to *mode* operation until another $DMXCOLR command is issued.
* The $DMXCOLR command does not block other *zone* commands. If the eDIN system issues other $DMXMODE commands the *zone* will return to *mode* operation.

The $DMXCOLR could be used in a couple of different situations.

Firstly, it could be used instead of using DMX *modes*. For example the user has a colour wheel that they use to directly set the colour of the DMX *zone*. The *zone* instantly adjusts its colour as the user adjusts the colour wheel.

Secondly, it could be used in a hybrid way to adjust the colour palette of the DMX *modes*. The DMX zone is normally set to particular DMX *mode* that uses say a single palette colour. If the user wants to adjust the *zone’s* colour, $DMXCOLR commands are sent while they are making the adjustment, so the *zone* mirrors the adjustments. Once the adjustment has been made, the new colour palette can be uploaded to the palette using the $DMXPALETTE command. The *zone* can be returned to its *mode* with a $DMXMODE command. As the colour palette is remembered by the eDIN system, this new colour will be used by the *zone* until the user makes another adjustment.

# Using eDIN+ DALI Broadcast Channels

Most DALI fittings are single channel (white) drivers. Our eDIN+ DALI Broadcast Output channels allow these fixtures to be used on a site without having to perform a full DALI commissioning process, and without having to re-commission when DALI fixtures are changed or repaired.

Our DALI Broadcast Output channels support the detection and reporting of DALI faults including reported Lamp failures and Missing fixtures. (NOTE: the DALI drivers must be able to report Lamp Failures. Not all drivers do this.)

To support these enhanced features, it is necessary to Initialise each channel, before use and after each repair. However, once the Initialisation has been started no further intervention is required except to verify the number of fixtures found during the initialisation is correct.

## DALI Broadcast Channel Types and Initialisation

The eDIN+ DALI Broadcast Output channel type can support several different types of DALI fixture:

* a conventional single channel DALI driver that drive (white) lamp/LED tape/etc.
* a 4-channel DALI driver that is used to drive up to 4 white lamps/LED tapes/etc.

All fixtures on a DALI Broadcast Output channel must be of the same type, but different channels can have different types.

Each DALI Broadcast Output channel must be initialised before it will work correctly. This initialisation process automatically commissions the fixtures so that they work correctly with our fault detection. Once the Initialisation has been started no further intervention is required except to verify the number of fixtures found during the initialisation is correct.

There are exceptions:

* A channel can be set to override or identify before being initialised.
* A channel that is set to conventional single-channel fixtures AND channel fault reporting is NOT required, does not need initialising.

Each time one or more fixtures are changed/repaired on a channel, that channel must be re-initialised to automatically commission the new fixtures before use.

A channel can be initialised in several ways:

* From the module’s SETUP menu. Navigate to SETUP->FIX->CHANx->INIT.  
  The display will show how many fixtures it finds and displays FDxx when the initialisation is finished. Press Forward (Right Hand) button to accept, or BACK (Left hand) button to reject.
* From the NPU’s web pages. Navigate to Monitor->Advance DALI Management page. Find the Module and Channels and use the Initialise Buttons. The fixtures will be displayed as they are found. The fixtures will be automatically accepted at the end of the Initialisation.

NOTE: for all 4-channel driver types the initialisation process sets a unique DALI short address for each driver, and spaces these addresses at every fourth address, e.g. at short addresses 0, 4, 8, etc. This means that there is a maximum of 16 drivers allowed on one channel (after which no more DALI short addresses are available).

This initial setup and initialisation cannot be performed via GATEWAY commands.

# Advanced DALI Operation

DALI universes in an eDin lighting system are accessed via Mode DALI channels, rather than native DALI operations. Mode DALI can be set to one of three operating modes: Broadcast, Standard (Group) or Advanced (Channel). In some of these modes, DALI broadcast and group operations are possible, as well as operations on the Mode DALI channels. These advanced DALI operations are also possible through the gateway.

The following advanced DALI identifiers can be used instead of the Mode DALI channel number in DALI commands and queries.

* BST – DALI Broadcast – all channels
* G00 to G15 – DALI Group – commissioned DALI groups
* F00 to F63 – DALI Fixtures (short address)

However, these identifiers will not work in all situations. This is so that these identifiers cannot get around various security features within the eDin system. It is also because using them in certain situations does not make sense.

Firstly, the advanced identifiers are converted to the equivalent Mode DALI channel number when possible.

* In Broadcast mode, BST -> Mode DALI channel 001.
* In Standard mode, G00 to G15 -> Mode DALI channels 001 to 016.
* In Advanced mode, F00 to F63 -> Mode DALI channel 001 to 064.

To aid debugging, the advanced identifiers are not replaced with Mode DALI channel numbers in the Gateway long acknowledgements, e.g.   
 !OK,DALISTOP,5,17,BST;   
only in the query replies and events, e.g.!DALISTOP,5,17,01;.

Secondly, identifiers only work in certain DALI operating modes.

* F00 to F63 only work when in Advanced mode
* G00 to G15 only work when in Standard and Advanced modes
* BST works in all modes: Broadcast, Standard and Advanced

Using them in the wrong operating mode is equivalent to specifying a non-existent channel and the operation will be ignored.

Thirdly, some queries such as Dali Channel status, ?DALI, can only be used against actual channels. If an advanced identifier does not convert to an actual Mode DALI channel, no reply will be generated in the same way as queries against non-existent channels.

# Advanced DALI Repair

Fixtures in a DALI universe need to be commissioned with correct settings in order to work correctly in an eDIN system. If a fixture fails and is replaced the new fixture needs to be programmed with the correct settings for it to work correct.

The eDIN lighting system continually monitors each DALI universe, and if it detects changes to a fixtures setting or if it detects missing or new fixtures it reports these problem as status error 22 - DALI Commissioning problem. In order to clear these problems the DALI universe needs to be repaired

## Identifying Faulty Fixtures

Fixture failures are reported using the same mechanism as normal channel errors and you monitor and identify fixture errors using the same methods as finding any other channel error. There are several ways to do this:

* Query each DALI fixture and inspect the status field.  
  e.g. ?DALIERR,1,17,F03; => !DALIERR,001,017,F03,<status>;
* Query for system errors and look for DALI fixture errors.  
  e.g. ?ERRORS; => !DALIERR,001,017,F03,<status>;
* Monitor (advanced) events for errors.  
  e.g. $EVTADV,1; => !DALIERR,001,017,F03,<status>;…

Depending on the configured runtime reporting mode of the UBC you will get a <status> status code of either 25 DALI Lamp failure or 26 DALI missing ballast or 22 DALI Commissioning Problem.

Note: Fixture errors will also be reported indirectly on the DALI channel that the fixture belongs to. For example if Fixture F03 reports a lamp failure and Fixture F03 is part of DALI channel 001, then DALI channel 001 will also report a lamp failure on its channel status. So one fixture error is likely to generate 2 reported errors; for our example,

!DALIERR,001,017,F03,025; !DALIERR,001,017,001,025;

You can decide if you want to log or report both errors, fixture errors only or dali channel levels only.

If the UBC is in Advanced mode there is a one-to-one association between the DALI channel and a DALI fixture, so fixture and DALI channel errors are interchangeable.

If the UBC is in Standard mode, there may be a number of fixtures in the channel’s associated DALI group. As a DALI channel can only report as single error code, if there are multiple fixtures that have faults, the channel will report missing fixtures before lamp failures.

## Step 1 – Perform a scan

Repairing a DALI universe involves a number of steps.

The first step is to perform a scan using the $DALISCAN command. This command can be used at any time – however normal DALI bus operations (e.g. $DALIFADE) are suspended until the scan is complete.

There are two ways to perform a scan. The first way is to issue commands and poll with queries to determine the progress of the operation. The alternative way is to turn on the advanced events and monitor the events generated by the operation. Both methods are described below.

To monitor the operation using events, advanced events must be turn on you begin the operation using the $EVTADV,1; admin command (or $EVENTS,1; that turns on all events).

First send the $DALISCAN command. Once the $DALISCAN command has been sent, the scan usually takes up to 30 sec to complete, however depending on the state of the DALI universe it could take up to 10 minutes or so.

To poll for the state of the scan use the ?DALISCAN query. This query returns the status of the scan – whether it is idle (0), it is searching (2), the search has completed successfully (1) or the search has ended in an error (4), and how many fixtures the search has found.

e.g.  
?daliScan,1,17;  
!OK,DALISCAN,001,017;<CR><LF>  
!DALISCAN,001,017,01,024;<CR><LF>

The $DALISCAN command will generate !DALISCAN events at the start and end of the scan that are the same format as in the ?DALISCAN query response.

e.g.  
$evtAdv,1; … $daliScan,1,17; …  
!DALISCAN,001,017,02,000;<CR><LF>  
…  
!DALISCAN,001,017,01,017;<CR><LF>

When the search has completed successfully, use the ?DALIFIX query to return the list of fixtures. The query returns a !DALIFIX message for each fixture it finds and for any missing fixtures it discovers. The list ends with a !DALIEND message.

(NOTE: There may be 2 !DALIFIX messages for the same short address - if the original fixture is missing, and a new fixture is found at the same short address.)

The !DALIFIX message reports the DALI fixture settings, such as long address, groups value and device type. The last field in the message reports the status of the fixture. Note: this fixture-status code is NOT the same as the channel and runtime status codes reported with the ?ERRORS and ?DALIERR queries although they do include codes for lamp failure and missing fixtures.

Fixtures that match the expected settings have status value 0 = OK or 1 = Lamp failure. These can be ignored as they do not need repairing.

Fixtures that are reported as 2 = missing, mean that the scan has not discovered an exact matching fixture. It may be for a number of reasons:

* the old fixture is unpowered or broken and so is not responding
* the old fixture has been removed
* that one or more of the settings in the old physical fixture have changed from the expected/commissioned values
* the old fixture has been replaced with a new fixture.

Fixtures that are reported missing need to be repaired – see example.

Fixtures that are reported as 5 = new mean that the scan has discovered a fixture but its settings do not match with any of the expected data. This may be for a number of reasons.

* the new fixture has been added to the system
* that one or more of the settings in the existing physical fixture have changed from the expected/commissioned values
* an old fixture has been replaced with a new fixture.

Fixtures that are reported as new are used in the repair process – see example.

Fixtures that are reported as 8 = address clash or 9 = unassigned mean that these fixtures do not have a unique short address so cannot be repaired. It is unlikely that you will see these fixture codes, as the normal $DALISCAN attempts to automatically fix these problems. You are only likely to see these if the scan has found and reported more than the maximum 64 fixtures on a single DALI universe, caused by DALI Universe wiring problems. You will need an electrician to sort out these wiring problems. You can perform further $DALISCAN to establish when the wiring problems have been resolved.

If you have enabled advanced events, the scan generates !DALIFIX messages for each fixture found (or found missing) in real time during the scan. The fixture list is finished with a !DALIEND message at the end of the scan, similar to the ?DALIFIX query.

NOTE: The !DALIFIX messages are generated as it finds the fixtures at irregular intervals. Do not be surprised if there are long gaps between some messages and short gaps between others.

## Step 2 – Identifying the correct new fixture to repair with

To repair a DALI universe you need to match missing fixtures with new fixtures. In order to do this you need to know the physical location of both the missing fixtures and the new fixtures. You can then match the missing and new fixtures in the same physical location.

The physical location of the missing fixtures must already be known and recorded during the original commissioning operation. The fixture could be recorded on a site plan, or be named appropriately. If the fixture is part of a DALI Group it may be possible to flash the DALI group using e.g. repeated $DALIFADE commands to locate the group, or by using the $SHOWDALI command.

The physical location of the new fixtures can only be determined by flashing each fixture in term and seeing the physical location of the flashing fixture. The easiest way to perform this is with the $SHOWDALI command, using the same Fxx identifier in the !DALIFIX message. See example below.

You can either identify the physical location of every new fixture as a single operation, recording their location for later use, or you can take each missing fixture in turn, and get the end-user to iterate through the new fixtures until they have found the correct one.

## Step 3 – Repairing a missing fixture with the matching new fixture

When you have identified which new fixture is in the same physical location as a missing fixture, you can repair the missing fixture using the $DALIREPAIR command.

The $DALIREPAIR command takes the Fxx identifier of the missing fixture and the Fxx identifier of its matching new fixture. It reprograms the new fixture with the correct settings of the missing fixture and updates the expected data in the configuration.

You repeat this step for every missing fixture.

You can use the ?DALIFIX query to obtain an updated list of fixture data or you can monitor the !DALIFIX events that are generated by the $DALIREPAIR commands and update your own list.

You can rescan the DALI universe at any point to be certain of the current status of the fixtures.

## Step 4 – Tidying the Expected Data

It may be that there is no new fixture to match a missing fixture, or no missing fixture to match with a new fixture. In other words, you have done all the repairing you want to do or can do and there is still missing and/or new fixtures present causing the status error 22 - DALI Commissioning problem to remain.

An alternative to repairing is to use the $DALIACCEPT command to import (or accept) the settings of the DALI fixture. Importing a missing fixture deletes that fixture from the commissioning data, and importing a new fixture adds the fixture and its current settings to the commissioning data.

NOTE: if there is both a missing and a new fixture with the same Fxx identifier, the single $DALIACCEPT command will both clear the missing fixture and add the new fixture at the same time, i.e. it performs the same as a $DALIREPAIR,x,x,Fnn,Fnn; when nn is the fixture short address.

NOTE: if a fixture is reported missing during a scan simply because it is unpowered, and you perform a $DALIACCEPT to remove it and clear error 26 DALI missing ballast. However, a new 22 DALI Commissioning Problem error will appear as soon when the fixture is repowered as the fixture is no longer in the commissioning data and looks like (and be reported as) a new fixture.

## Step 5 – End to repair session

When a UBC detects a scan or repair command it automatically stop checking for runtime problems (lamp failures or missing fixtures) so that it does not impede or get confused by the repair process. So it is important that when you have completed your repair session, you re-enable the runtime status monitoring by sending the $DALIDONE; command.

This clears any existing DALI errors and re-enables the checking. Any errors that are still present will be detected and reported in the usual manner.

## Step 6 – Back up the configuration

When you have completed the repair process and all fixtures are reported OK, it is time to make a back up of the configuration in the usual way.

## Example

The following shows the result of a DALI scan

$DALISCAN,1,17;  
!OK,DALISCAN,001,017;  
!DALISCAN,001,017,02,000;  
!DALIFIX,001,017,F00,04905615,00064,000,0;  
!DALIFIX,001,017,F01,53195828,00032,000,1;  
!DALIFIX,001,017,F01,14096720,00032,000,2;  
!DALIFIX,001,017,F02,08348761,00002,000,1;  
!DALIFIX,001,017,F03,21346988,00008,000,0;  
!DALIFIX,001,017,F04,18361283,00000,000,2;  
!DALIFIX,001,017,F05,00283700,00000,000,2;  
!DALIEND,001,017;  
!DALISCAN,001,017,01,005;

This shows that the scan was expecting 4 fixtures at short address 0,1,2 & 3. It has found the correct fixtures at address 0 and 3, but both fixtures 1 & 2 are missing.

This shows that the scan found 5 fixtures at short address 0, 1, 3, 4 and 5. The fixtures at 0 & 3 are expected. The fixtures at 1, 4 & 5 are new (or modified).

Let us take each missing fixture in turn, starting with fixture 1. We have both missing and new fixtures at short address 1. This might be because the old fixture has been replaced by a new fixture that has been correctly programmed (with the correct short address and groups) before installation. Note: the only difference between the 2 is the long address.

To confirm that the new fixture 1 is indeed the correct fixture, we use the $SHOWDALI and $SHOWOFF commands to start and stop the new fixture flashing

$showDALI,1,17,f1;  
!OK,SHOWDALI,001,017,F01;  
$showOff,1,17;  
!OK,SHOWOFF,001,017;

To repair fixture 1 we use the F01 identifier for both missing and new fixtures,

$daliRepair,1,17,f1,f1;  
!OK,DALIREPAIR,001,017,F01,F01;  
!DALIFIX,001,017,F01,14096720,00032,000,2;

Let us take the second and last missing fixture 2. There are now 2 possible new fixtures (4 & 5) to choose from to repair the missing fixture. We flash each fixture to determine which the correct one is.

$showDALI,1,17,f4;  
!OK,SHOWDALI,001,017,F04;  
$showDALI,1,17,f5;  
!OK,SHOWDALI,001,017,F05;  
$showOff,1,17;  
!OK,SHOWOFF,001,017;

We find that fixture 5 is in the correct physical location and we use this to repair fixture 2 with.

$daliRepair,1,17,f2,f5;  
!OK,DALIREPAIR,001,017,F02,F05;  
!DALIFIX,001,017,F02,00283700,00002,000,0;

We are now left with no missing fixtures but with one new fixture at short address 4. As there is nothing to match it with we need to accept/import this.

$daliAccept,1,17,f4;  
!OK,DALIACCEPT,001,017,F04;  
!DALIFIX,001,017,F04,18361283,00000,000,0;

A new scan confirms these changes

$DALISCAN,1,17;  
!OK,DALISCAN,001,017;  
!DALISCAN,001,017,02,000;  
!DALIFIX,001,017,F00,04905615,00064,000,0;  
!DALIFIX,001,017,F01,14096720,00032,000,0;  
!DALIFIX,001,017,F02,00283700,00002,000,0;  
!DALIFIX,001,017,F03,21346988,00008,000,0;  
!DALIFIX,001,017,F04,18361283,00000,000,0;  
!DALIEND,001,017;  
!DALISCAN,001,017,01,005;

All is well so we end the session and back up the revised configuration.

$DALIDONE;

Backing up the configuration cannot be done over the GATEWAY interface.

# Advanced DALI Emergency Lighting Support

It is possible to test DALI Emergency Lighting fixtures (device type 1) though the Gateway Interface. The Interface provides a subset of DALI commands and queries described in ref[2] to allow runtime test and monitoring. Configuring the type 1 fittings cannot be done through the Interface.

The eDIN lighting system does not itself test device type 1 features. It simply reports device type 1 fixtures in its $DALISCAN results, and allows third parties to perform testing via the Gateway interface.

All Gateway ‘EM’ commands and queries only work with device type 1 fixtures. Each Gateway command is implemented by sending a DALI ‘ENABLE DEVICE TYPE 1’ message (dali cmd 272) before any other DALI command or query.

The set of Gateway commands (almost) have a 1:1 correspondence with DALI commands:

* $EMREST - Put fixture in to Rest mode (dali cmd 224)
* $EMINHIBIT - Put fixture in to or out of Inhibit mode (dali cmd 225 / 226)
* $EMFUNC - Start a functional test (dali cmd 227)
* $EMDURA - Start a duration test (dali cmd 228)
* $EMSTOP - Stop a test (dali cmd 229)

There is also a command to identify Emergency fittings

* $EMSHOW – Identify EM (device type 1) fixture for 10 secs (dali cmd 240)

Note: This $EMSHOW command differs from the other $SHOWxxx commands: it is implemented by issuing a dali command 240, and it does not need a $SHOWOFF as it only lasts for 10 secs.

All the above commands work with any dali addressing mode, i.e. you can use BST, Gxx or Fxx identifiers with these commands.

There is a single Gateway query command to that combines 3 DALI queries.

* ?EMTEST - Query a fixture for test results -   
  <em-status> (dali query 253)  
  <em-failure> (dali query 252)  
  <em-duration> (dali query 243)

This query only works with individual fixtures (Fxx identifiers) for obvious reasons.

# Scene Setting

There are 2 ways to set the level of a scene. The simplest is *live* scene setting. The most powerful is *off-line* scene setting. Each method is completely different from the other.

### Live Scene Setting

This method allows you to change the levels of the existing channels defined for an existing scene. It does not allow you to change which channels are in the scene or the scenes default fade time. It does not allow you to define new scenes. You must also set the levels live, i.e. set the actual channels to the level you require. Then you save these levels.

First set the channels to the level you require. This is done by using the ‘Set channel to level’ type messages. Although not strictly necessary, it is recommended to set the channels to the existing scene levels before you start adjusting them. Do this using the $SCNRECALL command.

When all the channels in the scene are at the required level save these levels using the $SCNSAVE command. This command looks at the *live* level or state of each channel in the specified scene and saves these values in the scene definition. All subsequent scene recalls of this scene will use these new values.

Note:

* $SCNSAVE command only changes the scene definition if the scene allows editing (access level edit flag set).

It is not possible to exclude any channels from the $SCNSAVE operation, which is why it is recommended to set all channels and buttons to the current scene values by recalling the scene before you begin adjusting the levels. Then only the channels you have adjusted will change value.

Summary:

Step 1) Recall scene to set channels to current levels using $SCNRECALL (recommended).

Step 2) Adjust each channel level *live* as required using $CHANFADE, $DALIFADE.

Step 3) Save *live* levels using the $SCNSAVE command

### Off-line Scene Setting

This method allows you to define which channels are in an existing scene as well as their level. It does not change the live level of the channels during the process, so it allows you to define the channels and levels away from the live system then upload the new scene definition remotely. This method does not allow you to define new scenes (i.e. new scene numbers), although it does replace existing scene definitions with new definitions.

A full scene definition operation requires you to send a set of scene setting messages within a limited time period.

1. Send $SCNSET command. If you receive the !SCNSET acknowledge message it is safe to proceed. This internally clears the existing scene channels and levels.
2. (Optional) Send $SCNFADE command to set a new default value of scene fade time. If this command is not sent the existing default value is preserved. Note: not all built-in configurations use the default fade time of scenes when they perform a scene recall. So changing this value may not change how the internal system behaves. If this is the case, the fade time can only be changed by reconfiguration the system.
3. Send $SCNCHAN/DALI… commands for each channel in the scene. If you do not send a new $SCNxxx message for a channel it will not be in the new definition. If a channel is not acknowledged with a !SCNCHAN/DALI message the channel will not be in the new definition.
4. Send $SCNEND to complete the operation and save the new definition, or do nothing to abort the scene definition operation. The !SCNEND acknowledgment message is sent at the start of the scene save operation. No acknowledgement is sent if the operation is aborted.

Note:

* After the initial $SCNSET command, you must send the next message within 30 seconds (the scene setting timeout period). If no scene setting message is sent within this time the whole scene setting operation is aborted and the previous definition remains unaffected.
* Any scene setting message sent after an unacknowledged $SCNSET message, or after an aborted operation, is rejected/ignored, except for a new $SCNSET command that starts a new scene definition process.
* If a $SCNSET command is sent during an existing scene definition process, it is ignored. This is also true if a $SCNSET command is sent too early after a previous operation is to be aborted but the scene setting timeout has not yet expired.
* It is possible to abort an existing scene definition process by sending the $SCNABORT; command. The !SCNABORT acknowledgment message is sent if the abort was successful or if there is no scene setting process to abort. If there is no acknowledgement, there is an existing scene setting process and you have not been able to abort it.
* All scene setting commands must be sent on the same session. Any scene setting message sent from a different session is rejected during an existing scene definition process.

It is safe to send the complete set of commands without waiting for each acknowledgement. To verify if the scene definition operation was successful, you can either inspect the set of acknowledgement messages after the event, or you can send a scene definition query message ?SCNSET and inspect the results.

# System Discovery through the Gateway

It is expected that any third party application that uses the Gateway Interface knows about the specific EDIN system it is trying to control and monitor. It knows what scenes and channels exist and how to access them. There are tools in the Configuration Web pages to export this information.

However, the Gateway interface does have a few queries that allow limited discovery of an eDIN system at runtime. This discovery is centred around the Areas defined in the configuration. A list of Areas can be asked for together with the Scenes and wall Plates for an area. It is also possible to query the channels in a scene so that scene setting can be performed.

The discovery queries are ?AREANAMES, ?SCNNAMES and ?PLATENAMES. The first operation is to obtain a list of Areas using the ?AREANAMES; query. For each area the gateway returns a reply:

!AREANAME,<area-num>,<access>,<content>,<area-name>;

The important fields are the area-num and the area-name. The area-num is used by the other Gateway commands and queries when they need to refer to an Area. The area-name is usually how an end user will refer to an Area.

The content field declares if there are accessible scenes and plates (and channels) in the area. This could be use to decide if it is worth querying for these for this area. The content field is a bit-field and sets a different bit for scenes, plates and channels. Below is a table to decode the bit field.

|  |  |  |  |
| --- | --- | --- | --- |
| content value | has plates | has scenes | has channels |
| 0 | 🗶 | 🗶 | 🗶 |
| 1 | 🗶 | 🗶 | 🗹 |
| 2 | 🗶 | 🗹 | 🗶 |
| 3 | 🗶 | 🗹 | 🗹 |
| 4 | 🗹 | 🗶 | 🗶 |
| 5 | 🗹 | 🗶 | 🗹 |
| 6 | 🗹 | 🗹 | 🗶 |
| 7 | 🗹 | 🗹 | 🗹 |

The access field is not currently used and can be ignored.

Once a list of Areas has been obtained you can obtain a list of scenes, either for the whole system, if it is not very larger or for just a particular area, using the scene names query: ?PLATENAMES; or ?PLATENAMES,<area-num>;. Each plate returns a reply:

!PLATENAME,<addr>,<devcode>,<style>,<access>,<area-num>,<plate-name>;

The important fields are the addr, devcode, and the plate-name. The addr and devcode is used by the other Gateway commands and queries when they need to refer to a Plate. The plate-name is usually how an end user will refer to a Plate.

The access field declares what you can do with the plate through the Gateway. It is a bit-field that has a different bit for the following operations:

* viewable – you can query and monitor the plate and its buttons
* controllable – you can perform button presses using $BTNSTATE
* editable – there are no editable operations - ignore

See the access table above to decode the bit field.

The style field declares what the plate looks like and you can use this to display the plate in different ways depending on style. The possible style values returned are shown in the table below.

|  |  |
| --- | --- |
| style value | Description |
| 0 | Unknown style |
| 1 | SGP plate style |
| 2 | Coolbrium plate style |
| 3 | Icon plate style |

If you are performing scene setting it is useful to know the names of the channels being adjusted. You can obtain these using the Channel names query: ?CHANNAMES,<scn-num>;. Each channel returns a reply depending on channel type:

!CHANNAME,<addr>,<devcode>,<chan-num>,<access>,<area-num>,<chan-name>;

or  
!DALINAME,<addr>,<devcode>,<dali-num>,<access>,<area-num>,<chan-name>;  
or  
!DMXNAME,<addr>,<devcode>,<zone-num>,<access>,<area-num>,<zone-name>;

The important fields are the addr, devcode and chan-num/dali-num that are used by the other Gateway commands and queries when they need to refer to a Channel, and the chan-name that is usually how an end user will refer to a Channel.