MODE LIGHTING
CONTROLLING THE FUTURE OF LIGHTING

Title:	Gateway Interface Volume 3 - Developer
Version:	2.0.3
Author:	Jeff Heine
Date:	08 March 2022

History

Date	Ver	Initials	Changes
24-May-19	2.0.0	Jph	Initial draft
11-Jun-19	2.0.1	Jph	Document: HTTP header ModeLightingTimeout
14-Dec-21	2.0.2	jph	Modification for Gateway Ver 2.01
1-Mar-22	2.0.3	Jph	Modifications for Gateway Ver 2.02 Add External control section and `$MASTERTICK` command Add Offline scene setting section

Table of Contents

1 Introduction

1.1 Purpose and scope

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

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

1.2 Terminology

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

1.3 References

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

2 Overview of developer features

This Volume 3 document describes features of the eDIN+ GATEWAY interface that only software developers are likely to need. Some provide the same facilities that are available via the administrator pages of an eDIN+ system. Others provide access that allows the eDIN+ system to be extended. It assumes the reader is familiar with the eDIN+ system, and familiar with using the GATEWAY interface and the topics covered in Volumes 1 & 2.

This document is written as a reference manual, rather than an application guide. It has chapters covering facilities of the eDIN+ system and GATEWAY. Developers may pick and mix these facilities as required.

These developer facilities work closely with the eDIN+ internal workings and many have side-effects on the operation of an eDIN+ system. As such they require extra care when used. If you intend to use these facilities, please contact Mode Lighting to discuss your application, to ensure that there are no unintended consequences for your application. This is in contrast to the topics in Volume 1 & 2 that do not require the same level of care when using.

As these facilities require extra care, they require the connection to have *administrative privileges* or above. Connections automatically have *administrative privileges* if user accounts are not being used. Otherwise the connection must use an account that has the appropriate privileges. Refer to Volume 2 for further information on user accounts and connections.

2.1 Developer Features

2.1.1 The Info web service and csv info files.

This is a web service separate from the GATEWAY interface. It provides fuller access to eDIN+ configuration information than can be obtained using GATEWAY discovery queries. It allows an application to import and export the Comma-Separated-Variable formatted information files that are available manually via the configuration editor web pages or offline editor. The files include a full list of modules and channels, as well as a full list of scenes and their scene setting information.

2.1.2 Offline Scene Setting

Volume 1 describes how it is possible to permanently change the channel levels in a scene using the `$SCNSAVE` command, that saves the current *live* level of the channels. This has a number of limitations:

i) You cannot adjust a scene silently or in the background – the channels must be set to the desired level and this will be visible to anyone.

ii) You cannot adjust any DALI *virtual channel* (`BST` or `Gxx` channels) as these channels do not have a level state that can be read back by the `$SCNSAVE` command.

iii) You cannot change which channels are in a scene

There are a set of commands that allow you to adjust individual scene elements that overcome these limitation.

2.1.3 External Control through GATEWAY

It is possible for an external system to be the primary controller of an eDIN+ system, and control the system via the GATEWAY interface, but have a back-configuration in the eDIN+ system that kicks in when it detects the primary controller is absent, perhaps due to a lost connection. There is a section in this topic that explains what you need to do in this arrangement.

2.1.4 Advanced Channel API

There are a number of commands and queries that are useful for advanced operations, but are unlikely to be used otherwise, and can disrupt normal eDIN+ system operation. These include Channel identification and test, Module discovery, and DALI fixture health queries.

2.1.5 Advanced DALI Repair – fixing DALI commissioning errors

DALI fixtures need to be programmed or commissioned to work in the eDIN+ system. If an existing fixture breaks and is replaced the new fixture needs to be programmed or re-commissioned to work properly within the system. The eDIN+ system works hard to hide the details of this from users, but does generate a status error code `22 - DALI Commissioning problem` when it detects a problem.

There are a set of commands and queries that allows a third party to fix the system and clear error code `22`, by finding and repairing the relevant fixtures.

2.1.6 Advanced Sensor repair

To be completed

2.1.7 The XDALI API – the DALI back door

The eDIN+ system works hard to hide any DALI bus that exists in the system and does not support the full range of features available on DALI. The XDALI set of commands and queries provides a way to send any DALI message and perform any DALI operation directly on a DALI bus bypassing an eDIN+ system entirely.

3 The Info web service and csv files.

The Info web service is separate from the GATEWAY interface. It provides fuller access to eDIN+ configuration information than can be obtained using GATEWAY discovery queries.

Some of the uses for this web service include:

- It can be used for system discovery - to obtain lists of modules and channels, as well as lists of scenes and their scene setting information.
- It can be used for offline scene setting – by importing the scene setting information back in to the eDIN+ system.

The eDIN+ system can export and import configuration information to and from text files conforming to the Comma-Separated-Variable format. This can be done manually via the configuration editor web pages or offline editor, or via this web service.

3.1 Accessing the web service

The Info web service is through an http connection uses plain text AJAX-like HTTP GET and POST requests to the URL

```
      http://<npu_ip_address>/info?<parameters>
```

using the `Content-Type: application/csv`.

Further details are given in the sections following this.

3.2 Exporting configuration information

Exporting to obtain information from the eDIN+ system needs *administrator privileges*. This is automatically given if no user accounts are set up. If user accounts have been set up you need to send username and password of an administrator account using the HTTP BASIC Authorization mechanism – see Volume 2 Users.

To retrieve configuration information use GET requests. There are three different lists you can retrieve. To specify which type add the following (mandatory) parameter to the URL:

- `what=names` - list of modules and their channels
- `what=levels` - list of scenes and the channels they control
- `what=dali` - list of DALI buses and their fixtures

Each list has optional parameters that can filter the information returned.

- `what=names`
  Return information for all *areas*, *plates* and *modules*.
- `what=names&foraddess=<addr-list>`
  Only return information for *plates* and *modules* with the MBus addresses specified. Separate each MBus address with a comma character.
- `what=names&fordevicecode=<devcode-list>`
  Only return information for *plates* and *modules* with the device codes specified. Separate each device code with a comma character.
- `what=names&foraddess=<addr-list>&fordevicecode=<devcode-list>`
  Filter *plates* and *modules* for both address and device code.

- `what=levels`
  Return information for all *areas* and all *scenes*.
- `what=levels&forarea=<area-num-list>`
  Only return *area* and *scene* information for the areas specified.
  Separate each area number with a comma character.
- `what=levels&forscene=<scene-num-list>`
  Only return scene information for *scenes* with the scene numbers specified.
  Separate each scene number with a comma character.
- Do *not* use both `forarea` and `forscene` together. They do not work together. The `forscene` option takes precedence over `forarea`.

- `what=dali`
  Return DALI fixture commissioning data for all *DALI universes*.
- `what=dali&foraddess=<addr-list>`
  Only return DALI fixture commissioning data *DALI universes* with the MBus addresses specified. Separate each MBus address with a comma character.

For all lists there are the following optional global parameters that can be added

- `&where=<filename>`
  This `where` parameter is useful when embedding the URL in to HTTP web pages to perform *file* downloads and save the data as a file.
  The filename is returned in the HTTP response header as
  `Content-Disposition: attachment; filename="<filename>.csv";`
  This response header is usually ignored and not processed by most HTTP stacks, but browsers and other applications do notice that the response data has been labelled as an attachment and will use the `<filename>` to automatically save the data as that file.

Example in JQuery:
```
$.ajax({
      type: 'GET',
      url: 'http://'+ipaddr+'/info?what=names&foraddress=6,17',
      contentType: 'application/csv; ',
      dataType: 'text',
      username: 'Administrator',
      password: 'use the proper password here',
    })
```

3.3 Importing configuration information

Importing data using this facility is always considered as *editing* the configuration. As such all import operations change the `edit-stamp` of the `SYSTEMID` – see Volume 1 System Discovery.

Importing will change the configuration in the eDIN+ system, so you *MUST* supply the eDIN+ system's *configuration password* using the HTTP BASIC Authorization mechanism. This means *user accounts MUST be setup* to use this import facility. This password protection is to stop accidental changes that can potentially wipe out a large part of the configuration.

To change configuration information use POST requests. There are three different types of data you can change. To specify which type add the following (mandatory) parameter to the URL:

- `what=names` – change module and channel names and areas
- `what=levels` - perform scene setting
- `what=dali` – re-registers commissioned (expected) DALI fixtures and their settings

Example in JQuery:

```
$.ajax({
      type: 'POST',
      url: 'http://'+ipaddr+'/info?what=names',
      data: '!EDIN NAMES FILE\nAREA,1,First Area\n'
      contentType: 'application/csv; ',
      dataType: 'text',
      username: 'configuration',
      password: 'use the proper password here',
    })
```

The best way to learn the correct format for the data you are importing is to export the relevant data, modify the appropriate fields, then re-import it. However you can create and import only the data you want to change.

Here are some things that apply to all imported data

1) Data conforms to the Comma-Separated-Variable format. Each item must be on separate lines (using the CR, LF or CR-LF to separate items/lines), and each item has properties or fields separated by the ‘,’ comma character.
2) The *first item* (line) must identify the type of data by starting with the specific fixed text identification. Text after the proper identification is ignored.
3) If an item *begins with* the (the first character of a line is the) ‘!’ character, the item and rest of line is treated as a *comment* and ignored.
4) Blank lines are allowed, e.g. for readability by a human, and ignored.
5) Whitespace characters within a line (space, tab, etc) are *NOT* automatically removed. You must not pad an item with whitespace for better readability.
6) The *first field* of an item is a fixed text *token* that identifies the type of item.
7) Each item type has its own format the defines the number of fields/properties for that item.
   The token is *case-insensitive* – it can be any combination of upper and lower case for the necessary letters.
   Numeric fields must exist and contain at least one numeric digit 0-9. Numbers are allowed to have leading zeros, (and start with a plus or minus sign), but are *not* allowed to have leading spaces.
   Text fields/strings are *case-sensitive* – you get what you specify, including whitespace characters. Text fields can be blank and contain no characters.
8) There are *no set rules* for which item property fields are used to import *new values* and change the configuration, and which are only used to identify existing items within the configuration to modify.
   An item could be used to define a completely new item in the configuration, or it could be used to identify an existing item but modify one or more of its properties.
   Refer to the specific file formats below.
9) Items that have all the (required) fields described in their format are recognised and accepted. Any additional text/fields *after* the requied fields are ignored.

10) Items that are not recognised, or not allowed to be imported or changed, are simply ignored.
11) The syntax and values for each item closely follows the equivalent or similar GATEWAY discovery and status queries where they exist.

3.3.1 Names csv file format

The data MUST begin with the first line of text: `!EDIN NAMES FILE`

It is not possible to define new items in the configuration. The following existing items can be modified:

Item	Format	
`PROJECTNAME`	`PROJECTNAME,`**`<name-string>`**	Change the configuration Project name text
`PROJECTVERSION`	`PROJECTVERSION,`**`<version-string>`**	Change the configuration Project version text
`PROJECTOWNER`	`PROJECTOWNER,`**`<owner-string>`**	Change the configuration Project owner text
`AREA`	`AREA,<area-no>,`**`<name-string>`**	Change an existing Area's name text
`PLATE`	`PLATE,<addr>,<devcode>,`**`<area-no>,<name-string>`**	Change an existing Plate's area number and name text
`MODULE`	`MODULE,<addr>,<devcode>,`**`<area-no>,<name-string>`**	Change an existing Module's area number and name text
`CHAN`	`CHAN,<addr>,<devcode>,<chan-no>,`**`<area-no>,<name-string>`**	Change an existing output channel's area number and name text
`DALI`	`DALI,<addr>,<devcode>,<dali-no>,`**`<area-no>,<name-string>`**	Change an existing DALI output channel's area number and name text
`DMX`	`DMX,<addr>,<devcode>,<zone-no>,`**`<area-no>,<name-string>`**	Change an existing DMX zone's area number and name text
`INPSTATE`	`INPSTATE,<addr>,<devcode>,<chan-no>,`**`<area-no>,<name-string>`**	Change an existing contact input channel's area number and name text
`INPPIR`	`INPPIR,<addr>,<devcode>,<chan-no>,`**`<area-no>,<name-string>`**	Change an existing PIR input channel's area number and name text
`INPLEVEL`	`INPLEVEL,<addr>,<devcode>,<chan-no>,`**`<area-no>,<name-string>`**	Change an existing analogue input channel's area number and name text

Example Data:
```
!EDIN NAMES FILE

!the next line changes the project name
PROJECTNAME,Example Project

!the next line changes the name for area no 1
AREA,1,First Area

!the next lines move the first two DALI channels on UBC at address 3 in to
area no 1 and change their name
DALI,3,17,1,1,DALI Channel 1
DALI,3,17,2,1,Main Downlights
```

3.3.2 Levels csv file format

The data MUST begin with the first line of text: `!EDIN LEVELS FILE`

It is not possible to define new items in the configuration. The following existing items can be modified:

Item	Format	
`AREA`	`AREA,<area-no>,`**`<name-string>`**	Change an existing area's name text
`SCENE`	`SCENE,<scene-no>,`**`<name-string>`**	Change an existing scene's name text
`SCNFADE`	`SCNFADE,<scene-no>,`**`<fadetime-ms>`**	Change an existing scene's fade time in milliseconds
`SCNCHANLEVEL`	`SCNCHANLEVEL,<scene-no>,<addr>, <devcode>,<chan-no>,`**`<level>`**	Change an existing scene channel's level
`SCNDALILEVEL`	`SCNDALILEVEL,<scene-no>,<addr>, <devcode>,<dali-no>,`**`<level>`**	Change an existing scene DALI channel's level
`SCNDMXLEVEL`	`SCNDMXLEVEL,<scene-no>,<addr>, <devcode>,<zone-no>,`**`<level>`**	Change an existing scene DMX zone's level

`SCNCHANRGBCOLR` `SCNDMXRGBCLOR`	`SCNCHANRGBCOLR,<scene-no>,<addr>, <devcode>,<chan-no>,`**`<colour>`**
	
	`SCNDMXRGB,<scene-no>,<addr>, <devcode>,<zone-no>,`**`<colour>`**
		Change an existing scene channel/zone's colour, `preset` value or `#rrggbb` colour

`SCNCHANRGBPLAY` `SCNDMXRGBPLAY`	`SCNCHANRGBPLAY,<scene-no>, <addr>,<devcode>,<chan-no>,`**`<seq>`**
	
	`SCNDMXRGBPLAY,<scene-no>,<addr>, <devcode>,<zone-no>,`**`<seq>`**
		Change an existing scene channel/zone's sequence value

`SCNCHANTWCOLR` `SCNDMXTWCOLR`	`SCNCHANTWCOLR,<scene-no>,<addr>, <devcode>,<chan-no>,`**`<kelvin>`**
	
	`SCNDMXTWCOLR,<scene-no>,<addr>, <devcode>,<zone-no>,`**`<kelvin>`**
		Change an existing scene channel/zone's colour temperature, `preset` value or `#kelvinK`

Example Data:
```
!EDIN LEVELS FILE

!the next line changes the fade time to 1 sec for scene no 3
SCNFADE,3,1000

! the next lines change the levels for scene no 5
SCNCHANLEVEL,5,1,12,3,255
SCNDMXLEVEL,5,2,15,1,255
SCNDMXRGBCOLR,5,2,15,1,#FF0080
```

3.3.3 Dali csv file format

The data MUST begin with the first line of text: `!EDIN DALI COMMISSIONING FILE`

It is possible to perform the following:

- Completely clear out existing commissioning data for an existing DALI universe, and import new commissioning data for that universe.
- Modify an existing DALI fixture (on an existing DALI universe)
- Add new DALI fixtures to an existing DALI universe.

It is not possible to delete an existing DALI fixture except by deleting the data for the whole DALI universe and re-importing the whole universe without the desired fixture.

Note:

- This data defines or modifies the data associated with the *expected fixture list* with an eDIN+ system. It does not reprogram or commission the actual physical fixtures.
- Keep all `BALLAST` fixture items/lines for the same DALI universe together. Do not interleave `BALLAST` items for different DALI universe.
- Use the `DALIUNIVERSE` item *before* any `BALLAST` items for that universe, as the `DALIUNIVERSE` item deletes all existing data (including that comes before it).

Item	Format	
`DALIUNIVERSE`	`DALIUNIVERSE,<addr>,<devcode>`	Delete all existing fixture commissioning data for an existing DALI universe
`BALLAST`	`BALLAST,<addr>,<devcode>, <short-addr>,`**`<long-addr>, <type>,<groups>`**	Define a new fixture for an existing DALI universe, - or - Change the type, groups and long address values of an existing fixture.

Example Data:

```
!EDIN DALI COMMISSIONING FILE

! the next item deletes all DALI commissioning data on UBC 3,17
DALIUNIVERSE,3,17
! the next items redefine 5 new fixtures for the universe
BALLAST,3,17,0,2109473,0,1
BALLAST,3,17,1,7811888,0,2
BALLAST,3,17,2,12129130,0,4
BALLAST,3,17,13,10722243,0,8
BALLAST,3,17,24,6492078,0,16
```

4 Offline Scene Setting

Offline scene setting does not rely on live channel levels. It allows you to redefine a scene's definition including

- its fade time,
- which channels are in the scene, and
- the level or state of these channels.

It is a powerful tool as you can completely change all aspects of a scene. But this does mean that the process is more complex than the simple `$SCNSAVE` method and it is also possible to really mess a scene up if care is not given.

Although this method is more complex than the live `$SCNSAVE` method it has a number of advantages:

i) As this method does not rely on live channel levels, you can define the channels and levels away from the live system then upload the new scene definition remotely.
ii) You can adjust channels that do not have a live state or level, specifically DALI *virtual channels* (`BST` or `Gxx` channels).
iii) You can change which channels are in a scene.
iv) As this method does not rely on live channel levels, you can adjust a scene almost silently and in the background – the live level of the channels are not change.

There are a few limitations and disadvantages:

i) You cannot create new scenes (i.e. new scene numbers), but it does allow you to replace existing scene definitions with completely new definitions.
ii) You must always send the complete scene definition each time. You must send the complete channel list and their levels / states every time, not just those channels that change their value. This means that you need to know the complete existing channel list and their assigned values within the existing scene definition, to make any adjustments on even just one channel.
iii) The scene set operation must be completed within a finite time, and there is a chance the operation will fail. See below for further details.

4.1 Offline Scene Set Overview

To set a scene using this method, a set of commands is sent that define a scene: its fade time, the channels within the scene, together with their level / state / colour. To implement this method successfully, this set of commands must be applied to the system in a single operation. To aid this the set of commands is sent framed with a `$SCNSET` command at the start and a `$SCNEND` command at the end.

Things to be aware of

- Like `$SCNSAVE`, using `$SCNSET` is an *adjustment* and will change the `adjust-stamp` in `?SYSTEMID` query.
- Like `$SCNSAVE`, this method only changes the scene definition if the scene allows editing (access level edit flag set). Also, only controllable channels can be have their state /level / colour adjusted, and only editable channels can be added or deleted from a scene – see Volume 2 about scene and channel accessibility.

- Like `$SCNSAVE`, scenes do *not* have to include *both* level and colour entries for colour or tuneable white channels.
- Like `$SCNSAVE`, you can only change existing scenes. You cannot define new scenes (new scene numbers).
- Unlike `$SCNSAVE` to adjust a scene you need to know and send the information for all channels in the scene, not just the channels that change.
- Unlike `$SCNSAVE` **this `SCNSET` operation can fail** and not adjust the scene. You need to check and handle this if it is important to you – see below for further information.
- Unlike `$SCNSAVE` you can add and delete channels from the scene.

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. The scene must already exist in the NPU's configuration. Of course you can define any number of scenes outside of the eDIN+ system, and send in the appropriate channel level commands via the GATEWAY.

As this method redefines which channels are in a scene, the set of commands MUST include a command for every channel in the scene, even if a channel is not being adjusted. A command for the scene's fade time need only be included if the fade time is being changed, otherwise the fade time remains unchanged.

If no accounts have been set up in the NPU's configuration then all `SCNSET` commands will be actioned as all scenes and channels have editable access. A successful `$SCNSET` command list will delete all existing channels and (providing the channels exist) replace them with the new channel definitions.

If user accounts have been set up in the NPU's configuration (see Volume 2 Users & Access), then any user can use this method, i.e. these commands and queries are NOT restricted *administrator* accounts only. However, the scene must be editable and the channels controllable or editable

- Un-editable scenes are invisible to `SCNSET` commands and queries. They are ignored by the system and will not generate `$SCNEND` and `?SCNSET` replies.
- Un-editable channels cannot be deleted from a scene. They will persist in the scene's definition even when not present in the `SCNSET` command list.
- Un-editable channels cannot be added to a scene. The respective `SCNSET` command will be ignored.
- Un-controllable channels cannot have their level / state / colour changed. The respective `SCNSET` command will be ignored.

The scene set operation is almost invisible to the end user. It does not change the live level of the channels during the process. But it is not completely invisible as the operation invalidates the scene's state, so plate buttons and others monitoring the scene state may be affected.

As the GATEWAY interface supports multiple simultaneous connections, it is technically feasible that a scene is sent two new definitions simultaneously. To stop these 2 new definitions being merged, under the hood the `$SCNSET` command also acts as a request for the scene's internal *token*. Only 1 connection can hold a scene's *token* at any one time, and if a connection is not granted the token, then its scene set commands are ignored. To avoid a connection locking out all other connections, the token is granted for a limited time and automatically expires if the connection does not explicitly finish the scene set operation. This means that you must complete a scene set operation in a finite time, and there is an outside possibility the scene set operation will fail / be ignored.

This internal token handling does not normally need to be considered. However you need to be aware of it for trouble shooting when things go wrong, and when building robust systems that have exception handling.

4.2 `SCNSET` Method

To perform adjustments to an existing scene the first step is to request the current definition using the `?SCNSET` query and optionally the `?SCNSETNAMES` query. The `?SCNSETNAMES` query returns a list of channel elements within the scene and their names. This could be useful when presenting the channel list to an end-user. The `?SCNSET` query returns a list of channel elements that are in the scene and the level / state / colour that they are set to by the scene. It also returns the scene's fade time. You can then modify this channel element list as desired and return it using the set of `SCNSET` commands. If you want to confirm the `SCNSET` operation, you can inspect the response to the commands to check that the operation was successful. Alternatively you can perform a second `?SCNSET` query and check that the returned list is as expected.

The set of `SCNSET` commands should be sent as a single transaction. If using the HTTP web service you should send the set of commands in a single POST request; the POST response will include the `!SCNSETACK` acknowledgment message to let you know that the `SCNSET` transaction is now complete and if the `SCNSET` operation was performed.

A full `SCNSET` scene definition operation requires you to send a set of scene setting messages as a single transaction within a limited time period.

1. Send `$SCNSET` command to begin the transaction.
2. (Optional) Send `$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 `$SCNCHAN/DALI/DMX`… commands for each channel in the scene. If you do not send a new `$SCNxxx` message for a channel it will not be in the new definition.
4. Send `$SCNEND` to complete the transaction and save the new definition. The `!SCNSETACK` acknowledgment message is returned when the scene save operation is complete and lets you know if the operation was performed.

Note:

- If you are not using HTTP web service and are using one of the long-lived connections, after the initial `$SCNSET` command you must send the next `SCNSET` message within 30 seconds (the scene setting timeout period) until `$SCNEND` or `$SCNABORT` is sent. If no `SCNSET` message is sent within this time the whole `SCNSET` operation is aborted and the previous definition remains unaffected.
- All scene setting commands within a transaction must be sent on the same session. Any scene setting message after the initial `$SCNSET` command sent from a different session is

rejected. If using the HTTP web service each POST request has its own session, so you must send all commands within a single POST request.

- Only one *active* `SCNSET` transaction is allowed per scene. If a `$SCNSET` command is sent during an on-going transaction on another session, the whole transaction on this session will fail and be reported as such by the response to the terminating `$SCNEND` command.
- It is possible to abort all current transactions on a session by sending the `$SCNABORT;` command. If all goes well this command should not be needed.
  However this is useful on long-lived connections/sessions to tidy up the session if things have gone wrong and previous transactions have not been terminated successfully. On long-lived connections it may be prudent to prepend this command to all `SCNSET` transactions.
  On short-lived HTTP connections this command is not necessary, as each POST request opens a new session so there will never be previous transactions still open.

If you want to confirm that the `SCNSET` operation was successful you can inspect the response to each individual command in the transaction. However it may be more practical to simply re-query the scene definition with `?SCNSET` and compare the returned list against what is expected. You should only send the `?SCNSET` query after you have received the `!SCNSETACK` acknowledgment from the `SCNSET` operation. If using the HTTP web service, the `?SCNSET` query should be in a separate POST request.

Note:

- The `!SCNSETACK` acknowledgment message is sent in response to the terminating `$SCNEND` command. It is sent on completion of any scene save operation and completes the `SCNSET` transaction.
  It also contains a `status` value. This `status` value indicates if the transaction was able to obtain the internal *token* and perform the scene save operation or not. It does *NOT* indicate that all channel messages were successfully processed and saved.
- Each `$SCNxxx` channel message is processed independently. If there is an error in the message or the channel does not exist, then the channel will not be added to the scene. However the rest of the transaction will be performed and not aborted by this error. If you want to confirm that all channels have been successfully saved, you should re-query the scene definition.
- It is only possible to set a channel element level/state/colour via the GATEWAY. The channel names reported by the `?SCNSETNAMES` query are for information purposes only. They are fixed by the configuration and cannot be changed via the GATEWAY.

4.3 Scene Set Commands

Refer to the notes in the previous section `SCNSET` Method in conjunction with these commands. The `SCNSET` channel commands mirror direct output channel commands detailed in the section Channel API in Volume 1.

- Define a scene (time limited)
  `$SCNSET,<scn-num>;`
  Long-ack Format: `!OK,SCNSET,<scn-num>;`
  From: v02.02

  followed by zero or more 'Set Scene Item'

From: v02.02

(optional)`$SCNFADE,<scn-num>,<fadetime(ms)>;` or
Long-ack Format: `!OK,SCNFADE,<scn-num>,<fadetime(ms)>;`

Set channel to level
`$SCNCHAN,<scn-num>,<addr>,<devcode>,<chan-num>,<level>;` or
`$SCNDALI,<scn-num>,<addr>,<devcode>,<chan-num>,<level>;` or
`$SCNDMX,<scn-num>,<addr>,<devcode>,<zone-num>,<level>;` or
Long-ack Format: `!OK,SCNxxx,<scn-num>,<addr>,<devcode>,<chan/zone-num>,<level>;`

set DALI virtual channels to level
`$SCNDALI,<scn-num>,<addr>,<devcode>,BST,<level>;` or
`$SCNDALI,<scn-num>,<addr>,<devcode>,G<xx>,<level>;` or
Long-ack Format: `!OK,SCNDALI,<scn-num>,<addr>,<devcode>,<dali-id>,<level>;`

Set channel static pre-set colour
`$SCNDMXRGBCOLR,<scn-num>,<addr>,<devcode>,<zone-num>,<preset>;` or
`$SCNCHANRGBCOLR,<scn-num>,<addr>,<devcode>,<chan-num>,<preset>;` or
Long-ack Format: `!OK,SCNDMXRGBCOLR,…; or !OK,SCNCHANRGBCOLR,…;`

Set channel direct static colour
`$SCNDMXRGBCOLR,<scn-num>,<addr>,<devcode>,<zone-num>,#<wrgb>;` or
`$SCNCHANRGBCOLR,<scn-num>,<addr>,<devcode>,<chan-num>,#<wrgb>;` or
Long-ack Format: `!OK,SCNDMXRGBCOLR,…; or !OK,SCNCHANRGBCOLR,…;`

Set channel to play colour sequence
`$SCNDMXRGBPLAY,<scn-num>,<addr>,<devcode>,<zone-num>,<preset>;` or
`$SCNCHANRGBPLAY,<scn-num>,<addr>,<devcode>,<chan-num>,<preset>;` or
Long-ack Format: `!OK,SCNDMXRGBPLAY,…; or !OK,SCNCHANRGBPLAY,…;`

Set channel to pre-set colour temperature
`$SCNCHANTWCOLR,<scn-num>,<addr>,<devcode>,<chan-num>,<preset>;` or
`$SCNDMXTWCOLR,<scn-num>,<addr>,<devcode>,<zone-num>,<preset>;` or
Long-ack Format: `!OK,SCNDMXTWCOLR,…; or !OK,SCNCHANTWCOLR,…;`

Set channel to direct colour temperature
`$SCNCHANTWCOLR,<scn-num>,<addr>,<devcode>,<chan-num>,#<kelvin>K;` or
`$SCNDMXTWCOLR,<scn-num>,<addr>,<devcode>,<zone-num>,#<kelvin>K;` or
Long-ack Format: `!OK,SCNDMXTWCOLR,…; or !OK,SCNCHANTWCOLR,…;`

followed by 'End of scene definition'
`$SCNEND,<scn-num>;`
Long-ack Format: `!OK,SCNEND,<scn-num>;`
Reply: 'Report scene set scknowledge'
`!SCNSETACK,<scn-num>,<status>;`
`status` = `1` for `SCNSET` operation performed, `0` for operation not performed

Only editable scenes (Edit access bit set) can be defined.

Only controllable channels (Control access bit set) can be included in the scene definition. View-only channels are left unaffected by the scene set operation.
The set of messages must be sent as a single transaction. On long-lived connections each message must be sent within 30 seconds of the previous message. Otherwise the whole scene definition process is aborted. If using the HTTP web service the set of messages must be in a single POST request.

Note: a scene setting operation can be aborted by sending
'Abort all active scene definitions on session'
`$SCNABORT;`
Long-ack Format: `!OK,SCNABORT;`
From: v02.02

e.g.
```
$scnAbort;        (optional tidy) (only on long-lived connections)
$scnSET,3;
$scnFade,3,10000;
$scnChan,3,2,21,5,255;
$scnChanRgbColr,3,2,21,5,#ff7f00;
$scnChanTwColr,3,2,21,2,#2200K;
$scnDALI,3,3,17,12,200;
$scnDMXRgbColr,3,4,15,1,4;
$scnEnd,3;
!OK,SCNABORT;<CR><LF>
!OK,SCNSET,00003;<CR><LF>
!OK,SCNFADE,00003,00010000;<CR><LF>
!OK,SCNCHAN,00003,002,21,005,255;<CR><LF>
!OK,SCNCHANRGBCOLR,00003,002,21,005,#FF7F00;<CR><LF>
!OK,SCNCHANTWCOLR,00003,002,21,002,#2200K;<CR><LF>
!OK,SCNDALI,00003,003,17,012,200;<CR><LF>
!OK,SCNDMXRGBCOLR,00003,004,15,001,004;<CR><LF>
!OK,SCNEND,00003;<CR><LF>
!SCNSETACK,00003,1;<CR><LF>
```

4.4 Scene Set Queries

You can use the system discovery messages described in Volume 1 to obtain information about the scenes including `?SYSTEMID`, `?AREANAMES` and `?SCNNAMES` queries. In addition there are two new queries `?SCNSETNAMES` and `?SCNSET` for channel names and channel states associated with `SCNSET` operations

4.4.1 Scene definition query

- Request scene definition
  This is used to support scene setting operations. It returns a list of channel elements that are in the scene and the level/state/colour that they are set to by the scene. It also return the scene's fade time.

  From: v02.02
  `?SCNSET,<scn-num>;`
  Long-ack Format: `!OK,SCNSET,<scn-num>;`
  Reply: start of scene definition
  `!SCNSET,<scn-num>;`
  followed by zero or more 'Report Scene Item'

Scene fade time
`!SCNFADE,<scn-num>,<fadetime(ms)>;` or

Set channel to level
`!SCNCHAN,<scn-num>,<addr>,<devcode>,<chan-num>,<level>;` or
`!SCNDMX,<scn-num>,<addr>,<devcode>,<zone-num>,<level>;` or
`!SCNDALI,<scn-num>,<addr>,<devcode>,<dali-num>,<level>;` or
set DALI virtual channels to level
`!SCNDALI,<scn-num>,<addr>,<devcode>,BST,<level>;` or
`!SCNDALI,<scn-num>,<addr>,<devcode>,G<xx>,<level>;` or

Set channel static colour
`!SCNDMXRGBCOLR,<scn-num>,<addr>,<devcode>,<zone-num>,<preset>;` or
`!SCNDMXRGBCOLR,<scn-num>,<addr>,<devcode>,<zone-num>,#<wrgb>;` or
`!SCNCHANRGBCOLR,<scn-num>,<addr>,<devcode>,<chan-num>,<preset>;` or
`!SCNCHANRGBCOLR,<scn-num>,<addr>,<devcode>,<chan-num>,#<wrgb>;` or

Set channel to colour temperature
`!SCNCHANTWCOLR,<scn-num>,<addr>,<devcode>,<chan-num>,<preset>;` or
`!SCNCHANTWCOLR,<scn-num>,<addr>,<devcode>,<chan-num>,#<kelvin>K;` or
`!SCNDMXTWCOLR,<scn-num>,<addr>,<devcode>,<zone-num>,<preset>;` or
`!SCNDMXTWCOLR,<scn-num>,<addr>,<devcode>,<zone-num>,#<kelvin>K;` or

Set channel to play colour sequence
`!SCNDMXRGBPLAY,<scn-num>,<addr>,<devcode>,<zone-num>,<preset>;` or
`!SCNCHANRGBPLAY,<scn-num>,<addr>,<devcode>,<chan-num>,<preset>;`

followed by 'End of scene items'
`!SCNEND,<scn-num>;`

Only editable scenes will report their items.
Only controllable channels are reported.
e.g.
```
?scnSET,3;
!OK,SCNSET,00003;<CR><LF>
!SCNSET,00003;<CR><LF>
!SCNFADE,00003,00010000;<CR><LF>
!SCNCHAN,00003,002,21,005,255;<CR><LF>
!SCNCHANRGBCOLR,00003,002,21,005,#FF7F00;<CR><LF>
!SCNCHANTWCOLR,00003,002,21,002,#2200K;<CR><LF>
!SCNDALI,00003,003,17,BST,200;<CR><LF>
!SCNDMXRGBCOLR,00003,004,15,001,004;<CR><LF>
!SCNEND,00003;<CR><LF>
```

4.4.2 Channel discovery for scene set operations

- Request scene definition names
  This is used to support scene setting operations. It returns a list of channel elements that are in the scene and the name of the channel element. An element name item is reported in this query for every element item in the corresponding `?SCNSET` query.

  From: v02.02

`?SCNSETNAMES,<scn-num>;`
Long-ack Format: `!OK,SCNSETNAMES,<scn-num>;`
Reply: start of scene definition
`!SCNSETNAMES,<scn-num>;`
followed by zero or more 'Report Scene Item Name'

Channel level name
`!SCNCHANNAME,<scn-num>,<addr>,<devcode>,<chan-num>,<name>;` or
`!SCNDMXNAME,<scn-num>,<addr>,<devcode>,<zone-num>,<name>;` or
`!SCNDALINAME,<scn-num>,<addr>,<devcode>,<dali-num>,<name>;` or
DALI virtual channel name
`!SCNDALI,<scn-num>,<addr>,<devcode>,BST,<name>;` or
`!SCNDALI,<scn-num>,<addr>,<devcode>,G<xx>,<name>;` or

Colour channel name
`!SCNDMXRGBCOLR,<scn-num>,<addr>,<devcode>,<zone-num>,<name>;` or
`!SCNCHANRGBCOLR,<scn-num>,<addr>,<devcode>,<chan-num>,<name>;` or

Colour temperature channel name
`!SCNCHANTWCOLR,<scn-num>,<addr>,<devcode>,<chan-num>,<name>;` or
`!SCNDMXTWCOLR,<scn-num>,<addr>,<devcode>,<zone-num>,<name>;` or

Play colour sequence channel name
`!SCNDMXRGBPLAY,<scn-num>,<addr>,<devcode>,<zone-num>,<name>;` or
`!SCNCHANRGBPLAY,<scn-num>,<addr>,<devcode>,<chan-num>,<name>;`

followed by 'End of scene name items'
`!SCNSETNAMESEND,<scn-num>;`

Only editable scenes will report their items.
Only controllable channels are reported.
e.g.

```
?scnSetNames,3;
!OK,SCNSETNAMES,00003;<CR><LF>
!SCNSETNAMES,00003;<CR><LF>
!SCNCHANNAME,00003,002,21,005,Downlights;<CR><LF>
!SCNCHANRGBCOLRNAME,00003,002,21,005,Downlights;<CR><LF>
!SCNCHANTWCOLRNAME,00003,002,21,002,;<CR><LF>
!SCNDALINAME,00003,003,17,BST,All;<CR><LF>
!SCNDMXRGBCOLRNAME,00003,004,15,001,Meeting Room 1;<CR><LF>
!SCNSETNAMESEND,00003;<CR><LF>
```

5 External Control through GATEWAY

As said in volume 1, to use the Gateway interface via an NPU the NPU must have a configuration file created and loaded. This can either be a minimal configuration that simply defines the hardware modules in the system, or a fuller configuration that also provides controller functions including rules that are triggered when input channel events occur.

If you want to be the system controller, responding to input events and driving output channels, there are 2 ways to achieve this.

A. Minimal configuration with no backup controller
   Simply create an eDIN+ configuration that only defines the hardware modules in the system. Do not create any rules, scenes, areas, etc.

   Then use the Channel API commands and queries to control the system.

   Without any rules the in-built controller will be passive. It will listen to what is going on in the system and report back via the gateway interface, but it will not respond to any input events and drive any channels.

   It is recommended you enable and use GATEWAY events to respond in a timely fashion.

B. Configuration with backup controller
   It is possible for an external system to be the primary controller of an eDIN+ system, and control the system via the GATEWAY interface, but have a backup configuration in the eDIN+ system that kicks in when it detects the primary controller is absent, perhaps due to a lost connection.

   For this create an eDIN+ configuration that defines both hardware modules in the system, and the rules, scenes, areas, etc that give you the backup behaviour you want.

   The only other thing you need to do is to inhibit this internal eDIN+ controller, so that it does not compete/override your primary controller. You do this by repeatedly announcing your presence via the GATEWAY interface and the `$MASTERTICK` command.

   It is recommended you enable and use GATEWAY events to respond in a timely fashion.

5.1 Inhibit In-Built Backup Control

If you decide to have a back up configuration that defines scenes and rules you need to inhibit the in-built controller when you want to control the system. You do this using the `$MASTERTICK` command.

5.1.1 Master Tick command

This command can only be used on connections with *administrator privileges* (see Volume 2 Users for further details).

Use this command to take control of the eDIN+ system and inhibit the internal eDIN+ controller. This command must be sent repeatably to let the internal controller know that the external controller is still alive and in control of the system. When the internal eDIN+

controller no longer detects that this command is being sent, it will take back control of the system.

This command should be sent every second, as this command also provides a heartbeat for the system. It does this by sending the current time across the eDIN+ system so that all parts can synchronise themselves. As such this command contains the current time parameter which is the number seconds since 00:00:00 Jan 1 1970 UTC. Note: this current time should be UTC time, not the time local to your local time zone.

The internal controller waits about 5 seconds of no `$MASTERTICK` messages before it takes back control. This means that the control of the system is stable if the external system stalls for a short time and misses sending the `$MASTERTICK` messages for a couple of seconds or so, but the internal backup system kicks in reasonably quickly if external control is lost, so that e.g. lights can be brought on by a user in a timely fashion.

- Take control and issue time tick
  Command: `$MASTERTICK,<seconds_since_1_jan_1970_utc>;`
  Long-ack Format: `!OK,MASTERTICK,<seconds_since_1_jan_1970_utc>;`
  From: 02.02
  ```
  $masterTick,1646218969;
  !OK,MASTERTICK,1646218969;<CR><LF>
  ```

6 Advanced Channel API

6.1 Advanced Module Discovery

The main discovery queries – described in Volume 1 – provide information about areas and scenes and plates, with limited information about module hardware.

The advanced module discovery queries `?MODULENAMES` returns information about the modules in an eDIN+ system.

It follows the same format as the `?PLATENAMES` query, but can be filtered by device code instead of area.

There is another difference to `?PLATENAMES`, in that `?MODULENAMES` can be used without a configuration loaded in to the eDIN+ system. This makes it a useful companion to the advanced XDALI API that also does not need a configuration loaded. The `?MODULENAMES` query can be used to find available UBCs and DALI universe that can then have DALI messages injected directly on the DALI buses using the XDALI API.

6.1.1 Module Discovery Queries

The `?MODULENAMES` query and response follows the same format as the `?PLATENAMES` message, and the `!MODULENAMES` response returns items to match the items in the `!PLATENAMES` response.

However, most items are not particularly useful. Currently, `device-style` is unused and always returns `0`, modules are not usually assigned to an area so `area-num` is likely to be `0` and `access` fully public (`7`), and modules are not usually given a name.

The main benefit of the `!MODULENAMES` response is that it exists at all, indicating that the module exists, and provides the address and device code that is required to access the module in other GATEWAY messages.

Summary
`addr,devcode` = address and device-code of module used to identify it.
`device-style` = `0` currently unused
`access` = bit-field flags `1` (viewable) + `2` (controllable) + `4` (editable).
`area-num` = area number used to identify the area, as a decimal value.
`module-name` = descriptive name for plate in UTF-8 encoded text

- Request module names
  This returns a list of modules and their properties. You can query for all available modules, or just modules of one particular type or device code.

Query: `?MODULENAME;`
Query: `?MODULENAME,<device-code>;`
Long-ack Format: `!OK,MODULENAME;` or `!OK,MODULENAME,00000;`
From: v02.00
Reply: set of module names
zero or more 'Report Module Name'
```
!MODULENAME,<addr>,<devcode>,<device-style>,<access>,<area-
num>,<module-name>;
```
If the device code parameter is not given in the query then all modules are reported.
e.g.
```
?MODULENAME,17;
!OK,MODULENAME,017;
!MODULENAME,001,017,00,07,00000,;<CR><LF>
!MODULENAME,002,017,00,07,00000,;<CR><LF>
```

6.2 Channel Test and Fixture Identification

It is sometimes useful to visibly flash a lighting channel or fixture. It may be necessary to tie the numbers used by the eDIN+ system to physical lighting circuits or fixtures, to locate a physical lighting circuit or check the circuit is correctly operating.

It is possible to temporarily override a lighting circuit and make it visibly gently flash. It is also possible to override a DALI fixture and cause it to flash or identify itself.

It is only possible to identify one channel or fixture on a module at a one time. Identifying a channel or fixture on a module will automatically turn off all other identification on that module. The command to turn off the identification is sent to the module rather than the specific channel.

Also, the identification override has an in-built timeout that will automatically turn off the override when the timeout is reached. This timeout is set to 5 minutes. Resending the appropriate `$SHOW…` command within this 5 minutes will reset the timeout and continue the identification for a further 5 minutes.

6.2.1 Identify channel or fixture

These commands allow you to temporarily override an output channel. Only channels with the Control access flag set will be affected by these commands and not all channel types can be overridden.

The identify channel commands allow you to temporarily override an output channel to determine the locations of the channel's fixtures. When sent to a channel, the channel will continually ramp from its minimum level to its maximum level and back to its minimum level, until the identify command is cancelled.
Notes:
- Only one channel per module can be identified at any one time. Identifying another channel on the module will automatically cancel the previous identify.
- A channel identify will automatically cancel itself after a built-in 300 seconds (5 mins) in case you forget to cancel it. Resending the command within the 5 mins will reset the time for a further 5 mins.
- A DALI fixture is identified using the advanced DALI addressing (described in Volume 2), starting with an 'F' (for fixture) followed by the short address of the fixture.

- Cancel identify channel on a module
  Command: `$SHOWOFF,<addr>,<devcode>;`
  Long-ack Format: `!OK,SHOWOFF,<addr>,<devcode>;`
  From: v02.00
  e.g.
  ```
  $showOff,08,2;
  !OK,SHOWOFF,008,002;<CR><LF>
  ```

- Identify channel
  Command: `$SHOWCHAN,<addr>,<devcode>,<chan-num>;`
  Command: `$SHOWDALI,<addr>,<devcode>,<dali-num>;`
  Long-ack Format: `!OK,SHOWCHAN,…;` or `!OK,SHOWDALI,…;`
  From: v02.00
  ```
  $showChan,08,2,1;
  !OK,SHOWCHAN,008,002,001;<CR><LF>
  $showDali,05,17,3;
  !OK,SHOWDALI,005,017,003;<CR><LF>
  ```

- Identify DALI fixture
  Command: `$SHOWDALI,<addr>,<devcode>,<dali-fixture>;`
  Long-ack Format: `!OK,SHOWDALI,…;`
  From: v02.00
  e.g.
  ```
  $showDali,2,17,F0;
  !OK,SHOWDALI,002,017,F00;
  ```

6.3 DALI Fixture Status

If configured to do so, a UBC can report problems with individual DALI fixtures. If it detects a problem it reports it with an
`!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 – see below.

6.3.1 Request DALI fixture status

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.

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. If you want information about actual DALI fixtures you have to scan the DALI bus using the `$DALISCAN` command – see the Chapter on Advanced DALI repair.

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>
  ```

7 Advanced DALI Repair

In general, DALI fixtures 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 can continually monitor the DALI fixtures, 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 UBC DALI universe needs to be repaired.

To facilitate the repair of DALI there are a number of tools in the GATEWAY interface to help. These tools can be used for DALI repair or with any other activity as appropriate.

There is an advanced Module Discovery query that allows you to find out what UBC and other modules exist.

There are advanced commands that will physically identify and output or DALI channel or DALI fixture. The command will start the lighting circuit or DALI fixture to flash on and off, or identify itself in some other way. In this way it is possible to tie up channel and fixture numbers used by the eDIN+ system with actual lighting circuits and fixtures.

There are advanced commands and queries to scan a DALI universe for fixtures. This will report all fixtures discovered on the DALI bus and their current status. It is then possible to accept a fixture's settings in to the eDIN+ commissioning data, or repair the settings in a physical fixture back to what they are expected to be by the eDIN+ system.

7.1 DALI Repair API and Events

This section details the commands, queries and events involved with advanced DALI repair operations. **All of the command and queries can only be used on connections with *administrator privileges* (see Volume 2 Users for further details).** All events are in the Advanced events class (see Volume 2 Events for further details).

7.1.1 Advanced DALI Sessions

When the eDIN+ system is performing DALI Repair API commands and queries which are changing the fixture information that the system knows about, it is important to turn off and reset the system's fixture error checking mechanism. This is known as entering a *repair session*.

This is done implicitly on each UBC module when any DALI Repair API command or query is received by the UBC. But it is also possible to *explicitly tell all* UBC modules to enter a *repair session* using the `$DALICAPTURE` command.

When a UBC module enters a repair session it clears and resets all DALI errors that it is reporting. The `$DALICAPTURE` command can be a useful tool in itself to reset the DALI errors in a system.

While a UBC module is in a repair session it no longer checks for fixture errors, so it is important to end the session on all UBC modules back to normal operation so that they can detect and report new fixture problems. This is done using the `$DALIDONE` command.

There is an internal timeout on repair sessions, set to 5 minutes, to automatically end the repair session and return the UBC back to normal operation. This timeout will be automatically extended with every Repair API command and query received by a UBC module, and can be extended explicitly with the `$DALICAPTURE` command.

- Start a repair session
  Command: `$DALICAPTURE;`
  Long-ack Format: `!OK,DALICAPTURE;`
  From: v01.00
  e.g.
  ```
  $daliCapture;
  !OK,DALICAPTURE;<CR><LF>
  ```

- End a repair session. This must be sent as the last command of a repair session, when no more scan or repair commands and queries are going to be sent. This resets the DALI universe status and restarts the runtime checking.
  Command: `$DALIDONE;`
  Long-ack Format: `!OK,DALIDONE;`
  From: v01.00
  e.g.
  ```
  $daliDone;
  !OK,DALIDONE;<CR><LF>
  ```

7.1.2 Advanced DALI Scans

To find out what fixtures actually exist on a UBC module and DALI universe you need to perform a scan of the DALI bus. This will collect information on what fixtures current can be seen on the DALI bus and tie them up with the expected fixture list that it has from an eDIN+ system's DALI Commissioning Data. To perform a scan use the `$DALISCAN` command.

To find out the progress of a scan use the `?DALISCAN` query. Alternatively, you can look for `!DALISCAN` events that are generated at the start and end of a scan.

See next section on DALI Fixture Information about obtaining the results of the scan.

<u>Summary</u>
Status = `0` (idle), `1` (search done), `2` (searching), `3` (programming), `4` (error)

- Perform a DALI scan
  Command: `$DALISCAN,<addr>,<devcode>;`
  Long-ack Format: `!OK,DALISCAN,000,000;`
  From: v01.00
  e.g.
  ```
  $daliScan,1,17;
  !OK,DALISCAN,001,017;<CR><LF>
  ```

- Request DALI scan (progress) status.
  Query: `?DALISCAN,<addr>,<devcode>;`
  Long-ack Format: `!OK,DALISCAN,000,000;`
  From: v01.00
  Reply: 'Report DALI scan status' – see event
  e.g.
  ```
  ?daliScan,1,17;
  !OK,DALISCAN,001,017;<CR><LF>
  !DALISCAN,001,017,1,24;<CR><LF>
  ```

- *Event: 'Report DALI scan status'*
  Class: `EVTADV`
  From: v01.00
  ```
  !DALISCAN,<addr>,<devcode>,<scan-status>,<num-of-fixtures>;
  ```
  e.g.
  ```
  !DALISCAN,001,017,1,24;<CR><LF>
  ```

7.1.3 Advanced DALI Live Fixtures

A scan collects information on the fixtures that can current be seen on a DALI bus. You can obtain the results of the scan using `?DALIFIX` query. (Note the `?DALISCAN` query returns the progress of the scan not the results of the scan.)

Note that the `?DALIFIX` query will return no fixture data until a scan is performed. The fixture data is updated by fixture events from the `$DALISCAN`, `$DALIREPAIR` and `$DALIACCEPT` operations. You can query the data as many times as you like without affecting it.

You can also obtain the same fixture data by monitoring for `!DALIFIX` fixture events directly. These event messages are generated in real time during a `$DALISCAN`, `$DALIREPAIR` and `$DALIACCEPT` operation, so no extra `?DALIFIX` query is necessary.
NOTE: The `!DALIFIX` event messages are generated as it finds or repairs the fixtures. During a scan these will be at irregular intervals. Do not be surprised if there are long gaps between some messages and short gaps between others.

The `!DALIEND` query response indicates there is no further fixture information. The `!DALIEND` event message is generated at the end of a scan and indicates the same thing.

Note that the `!DALIFIX` message includes a `fixture-status` but this is different, and has different codes and values, to the `!DALIERR` health status codes. **This can easily catch you out**. The scan-fixture-status codes are given in the table below.

Note that if a fixture does not have a short address it will be reported with the (unusual) `FXX` DALI identifier. This identifier is also used with the `fixture-status = 9 Unassigned` to indicate a previously known fixture is being removed/deleted.

- DALI Scan Fixture Status Codes

value	Error	Description
0	Ok	No Error with fixture
1	Lamp Failure	Fixture is reporting Lamp Failure
2	Missing	The expected fixture with its correct settings is not present on the DALI bus.
5	New	A fixture is present that does not match any of the expected fixtures in the DALI Commissioning Data
8	Address Clash	There is more than one fixture with the same short address. (Commissioning problem)
9	Unassigned	Fixture does not exist or does not have a valid DALI short address.

<u>Summary</u>
`dali-fixture` = *DALI identifier* for DALI fixture short address 0-63
`long-addr` = DALI fixture 24-bit long or random address
`groups` = DALI fixture 16-bit groups bit-field
`device-type` = DALI fixture 8-bit device type
`fixture-status` = 0 = OK, 2 = missing, 5 = new

- Request DALI Fixture Information
  Query: `?DALIFIX,<addr>,<devcode>;`
  Long-ack Format: `!OK,DALIFIX,<addr>,<devcode>;`
  From: v01.00
  Reply: list of fixtures in the specified DALI universe
  zero or more 'Report of DALI Fixture Information'
  followed by 'End of DALI Fixture Information' – see events
  e.g.
```
?daliFix,1,17;
!OK,DALIFIX,001,017;<CR><LF>
!DALIFIX,001,017,F00,04905615,00064,000,0;<CR><LF>
!DALIFIX,001,017,F01,53195828,00032,000,1;<CR><LF>
!DALIFIX,001,017,F01,14096720,00002,000,5;<CR><LF>
!DALIEND,001,017;<CR><LF>
```

- *Event: 'Report of DALI Fixture Information'*
  Class: `EVTADV`
  From: v01.00
```
!DALIFIX,<addr>,<devcode>,<dali-fixture>,<long-addr>,<groups>,
          <device-type>,<fixture-status>;
```
  e.g.
```
!DALIFIX,001,017,F00,04905615,00064,000,0;<CR><LF>
```

- *Event: 'End of DALI Fixture Information'*
  Class: `EVTADV`
  From: v01.00
```
!DALIEND,<addr>,<devcode>;
```
  e.g.
```
!DALIEND,001,017;<CR><LF>
```

7.1.4 Advanced DALI Repair

This section details commands that can be used that adjust both physical fixtures and DALI Commissioning Data, so that DALI fixture errors are no longer reported.

Repairing is a process of *matching* an *expected* fixture in the DALI Commissioning Data with an *actual physical* fixture in the DALI bus. This is when an existing fixture has been replaced by a new fixture, but the new fixture does not have the correct settings. The repair operation actually *reprograms* the physical fixture (where possible), so that the *physical* fixture's *settings* match those of the *expected* fixture. This includes the *short address* and *groups* value. And where it is not possible to adjust the physical fixture's settings, the expected fixture settings in the DALI Commissioning Data are adjusted to match the physical fixture's settings. This includes the *long address* and *DALI device type*.

Accepting is the process of only modifying the DALI Commissioning Data to match the physical fixture, without changing the physical fixture. You can *accept missing* fixtures to *delete* them, *accept new* fixture to *add* them, or *accept minor changes* to existing fixtures to fix them and stop them generating DALI errors.

See the section on The DALI repair Operation for more details on repairing and accepting.

Both repairing and accepting operations generate events. They generate an event for the repair or accept operation itself. They also generate DALI fixture events to report the updated status of the fixtures that are affected by the operations. The fixture events are also used to update the internal fixture list reported by the `?DALIFIX` query.

Although you can use the fixture events from repair and accept operations, it is may be easier to let the eDIN+ system track the fixture changes and obtain a full updated list of fixtures using a new `?DALIFIX` query when required. If a lot of repair and accept operations have been performed, it may be also beneficial to perform a new `$DALISCAN` to check the operations have been correctly performed on the actual DALI bus.

<ins>Summary</ins>
`missing-fixture` – the id of an expected fixture (usually with a status of `5 Missing`)
`new-fixture` – the id of a physical fixture (usually with a status of `5 New`)

`new-or-missing-fixture`
the id of a new or missing fixture that will added or deleted from the DALI Commissioning Data. If there are both new and missing fixtures at the same short address, the data for the existing fixture will be replaced by the new fixture.

- Repair a DALI fixture.
  Command: `$DALIREPAIR,<addr>,<devcode>,<missing-fixture>,<new-fixture>;`
  Long-ack Format: `!OK,DALIREPAIR,…;`
  From: v01.00
  e.g.
  ```
  $dalirepair,1,17,f0,f01;
  !OK,DALIREPAIR,001,017,F00,F01;<CR><LF>
  ```

- Import a DALI fixture.
  Command: `$DALIACCEPT,<addr>,<devcode>,<new-or-missing-fixture>;`

Long-ack Format: `!OK,DALIACCEPT,…;`
From: v01.00
e.g.
```
$daliaccept,1,17,f1;
!OK,DALIACCEPT,001,017,F01;<CR><LF>
```

- *Event: 'Repair of DALI Fixture'*
  Class: `EVTADV`
  From: v01.00
  ```
  !DALIREPAIR,<addr>,<devcode>,<missing-fixture>,<new-fixture>;
  ```
  e.g.
  ```
  !DALIREPAIR,001,017,F00,F01;<CR><LF>
  ```

- *Event: 'Import of DALI Fixture'*
  Class: `EVTADV`
  From: v01.00
  ```
  !DALIACCEPT,<addr>,<devcode>,<new or missing fixture>;
  ```
  e.g.
  ```
  !DALIACCEPT,001,017,F23;<CR><LF>
  ```

7.2 Fixing DALI Broadcast Channel Errors

Under most setups, DALI Broadcast channel errors heal themselves, and clear the error automatically once the physical problem has been fixed. For example if the channel reports a missing fixture and the faulty fixture is replaced, the reported lamp fault will automatically clear.

If the problem does not go away automatically, then the channel needs to be re-initialised – this is a simple operation performed using the module's own menu system, or via the eDIN+ system's web pages.

As such, there is no GATEWAY commands or queries needed to support DALI Broadcast channel repair.

7.3 The DALI Repair Process

In general, DALI fixtures on UBC modules 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.

Repairing a DALI universe involves a number of steps.

7.3.1 Step 1 (optional) – Prepare eDIN+ system for repair

The first step may be to explicitly *capture* all DALI universes in the eDIN+ system. This tells them to stop checking their fixtures and prepare for performing repair operations. This is done with the `$DALICAPTURE` command.

This step is optional as it prepares *ALL* DALI universes in the system. If you are only repairing one DALI universe, you may not want to affect all other universes.

If you omit this step, each DALI universe is *implicitly captured* when you issue it with repair commands (in the subsequent steps).

A consequence of using `$DALICAPTURE` is that when a DALI Universe is captured and prepares for repair, it clears all DALI errors that it is reporting. So this can be a useful tool in itself to reset the DALI error in a system. Be sure to use the `$DALIDONE` command to release the universes back to normal operation (see End of Session step).

7.3.2 Step 2 – Perform a scan

The first operation is to perform a *scan* on a DALI universe 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 and results of the operation, using `?DALISCAN` and `?DALIFIX` queries. The alternative way is to turn on the advanced events and monitor the events generated by the operation. Both methods are described below.

Using Polling

Start the scan by sending the `$DALISCAN` command. The scan usually takes about to 30 sec to complete, however depending on the state of the DALI universe it could take up to 10 minutes or so.

To find out when the scan is complete 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,1,17;
!OK,DALISCAN,001,017;<CR><LF>
!DALISCAN,001,017,01,024;<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.)

```
?daliFix,1,17;
!OK,DALIFIX,001,017;<CR><LF>
!DALIFIX,001,017,F00,04905615,00064,000,0;
…
!DALIEND,001,017;
```

The details of the `!DALIFIX` responses are detailed below as they are common to both polling and events.

Using Events

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

The scan also generates `!DALIFIX` events for each fixture found (and 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.

e.g.
```
$evtAdv,1;
!OK,EVTADV,1;
...
$daliScan,1,17;
!OK,DALISCAN,001,017;<CR><LF>
!DALISCAN,001,017,02,000;<CR><LF> - searching
…
!DALIFIX,001,017,F00,04905615,00064,000,0; - zero or more fixture events
…
!DALIEND,001,017;
!DALISCAN,001,017,01,024;<CR><LF> - successful scan with 24 fixtures found
```

<u>`!DALIFIXTURE` message</u>
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.

The DALI Fixture status codes are described in more detail below.

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.

7.3.3 Step 3 – Identifying the correct new fixture to repair with

To repair a DALI universe you need to match missing (known/expected) fixtures with new (actual/physical) 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.

It may be that you immediately know how to pair fixtures. For example:

- A single physical fixture has been replaced and there is exactly 1 missing and 1 new fixture in the list.
- A new fixture and missing fixture are reported at the same short address. This could simply be because an existing fixture's settings have changed, particularly the long address.

Other situations are not so straightforward and usually involve matching the physical location of the missing expected fixture with a physical fixture (and its short address) at that location by physically flashing or identifying the physical fixtures.

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` *channel* command.

The physical location of the new actual 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` *fixture* command, using the same `Fxx` identifier in the appropriate `New !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.

7.3.4 Step 4 – 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.

It is perfectly acceptable and often necessary to pair a missing fixture with a new fixture at the same short address, e.g. `$DALIREPAIR,3,17,F0,F0;`

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.

7.3.5 Step 5 – Tidying the commissioning data by accepting actual fixtures

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. Accepting a missing fixture deletes that fixture from the commissioning data, and accepting 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 same 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.

7.3.6 Step 6 – Checking and End of 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.

If you want to check that you have fixed all DALI errors, you should be able to know this by inspecting the statuses in the `?DALIFIX` fixture list – every status should be `0 Ok`. An alternative is to re-enable runtime checking 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. It usually takes up to 2 minutes for all errors to be reported.

In any case it is important that when you have completed your repair session, you re-enable the runtime status monitoring by sending the `$DALIDONE;` command.

7.3.7 Step 7 – Back up the configuration

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

7.3.8 Using the Info web service in the repair process

It is possible to obtain and modify the known/expected fixture settings contained in the DALI Commissioning Data via the INFO web service. This can be useful in certain situations.

Note that the Info service only affects one side to the repair process – the expected fixture data. The Info web service does not affect the actual physical fixtures – you will still need to repair these via the GATEWAY.

It is sometimes useful to clear out the existing commission data to start afresh. This may occur because you are re-loading an old configuration file that contains the wrong commissioning data but the actual fixture settings are correct. One way to repair the commissioning data is to

i) Clear all commissioning data for each affected universes via Info web service
ii) (If not already known) Perform a GATEWAY `$DALISCAN` and `?DALIFIX` to obtain a fixture list of actual fixtures – the list will only contain the actual fixtures, all with `New` status.
iii) Use the GATEWAY to `$DALIACCEPT` all fixtures

To clear out existing commission data for a DALI universe, import a blank list of fixtures for that universe,
e.g.

```
!EDIN DALI COMMISSIONING FILE\n
! the next line deletes all DALI commissioning data on UBC 3,17\n
DALIUNIVERSE,3,17\n
! nothing else - no BALLAST items
```

The Info service can be used to perform small commissioning changes. For example, to change a fixture from one Group (Mode channel) to another you could

i) Use the Info service to export the existing commissioning data for that fixture.
ii) Find and edit the Groups value for the required fixture
iii) Re-import the edited data using the Info service. This will now generate a runtime `Bad Commissioning` error as the commissioning expected data does not match up with the actual fixtures.
iv) Repair the affected actual fixture via the GATEWAY with a `$DALIREPAIR` command, e.g. for fixture with short address 0 use `$DALIREPAIR,3,17,F0,F0;`

To obtain the current DALI Commissioning Data use a GET request to

```
    http:/<ip-address-of-npu>/info&what=dali
```

To change the DALI Commission Data use a POST request to the same URL.

See the Chapter on Info web service for full information about GET and POST request to the service, including passwords and data format.

7.3.9 Repair Process 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,F01,53195828,00032,000,2;
!DALIFIX,001,017,F01,14096720,00032,000,5;
!DALIFIX,001,017,F02,08348761,00002,000,2;
!DALIFIX,001,017,F03,21346988,00008,000,0;
!DALIFIX,001,017,F04,18361283,00000,000,5;
!DALIFIX,001,017,F05,00283700,00000,000,5;
!DALIEND,001,017;
!DALISCAN,001,017,01,005;
```

This shows that there are 4 expected fixtures in the commissioning data, at short address 0,1,2 & 3.

```
!DALIFIX,001,017,F00,04905615,00064,000,0;
!DALIFIX,001,017,F01,53195828,00032,000,2;
!DALIFIX,001,017,F02,08348761,00002,000,2;
!DALIFIX,001,017,F03,21346988,00008,000,0;
```

The fixtures at address `F00` and `F03` are `0 Ok` and do not need repairing, but both expected fixtures `F01` & `F02` are `2 Missing` and need repairing.

This shows that the scan found 5 actual fixtures at short address 0, 1, 3, 4 and 5.

```
!DALIFIX,001,017,F00,04905615,00064,000,0;
!DALIFIX,001,017,F01,14096720,00032,000,5;
!DALIFIX,001,017,F03,21346988,00008,000,0;
!DALIFIX,001,017,F04,18361283,00000,000,5;
!DALIFIX,001,017,F05,00283700,00000,000,5;
```

The fixtures at address `F00` and `F03` are known about and `0 Ok`. The actual fixtures `F01`, `F04` & `F05` are `5 New` (or modified).

Let us take each missing fixture in turn, starting with fixture `F01`.

```
!DALIFIX,001,017,F01,53195828,00032,000,2;
!DALIFIX,001,017,F01,14096720,00032,000,5;
```

We have both a missing and new fixture at short address 1, however the only difference between them is the long address. 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.

To confirm that the new fixture `F01` 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 the missing fixture `F01` with new fixture `F01` 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 `F02`.

```
!DALIFIX,001,017, F02,08348761,00002,000,2;
!DALIFIX,001,017,F04,18361283,00000,000,5;
!DALIFIX,001,017,F05,00283700,00000,000,5;
```

There are now only two new fixtures (`F04` & `F05`) but either of these could be the correct one to repair the missing `F02` fixture. We flash each new 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 `F05` is in the correct physical location and we use this to repair fixture `F02` 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.

8 Advanced Sensor Repair

To be completed

9 The XDALI API

The eDIN+ system normally works hard to hide any DALI bus that exists in the system and does not support the full range of features available on DALI. However, the XDALI API set of commands and queries allows direct access to the DALI buses, providing a way to send any DALI message and perform any DALI operation directly on a DALI bus bypassing an eDIN+ system entirely.

All of the command and queries can only be used on connections with *administrator privileges* (see Volume 2 Users for further details). There are no events with XDALI API.

You can use the helper `?MODULENAMES,17;` query to discover available DALI Universes and their UBC modules.

The XDALI API supports the standard 16-bit Control Gear DALI2 messages. It does not support the extended 24-bit Control Device DALI2 messages.

<u>DALI2 message format</u>
The DALI2 standard defines the Control Gear 16-bit message format. There are 4 different formats:

- The DAP (Direct-Arc-Power) command that sets a fixture's brightness
  The first 8-bits identifies the message as DAP message and hold the address the fixture(s). The second 8-bits hold the brightness level.
- General Opcode messages that send a general command or query to addressed fixtures.
  The first 8-bits identifies hold the address the fixture(s). The second 8-bits hold the opcode that represents a command or query.
- Special Opcode messages that send a special command or query to all fixtures.
  The first 8-bits identifies hold the special opcode. The second 8-bits holds parameter data for the special command or query.
- Application Extended Opcode messages that send a command or query to a particular device type of addressed fixtures.
  These messages are the same format as General Opcode messages but they have to be immediately preceded with a special 'Enable Device Type' message.

There are 2 flags that messages can have:

- Needs Answer – specifies that the opcode is a query and has an 8-bit response
- Send Twice – some commands do not want to be accidentally received by a fixture as e.g. they change a stored setting or reset the fixture. To give extra protection the command message must be sent twice in quick succession (100ms).

A response is generated for a message that 'needs answer'. Some require a 'Yes' / 'No' response, others require an actual value. There can be the following legitimate responses:

- No response: either because the addressed fixture is not present or is a 'No' response.
- Bad response: either a corrupted response because 2 or more fixtures are responding at the same time (and should not be) or a 'Yes' response – it is valid that more than one fixture can respond to Yes/No queries.
- Good response: an 8-bit value has been successfully received. A 'Yes' response has the value 255.

9.1 XDALI API and Connections

The XDALI API commands and queries are used over connections in the same way as other messages. However the connection must have *administrator privileges* (see Volume 2 Users for further details).

For HTTP connections there is an additional optional connection parameter that controls the length of HTTP response timeout when queries are used.

9.1.1 HTTP Query Timeout control

XDALI API queries are sent directly to UBC modules and the responses come directly from UBC modules. This means that any HTTP request that contains an XDALI API query must wait until the UBC send the required response. However, if the UBC does not exist, or for some other reason, and the UBC does not send the required response, the HTTP request could take a long time to complete.

For this reason HTTP requests have a default timeout value of a few seconds. However, this default value may not be appropriate for all your situations, so it is possible for you to specify the timeout you want for each HTTP request using the custom HTTP request header `ModeLighting-Timeout`.

The `ModeLighting-Timeout` custom HTTP request header specifies the required timeout in seconds. For example, add the following line to your HTTP request header block to set the timeout to 20 seconds.

```
ModeLighting-Timeout: 20
```

9.2 XDALI API messages

The XDALI API follows the DALI2 format structure.

- The DAP command has its own `$XDALIDAP` command
- General Opcodes messages use the `$XDALI` command or `?XDALI` query
- Special Opcode messages use the `$XDALISP` command or `?XDALISP` query
- Application Extended Opcode messages use the `$XDALAPP` command or `?XDALIAPP` query

- All Opcode messages that need an answer, use the `?XDALI…` query. Otherwise they are commands and use `$XDALI…`
- Commands that need to be sent twice use the `$XDALI…X2` suffix. By explicitly identifying the command as a 'send twice' twice command, the system can ensure it is sent within the 100ms required by the DALI2 standard.

- All `XDALI…` messages use the usual *DALI identifier* for native DALI addressing: `BST`, `Gxx` and `Fxx`.

- All `XDALI…` queries return an 8-bit `resp-data` and a `resp-status`. The `resp-status` has the following values, following the DALI2 standard responses.

value	resp-status	Description
0	Ok	A 'Yes' response, or the value of the `resp-data` is valid.
1	No Response	A 'No' response or 'No fixture' error
2	Corrupt Response	A 'Yes' response or 'Corrupt Data' error due to more than one fixture replying erroneously or noise on bus.

Each `$XDALI…` command (except `$XDALIDELAY`) generates the equivalent event when the command has been sent on the DALI bus.

A few commands need time to be processed and should not be sent any other message during this period. For example the RANDOMISE command takes up to 300ms to perform and fixtures should not send any message before the 300ms delay has occurred.

- The `$XDALIDELAY` command forces a delay on to the DALI bus by blocking the bus for the required time, keeping it quiet, so that subsequence XDALI messages will be delayed until after this time.

9.2.1 XDALI DAP message

This sends the DAP' set Direct-Arc-Power' to one or more fixtures.

Summary
`dali-id` = `BST`, `Gxx` and `Fxx` where `xx` is group or short address
`dap-level = 0` off, `1-254` DAP level, `255` stop fade

- *Command:* `$XDALIDAP,<addr>,<devcode>,<dali-id>,<dap-level>;`
  Long-ack Format: `!OK,XDALIDAP…;`
  From: v02.00
  e.g.
  ```
  $xdaliDAP,3,17,254;
  !OK,XDALIDAP,003,017,254;<CR><LF>
  ```

- *Event: 'Direct DALI DAP message'*
  Class: `EVTADV`
  From: v02.00
  `!XDALIDAP,<addr>,<devcode>,<dali-id>,<dap-level>;`

9.2.2 XDALI General messages

This sends a general message to one or more fixtures.

Summary
`dali-id` = `BST`, `Gxx` and `Fxx` where `xx` is group or short address
`opcode =  0-200` as defined by the DALI2 standard
`resp-data, resp-status` – see general section above

- *Command:* `$XDALI,<addr>,<devcode>,<dali-id>,<opcode>;`
  Command: `$XDALIX2,<addr>,<devcode>,<dali-id>,<opcode>;`
  Long-ack Format: `!OK,XDALI…` or `!OK,XDALIX2…`
  From: v02.00

e.g.
`$xdali,3,17,bst,5;` recall max level command
`!OK,XDALI,003,017,BST,005;<CR><LF>`
`$xdaliX2,3,17,f23,32;` reset command
`!OK,XDALIX2,003,017,F23,032;<CR><LF>`

- *Query:* `?XDALI,<addr>,<devcode>,<dali-id>,<opcode>;`
  Long-ack Format: `!OK,XDALI,<addr>,<devcode>,<dali-id>,<opcode>;`
  From: v02.00
  Reply: Report of General DALI message
  `!XDALI,<addr>,<devcode>,<dali-id>,<opcode>,<resp-data>,<resp-status>;`
  e.g.
  `?xdali,3,17,bst,146;` query lamp failure
  `!OK,XDALI,003,017,BST,146;<CR><LF>`
  `!XDALI,003,017,BST,146,000,001;<CR><LF>` 'No' answer

- *Event: 'Direct DALI General message'*
  Class: `EVTADV`
  From: v02.00
  `!XDALI,<addr>,<devcode>,<dali-id>,<opcode>;`
  `!XDALIX2,<addr>,<devcode>,<dali-id>,<opcode>;`

9.2.3 XDALISP Special messages

This sends a special message to all fixtures.

<u>Summary</u>
`sp-cmd` = `161-201` value of 'address byte' as defined in the DALI2 standard
`data` = `0-255` value of 'opcode byte' as defined in the DALI2 standard
`resp-data, resp-status` – see general section above

- *Command:* `$XDALISP,<addr>,<devcode>,<opcode>,<data>;`
  Command: `$XDALISPX2,<addr>,<devcode>,<opcode>,<data>;`
  Long-ack Format: `!OK,XDALISP…` or `!OK,XDALISPX2…`
  From: v02.00
  e.g.
  `$xdaliSP,3,17,163,12;` store DTR0(data) command
  `!OK,XDALISP,003,017,163,012;<CR><LF>`
  `$xdaliSpX2,3,17,167,0;` randomise command
  `!OK,XDALISPX2,003,017,167,000;<CR><LF>`

- *Query:* `?XDALI,<addr>,<devcode>,<opcode>,<data>;`
  Long-ack Format: `!OK,XDALI,<addr>,<devcode>,<opcode>,<data>;`
  From: v02.00
  Reply: Report of Special DALI message
  `!XDALISP,<addr>,<devcode>,<opcode>,<data>,<resp-data>,<resp-status>;`
  e.g.
  `?xdaliSp,3,17,169,0;` compare query
  `!OK,XDALISP,003,017,169,000;<CR><LF>`
  `!XDALISP,003,017,169,000,000,002;<CR><LF>` 'Yes' (corrupted) answer

- *Event: 'Direct DALI Special message'*
  Class: `EVTADV`
  From: v02.00
  `!XDALISP,<addr>,<devcode>,<opcode>,<data>;`
  `!XDALISPX2,<addr>,<devcode>,<opcode>,<data>;`

9.2.4 XDALIAPP Application extended messages

This sends an application extended message to one or more fixtures, immediately preceding the specified message with the required 'ENABLE DEVICE TYPE x' special message.

<u>Summary</u>
`dali-id` = `BST`, `Gxx` and `Fxx` where `xx` is group or short address
`opcode` = `224-254` as defined by the DALI2 standard
`device-type` = `0-253` as defined by the DALI2 standard
`resp-data, resp-status` – see general section above

- *Command:* `$XDALIAPP,<addr>,<devcode>,<dali-id>,<opcode>,<device-type>;`
  Command: `$XDALIAPPX2,<addr>,<devcode>,<dali-id>,<opcode>,<device-type>;`
  Long-ack Format: `!OK,XDALIAPP…` or `!OK,XDALIAPPX2…`
  From: v02.00
  e.g.
  `$xdaliApp,3,17,g1,227,1;` start function test – type 1 emergency lighting
  `!OK,XDALIAPP,003,017,G01,227,001;<CR><LF>`
  `$xdaliAppX2,3,17,f8,240,1;` start identification – type 1 emergency lighting
  `!OK,XDALIAPPX2,003,017,F08,240,001;<CR><LF>`

- *Query:* `?XDALIAPP,<addr>,<devcode>,<dali-id>,<opcode>,<device-type>;`
  Long-ack Format: `!OK,XDALIAPP,…;`
  From: v02.00
  Reply: Report of General DALI message
  ```
  !XDALIAPP,<addr>,<devcode>,<dali-id>,<opcode>,<device-type>,
     <resp-data>,<resp-status>;
  ```
  e.g.
  `?xdaliApp,3,17,f18,253,1;` query emergency status - type 1 emergency lighting
  `!OK,XDALIAPP,003,017,F18,253,001;<CR><LF>`
  `!XDALIAPP,003,017,F18,253,1,010,0;<CR><LF>` good response = value of 10

- *Event: 'Direct DALI Application Extended message'*
  Class: `EVTADV`
  From: v02.00
  `!XDALIAPP,<addr>,<devcode>,<dali-id>,<opcode>,<device-type>;`
  `!XDALIAPPX2,<addr>,<devcode>,<dali-id>,<opcode>,<device-type>;`

9.2.5 XDALI Delay

Adds a minimum delay in to the internal DALI message queue, so there is at least the specified gap between messages.

<u>Summary</u>
`delaytime-ms` = `0-65000` delay in milliseconds

- *Command:* `$XDALIDELAY,<addr>,<devcode>,<delaytime-ms>;`
  Long-ack Format: `!OK,XDALIAPP…` or `!OK,XDALIAPPX2…`
  From: v02.00
  e.g.
  `$xdaliSpX2,3,17,167,0;` randomise command
  `$xdaliDelay,3,17,300;` wait 300ms
  `$xdaliSpX2,3,17,165,0;` initialise all command

```
!OK,XDALIX2,003,017,167,000;<CR><LF>
!OK,XDALIDELAY,003,017,00300;<CR><LF>
!OK,XDALIX2,003,017,165,000;<CR><LF>
```