

---

# zencontrol Docs

Nov 20, 2025



# Contents

<b>1</b>	<b>Third Party Interface Documentation</b>	<b>1</b>
	TPI Overview	2
	Supported Devices	2
	Licenses	2
	TPI & TPI Advanced over Stream-oriented Transports (RS232, RS485 and TCP)	3
	Serial Communication Parameters	3
	TPI & TPI Advanced over UDP & TCP	3
	TPI (Classic)	4
	TPI Advanced	5
	Frame Structures	5
	TPI Advanced Header	5
	Basic Request Frame	5
	DALI Colour Request Frame	6
	TPI Dynamic Subframe	7
	DMX Colour Request Frame	8
	TPI Advanced Response Frame	9
	TPI Event Multicast Frame	10
	The Sequence Counter Byte	11
	Calculating Checksums	11
	Error Codes	12
	DALI Addressing	12
	Special Values	13
	Instance Binary States	13
	Instance Types	13
	Instance Status & State Bitmasks	13
	TPI Event Types	14
	TPI Event Modes	14
	DMX Channel Block Types	15
	DMX Channel Personality Types	15
	DMX Channel Behaviour Masks	15
	DALI Status Masks	15
	DALI Control Gear Type Masks	16
	Commands	17
	Basic Commands	17
	Other Commands	18
	Examples	19
	TPI Advanced Examples	19
	QUERY_GROUP_LABEL	19
	QUERY_SCENE_LABEL	20
	QUERY_DALI_DEVICE_LABEL	20
	QUERY_PROFILE_LABEL	21
	QUERY_CURRENT_PROFILE_NUMBER	22
	TRIGGER_SDDP_IDENTIFY	23
	QUERY_TPI_EVENT_EMIT_STATE	23
	DALI_ADD_TPI_EVENT_FILTER	24
	DALI_CLEAR_TPI_EVENT_FILTERS	25
	QUERY_DALI_TPIEVENT_FILTERS	26
	ENABLE_TPI_EVENT_EMIT	27

SET_TPI_EVENT_UNICAST_ADDRESS . . . . .	28
QUERY_TPI_EVENT_UNICAST_ADDRESS . . . . .	29
QUERY_GROUP_NUMBERS . . . . .	30
QUERY_DALI_COLOUR . . . . .	31
QUERY_SCENE_NUMBERS . . . . .	31
QUERY_PROFILE_NUMBERS . . . . .	33
QUERY_OCCUPANCY_INSTANCE_TIMERS . . . . .	34
QUERY_INSTANCES_BY_ADDRESS . . . . .	35
QUERY_OPERATING_MODE_BY_ADDRESS . . . . .	36
DALI_COLOUR . . . . .	37
DMX_COLOUR . . . . .	38
QUERY_GROUP_BY_NUMBER . . . . .	39
QUERY_SCENE_BY_NUMBER . . . . .	40
QUERY_SCENE_NUMBERS_BY_ADDRESS . . . . .	41
QUERY_SCENE_LEVELS_BY_ADDRESS . . . . .	41
QUERY_GROUP_MEMBERSHIP_BY_ADDRESS . . . . .	42
QUERY_DALI_ADDRESSES_WITH_INSTANCES . . . . .	43
QUERY_DMX_DEVICE_NUMBERS . . . . .	44
QUERY_DMX_DEVICE_BY_NUMBER . . . . .	45
QUERY_DMX_LEVEL_BY_CHANNEL . . . . .	46
QUERY_DMX_DEVICE_LABEL_BY_NUMBER . . . . .	47
QUERY_SCENE_NUMBERS_FOR_GROUP . . . . .	48
QUERY_SCENE_LABEL_FOR_GROUP . . . . .	48
QUERY_CONTROLLER_VERSION_NUMBER . . . . .	50
QUERY_CONTROL_GEAR_DALI_ADDRESSES . . . . .	51
DALI_INHIBIT . . . . .	52
DALI_SCENE . . . . .	52
DALI_ARC_LEVEL . . . . .	53
DALI_ON_STEP_UP . . . . .	53
DALI_STEP_DOWN_OFF . . . . .	54
DALI_UP . . . . .	54
DALI_DOWN . . . . .	55
DALI_RECALL_MAX . . . . .	55
DALI_RECALL_MIN . . . . .	56
DALI_OFF . . . . .	56
DALI_QUERY_LEVEL . . . . .	57
DALI_QUERY_CONTROL_GEAR_STATUS . . . . .	58
DALI_QUERY_CG_TYPE . . . . .	59
DALI_QUERY_LAST_SCENE . . . . .	60
DALI_QUERY_LAST_SCENE_IS_CURRENT . . . . .	61
DALI_QUERY_MIN_LEVEL . . . . .	61
DALI_QUERY_MAX_LEVEL . . . . .	62
DALI_QUERY_FADE_RUNNING . . . . .	62
DALI_ENABLE_DAPC_SEQ . . . . .	63
QUERY_DALI_EAN . . . . .	64
QUERY_DALI_SERIAL . . . . .	65
QUERY_VIRTUAL_INSTANCES . . . . .	66
VIRTUAL_INSTANCE . . . . .	67
DALI_CUSTOM_FADE . . . . .	67
DALI_GO_TO_LAST_ACTIVE_LEVEL . . . . .	69
QUERY_DALI_INSTANCE_LABEL . . . . .	69
CHANGE_PROFILE_NUMBER . . . . .	70
QUERY_INSTANCE_GROUPS . . . . .	71
QUERY_DALI_FITTING_NUMBER . . . . .	72
QUERY_DALI_INSTANCE_FITTING_NUMBER . . . . .	73
QUERY_CONTROLLER_LABEL . . . . .	74
QUERY_CONTROLLER_FITTING_NUMBER . . . . .	75
QUERY_IS_DALI_READY . . . . .	75
QUERY_CONTROLLER_STARTUP_COMPLETE . . . . .	77
OVERRIDE_DALI_BUTTON_LED_STATE . . . . .	77

QUERY_LAST_KNOWN_DALI_BUTTON_LED_STATE . . . . .	78
DALI_STOP_FADE . . . . .	79
QUERY_DALI_COLOUR_FEATURES . . . . .	80
QUERY_DALI_COLOUR_TEMP_LIMITS . . . . .	81
SET_SYSTEM_VARIABLE . . . . .	82
QUERY_SYSTEM_VARIABLE . . . . .	83
QUERY_SYSTEM_VARIABLE_NAME . . . . .	84
QUERY_PROFILE_INFORMATION . . . . .	85
QUERY_COLOUR_SCENE_MEMBERSHIP_BY_ADDR . . . . .	86
QUERY_COLOUR_SCENE_0_7_DATA_FOR_ADDR . . . . .	87
QUERY_COLOUR_SCENE_8_11_DATA_FOR_ADDR . . . . .	87
BUTTON_PRESS_EVENT and BUTTON_HOLD_EVENT . . . . .	89
ABSOLUTE_INPUT_EVENT . . . . .	90
LEVEL_CHANGE_EVENT . . . . .	91
GROUP_LEVEL_CHANGE_EVENT . . . . .	91
SCENE_CHANGE_EVENT . . . . .	92
OCCUPANCY_EVENT . . . . .	93
SYSTEM_VARIABLE_CHANGED_EVENT . . . . .	94
COLOUR_CHANGED_EVENT . . . . .	95
PROFILE_CHANGED_EVENT . . . . .	96
GROUP_OCCUPANCY_EVENT . . . . .	97
LEVEL_CHANGE_EVENT_V2 . . . . .	98
Changelog . . . . .	99



# Third Party Interface Documentation

# TPI Overview

The Third Party Interface (TPI) is designed to allow third parties to integrate with a zencontrol controller using a UDP or RS485 protocol.

There are two versions of the TPI to consider.

1. **TPI** - this is the original/classic TPI. It can perform simple DALI commands, queries, scenes and also inhibit sensors from changing targets for a period of time.
2. **TPI Advanced** - this is the newer version which contains a superset of functionality including multicast-events, meta-data queries, DALI and DMX colour commands, and more.

## Supported Devices

### TPI

- Application Controller
- Room Controller

### TPI Advanced

- Application Controller Pro
- Field Controller
- ACx3 Pro

Only controllers with RS485 terminals support TPI over Serial.

## Licenses

Licenses, or “Add-Ons” can be obtained by emailing support. For more detailed instructions see [How do I purchase control system upgrades?](https://support.zencontrol.com) at <https://support.zencontrol.com>.

License	Description
TPI	The original TPI. Compatible with all controllers.
TPI Advanced	The newer super-set of TPI with additional features. Only compatible with Pro-series controllers.
TPI Serial	Enable RS485 serial support to TPI or TPI Advanced.

### Note

Some TPI Advanced features rely on additional addons that aren't related to the TPI itself, Control4 and Virtual Switches (Virtual Instances) for example.

## TPI & TPI Advanced over Stream-oriented Transports (RS232, RS485 and TCP)

Although the TPI is primarily targeted at UDP transport the majority of features are available over RS485 serial too.

The message contents and headers are the same over both transports although implementations will need to deal with stream-oriented semantics that aren't relevant when using UDP. All messages can have their length determined by reading the first few bytes of a message and this allows a developer to identify individual request messages in a stream of data.

TPI (Classic) Messages request frames are 7 bytes long and response frames are 3 bytes long.

TPI Advanced messages either have a fixed-length request frame format or for some sub-frame types a field in the request frame indicates the length of the frame in bytes. All TPI Advanced responses are variable length with a field indicating the frame length in bytes.

### Serial Communication Parameters

The serial parameters are as follows for RS232 and RS485. Note that you must have a TPI Serial license to use TPI or TPI Advanced over serial.

Parameter	Value
Baud	19200
Bits per Byte	8
Parity	None
Stop Bits	1

#### Warning

*TPI Events* aren't (yet) available over a serial connection.

## TPI & TPI Advanced over UDP & TCP

The TPI is primarily designed to be used over UDP. UDP doesn't guarantee delivery (or order of delivery) however it's a simple and lightweight transport that also supports multicast.

Parameter	Value
TPI & TPI Advanced UDP/TCP Port	5108
TPI Events Multicast Address	239.255.90.67
TPI Events Multicast Port	6969

TPI Requests are to be submitted to the IP address of the controller, not to the multicast address. There is currently a maximum of 5 concurrent TCP sessions.

#### Note

Although TPI (classic) is able to give responses to requests it may not always be clear if a response is related to the last request that was sent. TPI Advanced helps fix this issue by adding a `Sequence Number` that can be set for each request and included in the relevant response.

# TPI (Classic)

TPI Classic documentation can be found at:

[https://support.zencontrol.com/hc/en-us/article\\_attachments/360004332095/Application\\_Note-Ethernet\\_UDP\\_v8\\_25-05-20.pdf](https://support.zencontrol.com/hc/en-us/article_attachments/360004332095/Application_Note-Ethernet_UDP_v8_25-05-20.pdf)

The only difference that you should be aware of is that it's now possible to use TPI Classic over RS485 and RS232 if you have the TPI Serial license enabled. See *TPI & TPI Advanced over RS485 Serial*.

# TPI Advanced

## Frame Structures

### Tip

The control byte for all TPI Advanced requests is **0x04**

## TPI Advanced Header

The standard header for all TPI Advanced messages is 3 bytes long.

Control	Sequence Counter	Command
Byte 1	Byte 2	Byte 3

The bytes that come after this are more context dependent and are detailed in specific frame types.

## Basic Request Frame

Most commands in TPI Advanced use the following a 8-byte structure as shown below. All first-generation TPI commands that have been ported to TPI Advanced use this structure, with the main difference simply being the addition of the Sequence Counter and a slightly different byte order.

Control	Sequence Counter	Command	Address	Data Hi	Data Mid	Data Lo	Checksum
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8

Often the Data bytes are simply 0x00 0x00 0x00 except for commands that have more complex or lengthy input (eg. A number of seconds greater than 255) or have to use an `Address` to target that has a length greater than 8 bits (most IDs).

**Control** - This byte tells the controller what message schema to expect.

**Sequence Counter** - See [Sequence Counter Byte](#).

**Command** - this byte specifies the specific command being requested.

**Address** - this byte specifies the address or target of the command. This may be a DALI address, (half) of an instance number, or something else relevant to the context of the command.

**Data bytes** (if required) - data bytes are used to supply extra information relevant to the command being requested. This can sometimes be used to specify an "instance" number which is too large to be specified in a single byte. If no data required just leave these bytes as 0x00.

**Checksum** - A CRC8 checksum is placed at the end of all messages. See [Calculating Checksums](#).

## DALI Colour Request Frame

The DALI Colour frame is used specifically for DALI Colour commands. The message is fixed length but some data bytes may not be used if the colour type does not define usage.

Control	Sequence Counter	Command	Address	Arc Level	Colour Type	Colour Data	Checksum
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Bytes 7 ... 13	Last Byte

**Control** - This byte tells the controller what message schema to expect.

**Sequence Counter** - See [Sequence Counter Byte](#).

**Command** - this byte specifies the specific command being requested. For the DALI Colour command this is 0x0E.

**Address** - this byte specifies the address or target of the command.

**Arc Level** - Optional arc level to go to, during the colour fade. Set to 0xFF to do colour only fade.

### Colour Type

- XY 0x10
- Tc 0x20
- RGBWAF 0x80

**Colour Data** - Depending on the Colour Type specify the values in bytes for each channel of the colour type. Use 0xFF for any unused bytes.

- XY [X Hi byte] [X Lo byte] [Y Hi byte] [Y Lo byte] 0-0xFFFE usable range, set the value to 0xFFFF to leave a coordinate at its current value.
- Tc [Tc Hi byte] [Tc Lo byte] in kelvin (see note on how kelvin is converted to mirek by dali devices)
- RGBWAF [Red] [Green] [Blue] [White] [Amber] [Free colour] [Control] 0-0xFE usable range, set a colour at 0xFF to leave a colour at its current value. The Control byte for RGB maps to the dali RGB control byte. Set this to 0xFF as unused.

**Checksum** - A CRC8 checksum is placed at the end of all messages. See [Calculating Checksums](#).

**A note on Tc colour** - Dali devices use Mirek, instead of kelvin temperature. Mirek = 1 million / kelvin. 2000k is 500 mirek. At the TPI level, we take requests in Tc (K) but each request will be converted to the closest mirek. This creates a potential issue when querying the kelvin temperature of a device and expecting it to be at the value you requested.

For example, 4500k = 222.22 mirek. Dali devices do not take in floating point requests, so the resultant request would be 222 mirek = (1000000 / 222) = 4504k.

Therefore, it may be necessary to request the kelvin temperature that represents the nearest whole number mirek (by doing the same calculation here and sending 4504k).

## TPI Dynamic Subframe

This subframe type has a loose structure with dynamic length. The request data will be command specific.

TPI Advanced Header	Data Length	Data	Checksum
<i>Header Bytes</i> 1-3	Byte 4	Byte 5...n	Last Byte

**Data Length** - The number of bytes in Data section. This will vary based on the specific command.

**Data** - Freeform bytes, as documented for the specific command.

**Checksum** - A CRC8 checksum is placed at the end of all messages. See [Calculating Checksums](#).

## DMX Colour Request Frame

The DMX Colour frame is used specifically for DMX commands. This is a flexible way to create fade tasks and complex patterns across the entire 513 channel range.

TPI Advanced Header	Fade ID	Universe Mask	Start Channel	Stop Channel	Address Divisor
<i>Header Bytes</i> 1-3	Byte 4	Bytes 5-6	Bytes 7-8	Bytes 9-10	Byte 11

Block Mode	Personality Type	Fade Data	Fade Type A	Fade Type B	Data Length	Data	Check-sum
Byte 12	Byte 13	Bytes 14-18	Byte 19	Byte 20	Byte 21	Bytes 21 ... n, n < 16	Last Byte

### Note

The `Start Channel` and `Stop Channel` range represents a “closed bounded” interval. Eg. If you start at channel 1 and end at channel 3, then channel 1 and channel 3 will be affected by the command.

**TPI Advanced Header** - normal *TPI Advanced Header*

**Fade ID** - Give the fade an ID so that it can be referenced for cancellation or overwriting.

**Universe Mask** - The universe mask is 16 bits (two bytes). Use `0xFF 0xFF` for all, though `0x00 0x01` should work for all current single-universe products, including the ACX3 which has three singular universes. It’s possible that future products may support multiple universes per controller.

**Start Channel** - The “start address” for the pattern being created in the DMX buffer. This is two bytes long. Use `0x00 0x01` for channel 1.

**Stop Channel** The “end address” for the pattern being created in the DMX buffer. Channel numbers are two bytes long. Use `0x02 0x01` for channel 513 (the 512th usable channel).

**Address Divisor** - The number to use to “divide” or skip channel numbers when applying the pattern. For example: `0x02` to select every 2nd channel in the range. This interacts with the pattern you express in your Data - for each channel in Data the value will be distributed to a channel indicated by the divisor.

**Block Mode** - How to apply the range described by the `Start Channel`, `Stop Channel`. Useful inverting a range with *DMX\_BLOCK\_DIFFERENCE*, however *DMX\_BLOCK\_INTERSECTION* is a good default.

**Personality Type** - How the fade levels values should be handled. 8bit, 16bit, little endian, big endian, etc. *PERSONALITY\_DIM\_8BIT 0x00* is default.

**Fade Data** - 32 bits of space for basic fade data. Fade times in milliseconds can be up to 24bits long (16777216ms, 16777 seconds, 279 minutes).

- Byte 1: Fade Mode. `0x00` is Fade Time mode. `0x01` is Fade rate mode (not yet implemented) and the value for this will go in bytes 2 - 4.
- Byte 2: Fade Time Hi.
- Byte 3: Fade Time Mid.
- Byte 4: Fade Time Lo.

**Fade Type A** - A fade type. `0x01` Linear Fade is only supported.

**Fade Type B** - A fade type. Use `0x00` for no fade. Combining fades/sequencing not yet supported.

**Data Length** - The number of levels that are to be applied over the range. Currently a max of 16.

**Data** - Up to 16 levels can be used to represent a pattern to be applied over the range. This is useful if your DMX fixtures have channels arranged in an order such as Red, Green, Blue - you can set all three channels at the same time.

**Checksum** - A CRC8 checksum is placed at the end of all messages. See *Calculating Checksums*.

## TPI Advanced Response Frame

For all TPI Advanced requests, a single frame format is used in response. This frame format is variable length. Examples of responses can be found in [TPI Advanced Examples](#).

Response Type	Sequence Counter	Data Length	Data	Checksum
Byte 1	Byte 2	Byte 3.	Bytes 4...n (Optional)	Last Byte

### **Note**

`Data Length` may be 0. In this case there will be no `Data` bytes. Checksum will be in the position of `Data` and the entire frame will be 4 bytes long.

Response Type	Value	Description
OK	0xA0	The request was processed with no problems.
ANSWER	0xA1	The request was processed and an answer is in <code>Data</code> .
NO_ANSWER	0xA2	The request was processed, but there is no answer. Not necessarily an error.
ERROR	0xA3	An error occurred while processing. Check <code>Data</code> if there is any. See <a href="#">Reply Bytes</a>

**Sequence Counter** - See [Sequence Counter Byte](#).

**Data Length** - The length of `Data` to expect.

**Data** - If you have an error the Error Code is likely to be here if `Data Length` > 0. See [Error Codes](#) for more information. When the response is not an error you will typically find values related to the specific request (eg. bytes representing a text label).

**Checksum** - A CRC8 checksum is placed at the end of all messages. See [Calculating Checksums](#).

## TPI Event Multicast Frame

TPI Events are sent by the controller to IGMP Multicast groups so that 3rd party systems on the network can be notified via UDP multicast when events such as button presses occur.

The multicast IPv4 address is **239.255.90.67**. The port is **6969 on UDP**.

Header	Controller MAC Address	Target	Event Type	Data Length	Data	Check-sum
Bytes 1-2 ("ZC")	Bytes 3-8	Bytes 9-10	Byte 11	Byte 12	Bytes 13 ... n (optional)	Last Byte

**Header:** Always [0x5A, 0x43]. ASCII characters ZC.

**Controller MAC:** this is the MAC address of the controller. Some multicast clients do not have APIs that allow the sender MAC to be accessed from context so it must be included in every instance message.

**Target:** this is the context-specific number. Typically something like a DALI Address.

**Event Type:** the type of event. This determines the context of the Target and the Message Data. See [TPI Event Types](#) for event type information.

**Data Length:** indicates the type of event message.

**Data:** variable length data, possibly 0 bytes but up to 48. First byte may be an instance number (if applicable).

**Checksum** - A CRC8 checksum is placed at the end of all messages. See [Calculating Checksums](#).

### Tip

Event multicast must be enabled on controller start-up. It can also be disabled at any time using the `ENABLE_TPI_EVENT_EMIT_CMD`. If the Control4 add-on has been purchased and enabled you can be notified via the Control4 SDDP multicast when a controller starts.

For example event messages, see links in [TPI Event Types](#)

## The Sequence Counter Byte

A sequence counter value allows application developers to easily match requests with responses. This allows detection and handling for out-of-order responses and lost responses.

The sequence counter value does not change any logic within the controller. This feature can be ignored by using a constant value (eg. 0x00) but this is not recommended. Using the sequence number can improve error handling with minimal effort.

Put simply, if a **request** is sent with 0xBE as the Seq. Num byte expect a **response** with 0xBE as the Seq. Num. If sequence numbers between a request and response do not match then:

- Response may just arrive out of order. (Possible due to UDP)
- Or, the response datagram was lost. (Also possible due to UDP)

## Calculating Checksums

This can be calculated by XOR-ing together all preceding bytes of the message (excluding the checksum byte itself). Message integrity can be checked XOR-ing together all bytes of a message (including the checksum byte) and the value should be 0.

The following is an example given in Python.

```
# This calculates the checksum byte for a label query.
>>> checksum = 0x04 ^ 0x00 ^ 0x01 ^ 0x0A ^ 0x00 ^ 0x00 ^ 0x00
>>> print(checksum)
15 # 15 == 0x0F

# There are two methods for checking a checksum that might suit you.

# Method 1: Calculate the checksum, excluding the given checksum in the last byte, then compare to
↳ the last byte.
>>> packet = [0x04, 0x00, 0x01, 0x0A, 0x00, 0x00, 0x00, 0x0F]
>>> given_checksum = packet[-1]
>>> acc = 0x00
>>> for d in packet[:-1]: # All elements except for the last, which is the given checksum.
>>>     acc = d ^ acc
>>>
>>> print(f"Checksum is valid: { acc == given_checksum}")
Checksum is valid: True

# Method 2: XOR all elements including the checksum, if the result is 0 the checksum should be valid
↳ (assuming packet length <64).
>>> from functools import reduce
>>> checksum = reduce(lambda x, y: x ^ y, packet) # xor every element of the packet together.
>>> print(f"Checksum is valid: { 0 == checksum}")
Checksum is valid: True
```

## Error Codes

Error	Value	Description
ERROR_CHECKSUM	0x01	The checksum check failed.
ER-ROR_SHORT_CIRCUIT	0x02	A short on the DALI line was detected. This prevents DALI commands from being sent by the controller
ER-ROR_RECEIVE_ERROR	0x03	A receive error occurred
ER-ROR_UNKNOWN_CMD	0x04	The command in the request is unrecognised
ER-ROR_PAID_FEATURE	0xB0	The command requested relies on a paid feature that hasn't been purchased or is not enabled
ER-ROR_INVALID_ARGS	0xB1	Invalid arguments supplied for the given command in the request
ER-ROR_CMD_REFUSED	0xB2	The command couldn't be processed
ER-ROR_QUEUE_FAILURE	0xB3	The queue or buffer that's required to process the command in the request is full or broken
ER-ROR_RESPONSE_UNAVAIL	0xB4	The command in the request may stream multiple responses back, but this feature is not available for some reason
ER-ROR_OTHER_DALI_ERROR	0xB5	The DALI related request couldn't be processed due to an error
ERROR_MAX_LIMIT	0xB6	There are an insufficient number of the required resource remaining service the request
ER-ROR_UNEXPECTED_RES	0xB7	An unexpected result occurred.
ER-ROR_UNKNOWN_TARGET	0xB8	Device does not exist

## DALI Addressing

The DALI addressing scheme used in TPI Advanced is not "raw" DALI addressing. The table below describes how targeting of DALI commands are performed. For commands that support broadcast (such as TPI lighting commands), the user can use either of the broadcast values (127 was the original TPI designation, 255 is the advanced TPI broadcast value).

Target Type	Range	-
DALI External Control Gear	0 - 63	DALI short addresses.
DALI Groups	64 - 79	Groups are 64 + Group Number. Eg. Group 1 is address 65.
Dali Broadcast	127 or 255	Broadcast (If supported)
DALI External Control Devices	64 - 127	These are logically offset from the ECGs by 64.

### Note

Please note that commands that are not causing DALI commands to be sent out will generally NOT use this addressing scheme - particularly when the commands that operate only on a subset of the data (ie only the groups). For example, if there is a command to query the label for a group, the only possible values are 0-15 and therefore, 0-15 refers to group 0-15 for that command.

### Note

When targeting DALI Instances you must know the address the instance is associated with **and** the instance number. Typically the instance number (or scene number) goes in the *Basic Frame Type* Data Lo field.

## Special Values

### Instance Binary States

All the states in this table are aliased to either LO or HI. These are used when sending commands to an instance.

#### Warning

Note that logical low and logical high are **not** 0 and 1.

State	Value	Description
UNKNOWN	0x00	State is unknown
LO	0x01	Logical <b>Low</b> state
HI	0x02	Logical <b>High</b> state
INSTANCE_SHORT_PRESS	HI	Button short-press
INSTANCE_LONG_PRESS	LO	Button long-press
INSTANCE_ON	HI	On state
INSTANCE_OFF	LO	Off state
INSTANCE_OCCUPIED	HI	A sensor indicates that the area is occupied
INSTANCE_UNOCCUPIED	LO	A sensor indicates that the area is unoccupied

### Instance Types

These are returned from `QUERY_INSTANCES_BY_ADDRESS`. Instances are logical objects that represent input devices in DALI, and are part of later DALI standards such as IEC 62386-301 (push-button instances).

Type	Value	Description
PUSH_BUTTON	0x01	Expect INSTANCE_SHORT_PRESS and INSTANCE_LONG_PRESS events
ABSOLUTE_INPUT	0x02	E.g a slider or dial. Expect ABSOLUTE_INPUT_EVENT events
OCCUPANCY_SENSOR	0x03	Motion sensor. Expect INSTANCE_OCCUPIED events.
LIGHT_SENSOR	0x04	Light sensor. Events not currently forwarded.
GENERAL_PURPOSE_SENSOR	0x06	E.g water flow or power sensor. Events not currently forwarded.

Note: System variables have event support after v2.1.32. You can target a system variable with any of the instance types. This is best utilised with quantity based instances (light sensor, absolute input, and general purpose sensors).

### Instance Status & State Bitmasks

These come from the `QUERY_INSTANCES_BY_ADDRESS` command.

#### State Bits

State information is encoded into the 8 bits of a byte. These values are largely internal to the control system and not to be used by the user.

Not to be confused with *DALI Status Masks*.

Bit	Description
0 (LSB)	Is selected
1	Is_disabled
2	No Targets or has invalid target
3	Is soft disabled
4	Has System Variable Targets
5	Has database operations to do
6	-
7 (MSB)	-

## Status Bits

Status information is encoded into the 8 bits of a byte.

Bit	Description
0 (LSB)	Instance Error
1	Instance Active
2	-
3	-
4	-
5	-
6	-
7 (MSB)	-

## TPI Event Types

TPI Event types are found in *TPI Event multicast frames*. Data relating to the event can be found in the `Data` section of the frame. The target for the event can be found in the `Target` section of the frame.

Event	Value	Description
<i>BUTTON_PRESS_EVENT</i>	0x00	Button has been pressed
<i>BUTTON_HOLD_EVENT</i>	0x01	Button has been pressed and is being held down
<i>ABSOLUTE_INPUT_EVENT</i>	0x02	Absolute input has changed.
<i>LEVEL_CHANGE_EVENT</i>	0x03	Arc Level on an Address target has changed
<i>GROUP_LEVEL_CHANGE_EVENT</i>	0x04	Arc Level on a Group target has changed
<i>SCENE_CHANGE_EVENT</i>	0x05	Scene has been recalled
<i>OCCUPANCY_EVENT</i>	0x06	An occupancy sensor has been triggered, area is occupied.
<i>SYSTEM_VARIABLE_CHANGED_EVENT</i>	0x07	A system variable has changed
<i>COLOUR_CHANGED_EVENT</i>	0x08	A Tc, RGBWAF or XY colour change has occurred
<i>PROFILE_CHANGED_EVENT</i>	0x09	The active profile on the controller has changed.
<i>GROUP_OCCUPANCY_EVENT</i>	0x0A	A sensor targeting a group has detected motion
<i>LEVEL_CHANGE_EVENT_V2</i>	0x0B	Arc Level of address/group plus if/where it is dimming to

### Note

More may be appended to this list in the future.

## Instance Events

Events that are associated with DALI instances (eg. button presses) will have an instance number as the first byte of data.

## TPI Event Modes

TPI Event queries may indicate the active modes in the response. Multiple modes can be active at once. Use bitwise logic and checks to set and inspect the modes.

Mode	Flag Value	Description
DISABLED	0x00	TPI Events won't be transmitted
ENABLED	0x01	TPI Events will be transmitted
DALI_EVENT_FILTERING	0x02	DALI TPI Event filters are active
ENABLE_UNICAST_MODE	0x40	Enable Unicast Mode
DISABLE_MULTICAST_MODE	0x80	Disable Multicast (enabled by default)

## DMX Channel Block Types

Type	Value	Description
DMX_BLOCK_INTERSECTION	0x00	Perform a boolean intersection on the channel range indexes
DMX_BLOCK_DIFFERENCE	0x01	Perform a boolean difference on the channel range indexes

## DMX Channel Personality Types

Type	Value	Description
PERSONALITY_DIM_8BIT	0x00	8 bit dimming
PERSONALITY_DIM_16BIT_BE	0x01	16 bit dimming, with values expressed as Big Endian. Not yet implemented.
PERSONALITY_DIM_16BIT_LE	0x02	16 bit dimming, with values expressed as Little Endian. Not yet implemented.

### Note

8 Bit dimming 0x00 is probably what you want in pretty much all cases. Some DMX fitting manufacturers will do their own smoothing between 8bit channel values.

## DMX Channel Behaviour Masks

These describe whether a channel is expected to be an input or an output. It's possible for a channel to indicate both values (0x03) because these values are bitwise masks.

Type	Value	Description
DMX_BEHAVIOUR_TRIGGER	0x01	A trigger is an input which may be used for DMX -> DALI or other things.
DMX_BEHAVIOUR_OUTPUT	0x02	DMX output channel.

## DALI Status Masks

These are returned from [DALI\\_QUERY\\_CONTROL\\_GEAR\\_STATUS](#) for Control Gear.

Name	Value	Description
DALI_STATUS_CG_FAILURE	0x01	Control Gear Failure
DALI_STATUS_LAMP_FAILURE	0x02	Lamp Failure
DALI_STATUS_LAMP_POWER_ON	0x04	Power On
DALI_STATUS_LIMIT_ERROR	0x08	Limit error (an Arc-level > Max or < Min requested)
DALI_STATUS_FADE_RUNNING	0x10	A fade is running on the light
DALI_STATUS_RESET	0x20	Device has been reset
DALI_STATUS_MISSING_SHORT_ADDRESS	0x40	Device hasn't been assigned a short-address
DALI_STATUS_POWER_FAILURE	0x80	Power failure has occurred

## DALI Control Gear Type Masks

Returns a 32bit number that encompasses all device types that the control device has. The assembly of the number is little endian.

For example, if we get back 0x02, 0x01, 0x00, 0x00, the 32 bit number would be 0x00000102, indicating that the device is an EMERGENCY and COLOUR CONTROL.

These mask values are returned from *DALI\_QUERY\_CG\_TYPE*.

Name	Value	Device Type	Description
DALI_HW_FLUORESCENT	0x01	0	A fluorescent light
DALI_HW_EMERGENCY	0x02	1	An emergency light
DALI_HW_DISCHARGE	0x04	2	-
DALI_HW_HALOGEN	0x08	3	A halogen light
DALI_HW_INCANDESCENT	0x10	4	An incandescent light
DALI_HW_DC	0x20	5	Device uses DC power
DALI_HW_LED	0x40	6	A LED Light
DALI_HW_RELAY	0x80	7	A relay device
DALI_HW_COLOUR_CONTROL	0x100	8	Device has colour control/Type 8 capability
DALI_HW_LOAD_REFERENCING	0x8000	15	-
DALI_HW_THERMAL_GEAR_PROTECTION	0x10000	16	-
DALI_HW_DIMMING_CURVE_SELECTION	0x20000	17	-

### Note

Some values are not yet documented.

## Commands

All Commands are linked to examples.

### Basic Commands

The request frame type varies from command to command, however most use the *Basic frame type*. All TPI Advanced commands reply with the *TPI Advanced Response* frame type.

Each Command has an associated example in *TPI Advanced Examples*.

The following commands all use the *Basic frame type*.

Command	Value	Description
<a href="#">QUERY_GROUP_LABEL</a>	0x01	Query the label for a DALI Group by Group Number
<a href="#">QUERY_SCENE_LABEL</a>	0x02	Query the label for a DALI Scene by Scene Number
<a href="#">QUERY_DALI_DEVICE_LABEL</a>	0x03	Query the label for a DALI ECD or ECG by address
<a href="#">QUERY_PROFILE_LABEL</a>	0x04	Query the label for a controller profile
<a href="#">QUERY_CURRENT_PROFILE_NUMBER</a>	0x05	Query the current profile number
<a href="#">TRIGGER_SDDP_IDENTIFY</a>	0x06	Trigger a Control4 SDDP Identify
<a href="#">QUERY_TPI_EVENT_EMIT_STATE</a>	0x07	Query whether TPI Events are enabled or disabled
<a href="#">ENABLE_TPI_EVENT_EMIT</a>	0x08	Enable or disable TPI Events
<a href="#">QUERY_GROUP_NUMBERS</a>	0x09	Query the DALI Group numbers
<a href="#">QUERY_SCENE_NUMBERS</a>	0x0A	Query the DALI Scene numbers
<a href="#">QUERY_PROFILE_NUMBERS</a>	0x0B	Query all available Profile numbers
<a href="#">QUERY_OCCUPANCY_INSTANCE_TIMERS</a>	0x0C	Query an occupancy instance for its timer values
<a href="#">QUERY_INSTANCES_BY_ADDRESS</a>	0x0D	Query information of instances associated with an address
<a href="#">QUERY_GROUP_BY_NUMBER</a>	0x12	Query DALI Group information by Group Number
<a href="#">QUERY_SCENE_BY_NUMBER</a>	0x13	Query DALI Scene information by Scene Number
<a href="#">QUERY_SCENE_NUMBERS_BY_ADDRESS</a>	0x14	Query for DALI Scenes an address has levels for
<a href="#">QUERY_GROUP_MEMBERSHIP_BY_ADDRESS</a>	0x15	Query DALI Group membership by address
<a href="#">QUERY_DALI_ADDRESSES_WITH_INSTANCES</a>	0x16	Query DALI addresses that have associated instances
<a href="#">QUERY_DMX_DEVICE_NUMBERS</a>	0x17	Query DMX Device information
<a href="#">QUERY_DMX_DEVICE_BY_NUMBER</a>	0x18	Query for DMX Device information by channel number
<a href="#">QUERY_DMX_LEVEL_BY_CHANNEL</a>	0x19	Query DMX Channel value by Channel number
<a href="#">QUERY_SCENE_NUMBERS_FOR_GROUP</a>	0x1A	Query Scene Numbers attributed to a group
<a href="#">QUERY_SCENE_LABEL_FOR_GROUP</a>	0x1B	Query Scene Labels attributed to a group scene
<a href="#">QUERY_CONTROLLER_VERSION_NUMBER</a>	0x1C	Query Controller Version Number
<a href="#">QUERY_CONTROL_GEAR_DALI_ADDRESSES</a>	0x1D	Query Control Gear present in database
<a href="#">QUERY_SCENE_LEVELS_BY_ADDRESS</a>	0x1E	Query Scene level values for a given address
<a href="#">QUERY_DMX_DEVICE_LABEL_BY_NUMBER</a>	0x20	Query DMX Device for its label
<a href="#">QUERY_INSTANCE_GROUPS</a>	0x21	Query group targets related to an instance
<a href="#">QUERY_DALI_FITTING_NUMBER</a>	0x22	Query the fitting number for control gear/devices
<a href="#">QUERY_DALI_INSTANCE_FITTING_NUMBER</a>	0x23	Query the fitting number for an instance
<a href="#">QUERY_CONTROLLER_LABEL</a>	0x24	Query the label of the controller
<a href="#">QUERY_CONTROLLER_FITTING_NUMBER</a>	0x25	Query the fitting number of the controller itself
<a href="#">QUERY_IS_DALI_READY</a>	0x26	Query whether DALI is ready (or has a fault)
<a href="#">QUERY_CONTROLLER_STARTUP_COMPLETE</a>	0x27	Query if the controller startup sequence has completed
<a href="#">QUERY_OPERATING_MODE_BY_ADDRESS</a>	0x28	Query operating mode for the device at the given address
<a href="#">OVERRIDE_DALI_BUTTON_LED_STATE</a>	0x29	Override the button LED state on a DALI button
<a href="#">QUERY_LAST_KNOWN_DALI_BUTTON_LED_STATE</a>	0x30	Query the last known button LED state on a DALI button
<a href="#">DALI_ADD_TPI_EVENT_FILTER</a>	0x31	Request that filters be added for DALI TPI Events
<a href="#">QUERY_DALI_TPI_EVENT_FILTERS</a>	0x32	Query DALI TPI Event filters on a address
<a href="#">DALI_CLEAR_TPI_EVENT_FILTERS</a>	0x33	Request that DALI TPI Event filters be cleared
<a href="#">QUERY_DALI_COLOUR</a>	0x34	Query the Colour (RGBWAF or TC) on a DALI target
<a href="#">QUERY_DALI_COLOUR_FEATURES</a>	0x35	Query the DALI colour features/capabilities of gear
<a href="#">SET_SYSTEM_VARIABLE</a>	0x36	Set a system variable value
<a href="#">QUERY_SYSTEM_VARIABLE</a>	0x37	Query system variable
<a href="#">QUERY_DALI_COLOUR_TEMP_LIMITS</a>	0x38	Query DALI Colour Temp max/min and step in Kelvin

continues on next page

Table 1 – continued from previous page

Command	Value	Description
<i>SET_TPI_EVENT_UNICAST_ADDRESS</i>	0x40	Set a TPI Events unicast address and port
<i>QUERY_TPI_EVENT_UNICAST_ADDRESS</i>	0x41	Query TPI Events State, unicast address and port
<i>QUERY_SYSTEM_VARIABLE_NAME</i>	0x42	Query the name of a system variable (Pro controllers)
<i>QUERY_PROFILE_INFORMATION</i>	0x43	Query profile numbers, behaviours etc
<i>QUERY_COLOUR_SCENE_MEMBERSHIP_BY_ADDR</i>	0x44	Query scenes a device has colour data for (Pro controllers)
<i>QUERY_COLOUR_SCENE_0_7_DATA_FOR_ADDR</i>	0x45	Query colour scene data for 0-7 (Pro controllers)
<i>QUERY_COLOUR_SCENE_8_11_DATA_FOR_ADDR</i>	0x46	Query colour scene data for 8-11 (Pro controllers)
<i>DALI_INHIBIT</i>	0xA0	Inhibit sensors from affecting a DALI target for time
<i>DALI_SCENE</i>	0xA1	Call a DALI Scene on a address
<i>DALI_ARC_LEVEL</i>	0xA2	Set an Arc-Level on a address
<i>DALI_ON_STEP_UP</i>	0xA3	On-if-Off and Step Up on a address
<i>DALI_STEP_DOWN_OFF</i>	0xA4	Step Down and off-at-min on a address
<i>DALI_UP</i>	0xA5	Step Up on a address
<i>DALI_DOWN</i>	0xA6	Step Down on a address
<i>DALI_RECALL_MAX</i>	0xA7	Recall the max level on a address
<i>DALI_RECALL_MIN</i>	0xA8	Recall the min level on a address
<i>DALI_OFF</i>	0xA9	Set a address to Off
<i>DALI_QUERY_LEVEL</i>	0xAA	Query the the level on a address
<i>DALI_QUERY_CONTROL_GEAR_STATUS</i>	0xAB	Query the status data on a address, group or broadcast
<i>DALI_QUERY_CG_TYPE</i>	0xAC	Query Control Gear type data on a address
<i>DALI_QUERY_LAST_SCENE</i>	0xAD	Query Last heard DALI Scene
<i>DALI_QUERY_LAST_SCENE_IS_CURRENT</i>	0xAE	Query if target has changed since last heard scene
<i>DALI_QUERY_MIN_LEVEL</i>	0xAF	Query the min level for a DALI device
<i>DALI_QUERY_MAX_LEVEL</i>	0xB0	Query the max level for a DALI device
<i>DALI_QUERY_FADE_RUNNING</i>	0xB1	Query whether a fade is running on a address
<i>DALI_ENABLE_DAPC_SEQ</i>	0xB2	Begin a DALI DAPC sequence
<i>VIRTUAL_INSTANCE</i>	0xB3	Perform an action on a Virtual Instance
<i>DALI_CUSTOM_FADE</i>	0xB4	Call a DALI Arc Level with a custom fade-length
<i>DALI_GO_TO_LAST_ACTIVE_LEVEL</i>	0xB5	Command DALI addresses to go to last active level
<i>QUERY_VIRTUAL_INSTANCES</i>	0xB6	Query for virtual instances and their types
<i>QUERY_DALI_INSTANCE_LABEL</i>	0xB7	Query DALI Instance for its label
<i>QUERY_DALI_EAN</i>	0xB8	Query the DALI European Article Number at an address
<i>QUERY_DALI_SERIAL</i>	0xB9	Query the Serial Number at a address
<i>CHANGE_PROFILE_NUMBER</i>	0xC0	Request a Profile Change on the controller
<i>DALI_STOP_FADE</i>	0xC1	Request a running DALI fade be stopped.

## Other Commands

Command	Frame Type	Value	Description
<i>DALI_COLOUR</i>	<i>DALI Colour</i>	0x0E	Set the DALI level and colour of a DALI colour light
<i>DMX_COLOUR</i>	<i>DMX Colour</i>	0x10	Send values to a set of DMX channels and configure fading

# Examples

## TPI Advanced Examples

### QUERY\_GROUP\_LABEL

**Frame Type:** *Basic*

Get the label for a DALI Group. Group is expressed as 0-15, not the addressing scheme presented earlier in the document. Response data can be up to 64 bytes. Group labels are limited to this size in the cloud. If there is no label, response will REPLY\_ANSWER with 0 for data length.

#### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x01	Command
3	0x0A	Group 0-15 (0x0A=G10)
4	0x00	-
5	0x00	-
6	0x00	-
7	0x0F	Checksum

#### Response with Label

Response data contains Foo as the label for Address 10.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3 - 5	0x466F6F	Foo
6	0xE4	Checksum

Response data for no label would be as follows.

Byte Index	Byte Value	Description
0	0xA2	Response Type (REPLY_NO_ANSWER)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA2	Checksum

## QUERY\_SCENE\_LABEL

**Frame Type:** *Basic*

Get the label for DALI Scene 10. As this has no context of which group is associated, this shouldn't be used and is only left in for legacy purposes. Use [QUERY\\_SCENE\\_LABEL\\_FOR\\_GROUP](#). Max 64 characters. Also note that the controller supports scenes 0-12 for user usage.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x02	Command
3	0x0A	Scene Number 0-12 (0x0A - Scene 10)
4	0x00	-
5	0x00	-
6	0x00	-
7	0x0C	Checksum

### Response with Label

Response data contains Foo as the label for Scene 10.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3 - 5	0x466F6F	Foo
6	0xE4	Checksum

## QUERY\_DALI\_DEVICE\_LABEL

**Frame Type:** *Basic*

Get the label for a DALI device (control gear and control device). If the device has no label, response will be type REPLY\_ERROR. Max 64 characters.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x03	Command
3	0x0A	Address 0-63 CG, 64-127 for CD
4	0x00	-
5	0x00	-
6	0x00	-
7	0x0D	Checksum

### Response with Label

Response data contains Foo as the label for device at Address 10.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3 - 5	0x466F6F	Foo
6	0xE4	Checksum

## QUERY\_PROFILE\_LABEL

Frame Type: *Basic*

Get the label for the Profile given a Profile number.

### Note

Profile Numbers are 2 bytes long.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x04	Command
3	0x00	-
4	0x00	-
5	0x00	Data Mid (Upper byte if needed)
6	0x01	Data Lo (Profile ID: 0x01)
7	0x01	Checksum

### Response with Label

Response data contains Foo as the label for Profile 1.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3 - 5	0x466F6F	Foo
6	0xE4	Checksum

## QUERY\_CURRENT\_PROFILE\_NUMBER

Frame Type: *Basic*

Get the current/active Profile number. Useful for providing to the *Profile Label Command*.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x05	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	-
7	0x01	Checksum

### Response with Profile number

Response data contains 0x0001 as the number for the current active profile (Profile 1 for example).

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x02	Data Length
3	0x00	Data profile ID Hi Byte
4	0x01	Data profile ID Lo Byte
5	0xA2	Checksum

#### Note

Profile Ids are 2 bytes long.

## TRIGGER\_SDDP\_IDENTIFY

Frame Type: *Basic*

Trigger a Control4 SDDP Identify command. This causes the controller to be displayed/identified clearly within Control4. If feature not paid for, will respond with ERROR\_PAID\_FEATURE.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x05	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	-
7	0x01	Checksum

### Response

Just OK. No extra data.

Byte Index	Byte Value	Description
0	0xA0	Response Type (OK)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

## QUERY\_TPI\_EVENT\_EMIT\_STATE

Frame Type: *Basic*

Get the current TPI Event multicast emitter state.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x07	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	-
7	0x03	Checksum

### Response with boolean value

Responses return current state in the data. 0x01 indicates that TPI Events are enabled for transmit or 0x00 for disabled. Values > 1 indicate that event filtering is active. See *TPI Event Modes* for the specific modes that may be active.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x01	Data Length
3	0x01	State (Enabled in this case)
4	0xA1	Checksum

## DALI\_ADD\_TPI\_EVENT\_FILTER

**Frame Type:** *Basic*

Add DALI event filters to stop specific events from being broadcast as TPI Events. Filters are listed in *TPI Event Types*. You must specify a bitmask of events. To filter all events you can use a mask of 0xFFFF. To filter a DALI ECD you must specify the instance number to filter on or use 0xFF for ECGs. To filter all ECG (groups / address) for an event you can use 0xFF for the Address.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x31	Command
3	0x00	Address 0
4	0xFF	Instance Number
5	0x01	Event Mask Hi
6	0x08	Event Mask Lo
7	0xC3	Checksum

Event bitmask is formed by left bit-shifting the event type number (from *TPI Event Types*) and if you want to add multiple events to the mask doing a bitwise OR on the mask. To silence LEVEL\_CHANGE\_EVENT and COLOUR\_CHANGED events the mask can be calculated like this:

```
level_change_event = 3
colour_changed_event = 8
event_mask = (1 << colour_changed_event) | (1 << level_change_event)
```

event\_mask is 264 which looks like 0b100001000 in binary. Notice the events marked positionally by 1's. 264 is too large to fit in a single byte so it must be split across 2 bytes. Upper byte will be 0x01 and lower byte will be 0x08.

### Response

A successful addition of a filter.

Byte Index	Byte Value	Description
0	0xA0	Response Type (REPLY_OK)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

## DALI\_CLEAR\_TPI\_EVENT\_FILTERS

**Frame Type:** *Basic*

Clear DALI event filters. Events are listed in *TPI Event Types*. You must specify a bitmask of events. This examples clears LEVEL\_CHANGE\_EVENT and COLOUR\_CHANGED events from DALI address 0. To clear all events use 0xFFFF and all events for this address can be emitted as TPI events again.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x33	Command
3	0x00	Address
4	0xFF	Instance number
5	0x01	Event Mask Hi
6	0x08	Event Mask Lo
7	0xC1	Checksum

### Note

For calculating the event mask see the note in the *DALI\_ADD\_TPI\_EVENT\_FILTER* example above.

### Response

A successful clearing of a filter. Note that if no such filter exists, you will receive a REPLY\_NO\_ANSWER.

Byte Index	Byte Value	Description
0	0xA0	Response Type (REPLY_OK)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

## QUERY\_DALI\_TPI\_EVENT\_FILTERS

**Frame Type:** *Basic*

Query active DALI event filters. Also returns the TPI event modes active in the first byte. ECG filters must specify an *instance number* of 0xFF, and ECDs must have their instance number specified (unless querying all events for the ECD). If address 0xFF and instance number 0xFF is specified, will return ALL active tpi events on all addresses. As the data payload can only be up to 64 bytes and there are up to 64 event filters, it may be necessary to query several times. The parameter "Result to start at" allows paging of results. For example, if you have all 64 event filters active, you will receive results 0-14 in the first response, you then specific to start at 15 and receive 15-29. To complete the set, you would request 30, 45, 60 as starting numbers or until you receive (NO\_ANSWER) for no more active filters.

Events are listed in *TPI Event Types*.

### Request

Query for all event filters associated with address 0

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x32	Command
3	0x00	Address
4	0x00	Result to start at
5	0x00	-
6	0xFF	Instance Number (all instances)
7	0xC9	Checksum

### Response

An 2 result event mask for the given address

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x09	Data Length
3	0x08	<i>TPI Event Modes Active</i>
4	0x02	Result 0 Address
5	0x00	Result 0 Instance Number
6	0x03	Result 0 Event mask upper byte
7	0x04	Result 0 Event mask lower byte
8	0x02	Result 1 Address
9	0x01	Result 1 Instance Number
10	0x05	Result 1 Event mask upper byte
11	0x06	Result 1 Event mask lower byte
12	0xA3	Checksum

Event filter enabled status

This 2-byte response is a 16bit event mask split across two bytes. The 16-bit value is 264.

To check if an event if flagged in an event mask you can use a bitwise AND against an event mask containing the events you want to check for. If the result is greater than 0 then the event must be present in the query result.

```
event_mask = (upper_byte << 8) | lower_byte
colour_change_event = 8
events_to_check_for = (1 << colour_change_event)

if ((events_to_check_for & event_mask) > 0):
    print("Colour change events are being filtered out.")
```

## ENABLE\_TPI\_EVENT\_EMIT

**Frame Type:** *Basic*

Enable or disable TPI Advanced UDP event messages. By default these event messages are sent using Multicast, however Unicast can be configured and both modes can be used at the same time if necessary.

See [SET\\_TPI\\_EVENT\\_UNICAST\\_ADDRESS](#) for more on how to enable Unicast mode.

### Request

Use 0x01 to enable TPI Events and 0x00 in a Basic frame `Address` position to turn TPI Events on/off. By default, TPI Events will be in Multicast Mode, but not enabled. When a controller boots you must re-assert whether events should be enabled as modes and filters aren't persistent.

For more information on the Event Mode values see [TPI Event Modes](#).

### Tip

Consider using [QUERY\\_TPI\\_EVENT\\_EMIT\\_STATE](#) as a method to periodically "ping" the controller and if necessary re-assert the state depending on the response.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x08	Command
3	0x01	Enable (or 0x00 for disable). See <a href="#">TPI Event Modes</a> for more values.
4	0x00	-
5	0x00	-
6	0x00	-
7	0x0D	Checksum

### Response with boolean value

Responses return current state in the data. 0x01 indicates that TPI Events are enabled for transmit or 0x00 for disabled.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x01	Data Length
3	0x01	State (Enabled in this case). See <a href="#">TPI Event Modes</a> .
4	0xA1	Checksum

## SET\_TPI\_EVENT\_UNICAST\_ADDRESS

**Frame Type:** *Dynamic*

TPI Events Unicast Mode is useful if:

- Your control system can't support Multicast.
- You do not want to use Multicast on your network or have particular security concerns.
- You want to capture and process events entirely in your own system.

Typically you should configure Unicast using this command before you enable Unicast using `ENABLE_TPI_EVENT_EMIT` with an `Enable` value of `0x41` which is `BITWISE_OR(0x40 | 0x01)` which are the byte values for Unicast mode and TPI Events general enable.

### Request

This request configures TPI Events for Unicast to be sent to `192.168.10.10` on port `8811`.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x40	Command
3	0x06	Data Length
4	0x22	Port Upper Byte
5	0x6B	Port Lower Byte
6	0xC0	IP4 Byte 0 (192.x.x.x)
7	0xA8	IP4 Byte 1 (x.168.x.x)
8	0x0A	IP4 Byte 2 (x.x.10.x)
9	0x0A	IP4 Byte 3 (x.x.x.10)
10	0x63	Checksum

**Response** A simple "OK" response.

Byte Index	Byte Value	Description
0	0xA0	Response Type ( <code>REPLY_OK</code> )
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

## QUERY\_TPI\_EVENT\_UNICAST\_ADDRESS

**Frame Type:** *Basic*

Returns the TPI Event emit state, Unicast Port and Unicast Address.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x41	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	-
7	0x45	Checksum

**Response** An answer that shows TPI Unicast mode enabled (and Multicast not disabled), Port 8811 and 192.168.10.10 Address configured.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x07	Data Length
3	0x41	TPI Event State flags (Unicast enabled 0x40, TPI Events enabled 0x01)
4	0x22	Port Upper Byte
5	0x6B	Port Lower Byte
6	0xC0	IP4 Byte 0 (192.x.x.x)
7	0xA8	IP4 Byte 1 (x.168.x.x)
8	0x0A	IP4 Byte 2 (x.x.10.x)
9	0x0A	IP4 Byte 3 (x.x.x.10)
10	0xCC	Checksum

## QUERY\_GROUP\_NUMBERS

**Frame Type:** *Basic*

Query a list of DALI Group Numbers present on the controller. Originally, this was a list of any group that a control gear present in the database is a member of. This now also contains groups that are set up in the groups section on the cloud.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x09	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	-
7	0x0D	Checksum

### Response with Group Numbers

This response contains two Group Number values 0x07 and 0x0F. Note that this a variable sized response dependent on how many groups are mentioned on the bus.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x02	Data Length (number of groups in this case)
3	0x07	First Group
4	0x0F	Second Group
5	0xAB	Checksum

## QUERY\_DALI\_COLOUR

Frame Type: *Basic*

Query colour information from a DALI address. This reports back the colour type (TC, RGBWAF or possibly others in the future) and the bytes that represent the values for that colour type.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x34	Command
3	0x00	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0x30	Checksum

### Response with Colour Data

This response contains RGBWAF data - just red. See the *Colour Type* section in the DALI Colour Frame for a list of colour modes.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x07	Data Length (varies based on Colour Mode)
3	0x80	Colour Mode <i>Colour Type</i> section in the DALI Colour Frame for a list of colour modes.
4	0xFF	R - Red Byte
5	0x00	G - Green Byte
6	0x00	B - Blue Byte
7	0x00	W - White Byte
8	0x00	A - Amber Byte
9	0x00	F - Freecolour Byte
10	0xD9	Checksum

## QUERY\_SCENE\_NUMBERS

Frame Type: *Basic*

Query a list of DALI Scene Numbers. As this has no context of which group is associated, this shouldn't be used and is only left in for legacy purposes. Use *QUERY\_SCENE\_NUMBERS\_FOR\_GROUP*

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x0A	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	-
7	0x0E	Checksum

### Response with Scene IDs

This response contains two Scene Number values 0x07 and 0x0F. Scene Numbers are one byte long.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x02	Data Length (number of scenes in this case)
4	0x07	First Scene
6	0x0F	Second Scene
7	0xAB	Checksum

## QUERY\_PROFILE\_NUMBERS

**Frame Type:** *Basic*

Query a list of Profile Numbers. Superseded by *QUERY\_PROFILE\_INFORMATION*, which has additional information.

Useful for providing to the *Profile Label Command*. Note: Profiles MUST be assigned to each controller on the cloud (Assignment & Logic -> Control Assignment Section).

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x0B	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	-
7	0x0F	Checksum

### Response with Profile Numbers

This response contains two Profile Numbers values 0x07 and 0x0F. Profile Numbers are two bytes long.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x04	Data Length
3	0x00	First Profile Hi Byte
4	0x07	First Profile Lo Byte
5	0x00	Second Profile Hi Byte
6	0x0F	Second Profile Lo Byte
7	0xA2	Checksum

## QUERY\_OCCUPANCY\_INSTANCE\_TIMERS

**Frame Type:** *Basic*

Query Occupancy Instance timers.

### Request

Query an Occupancy Instance for timer values. Requires a Control Device DALI Address (64-127) and the instance number (0-31) of the occupancy sensor instance. Last detected value is the value in seconds since the last event message depicting OCCUPIED status. Counts to 0xFFFF (65535 seconds). It should be noted that the deadtime, hold and report times are set by the controller. Report time is set based on the quantity of occupancy sensors on the line (more occupancy sensors, longer report times).

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x0C	Command
3	0x40	Address (0x40 = CDA0)
4	0x00	-
5	0x00	-
6	0x01	Instance number for occupancy sensor
7	0x49	Checksum

### Response with Occupancy Instance Timers

This response contains Timer data for the occupancy instance with an Number of 0x01.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x05	Data Length
3	0x05	Deadtime (5 seconds)
4	0x3C	Hold (60 seconds)
5	0x78	Report (120 seconds)
6	0x00	Last Detect Hi byte
7	0xFE	Last Detect Lo Byte (254 seconds ago.)
8	0xE5	Checksum

## QUERY\_INSTANCES\_BY\_ADDRESS

**Frame Type:** *Basic*

Query a DALI address to see if it has associated Instances. Returns Instance metadata.

See also [Query DALI Addresses with Instances command](#) which may be a helpful command for finding targets for this command.

### Request

Query DALI Address 65 to see which Instances are associated with this address.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x0D	Command
3	0x41	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0x48	Checksum

### Response with Instance metadata

Responses return 0 or more four byte Instance descriptions.

This example shows a single instance returned. On a reply with multiple instances associated with an address bytes 3 - 6 (4 bytes) would be repeated but with the appropriate values for each instance on the address, and would have a Data Length of 8.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x04	Data Length (divide by 4 to get number of Instances)
3	0x01	Instance Number
4	0x01	Instance Type. (See <a href="#">Instance Types</a> )
5	0x00	Status Bits. (See <a href="#">Instance Status &amp; State Bitmasks</a> )
6	0x00	State Bits. (See <a href="#">Instance Status &amp; State Bitmasks</a> )
7	0xA5	Checksum

## QUERY\_OPERATING\_MODE\_BY\_ADDRESS

**Frame Type:** *Basic*

Query the DALI operating mode given an address. Operating modes are manufacturer dependant in functionality. If the device does not exist in the database, ERROR\_UNKNOWN\_TARGET will be returned.

### Request

Query DALI Address 126 to see what the operating mode is.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x28	Command
3	0x7E	Address (126, control device A62)
4	0x00	-
5	0x00	-
6	0x00	-
7	0x52	Checksum

### Response with Operating Mode

This is an example response with the default operating mode (0). If Data had a value of 0x64 then the operating mode would be 0x64.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x01	Data Length
3	0x00	Data: Operating Mode 0
4	0xA0	Checksum

## DALI\_COLOUR

**Frame Type:** *DALI Colour*

Send a DALI Colour and Arc command. This command tells a DALI Colour device (eg. DALI Device Type 8 in IEC 62386) to go show a colour.

This is the only TPI Advanced command that does not require a Pro-series controller.

### Request

Set a RGBWAF colour (Red) on DALI address 1.

Each channel of the colour space gets a byte after `Colour Type`. If the XY colour space is used, then the length of the entire message would be 11 bytes long.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x0E	Command
3	0x01	Address
4	0xFE	Arc Level
5	0x00	Colour Type
6	0xFF	Colour Red
7	0x00	Colour Green
8	0x00	Colour Blue
9	0x00	Colour White
10	0x00	Colour Amber
11	0x00	Colour Freecolour
12	0x0A	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA0	Response Type (OK)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

If there is an error response type and `Data` of 0x01 then this is likely because the system was busy, so try again and it may work the next time.

## DMX\_COLOUR

**Frame Type:** *DMX Colour*

Send a DMX Colour and Fade command. This command sets up a fade task which transitions from the current values to your specified values.

See *DMX Colour Request Frame* for more information.

### Request

Send a command to repeat 0xFF 0x00 0x00 from channel 1 to channel 255. A fade time of 3 seconds is set in the command. If you have RGB lights set up with their channels aligned to every 3rd address this will cause them to transition to be Red from whatever colour they currently are over the next 3 seconds. This will only occur on a single universe.

#### **Note**

Ideally DMX universes shouldn't use the full 512 channels because a smaller universe allows the DMX refresh rate to be higher than 44 Hz. Once you issue a fade task that changes values across the entire universe, you can't (currently) shrink it back down to a smaller universe.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x10	Command
3	0x01	Fade ID
4	0x00	Universe Mask Hi
5	0x01	Universe Mask Lo (Universe 1)
6	0x00	Start channel Hi
7	0x01	Start channel Lo (1)
8	0x02	Stop channel Hi
9	0x00	Stop channel Lo ((Hi << 8)   (Lo) = 512)
10	0x01	Address divisor 1 (distribute pattern value to every channel in range)
11	0x00	Block Mode - Intersection (See <i>DMX Channel Block Types</i> )
12	0x00	Personality Type - 8 Bit dimming. Default. See <i>DMX Personality Types</i>
13	0x00	Fade Time mode
14	0x00	Fade Time Hi
15	0x0B	Fade Time Mid
16	0xB8	Fade Time Lo (3000 milliseconds)
17	0x01	Fade Type A - Linear fade
18	0x00	Fade Type B - None
19	0x03	Data Length (3, for 3 channels, RGB)
20	0xFF	Red
21	0x00	Green
22	0x00	Blue
23	0x58	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA0	Response Type (OK)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

If there is an error response type and Data of 0x01 then this is likely because the system was busy, so try again and it may work the next time.

## QUERY\_GROUP\_BY\_NUMBER

Frame Type: *Basic*

Query Group information given a DALI Group Number. If there are no members of the group, the response will be (NO\_ANSWER).

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x12	Command
3	0x01	Group Number 0-15 (0=G0, 1=G1)
4	0x00	-
5	0x00	-
6	0x00	-
7	0x17	Checksum

### Response with Group Information

This response contains group data.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3	0x01	DALI Group Number
4	0x01	Group Occupancy Status (Boolean)
5	0xFE	Group Actual Level (254)
6	0xA2	Checksum

## QUERY\_SCENE\_BY\_NUMBER

**Frame Type:** *Basic*

Query Group information given a Scene Number. As this has no context of which group is associated, this shouldn't be used and is only left in for legacy purposes.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x13	Command
3	0x01	Scene Number
4	0x00	-
5	0x00	-
6	0x00	-
7	0x16	Checksum

### Response with Scene Information

This response contains scene data.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x02	Data Length
3	0x04	DALI Scene number
4	0x02	DALI Group number (or 0xFF for no group)
5	0xA5	Checksum

## QUERY\_SCENE\_NUMBERS\_BY\_ADDRESS

**Frame Type:** *Basic*

Query DALI Scene numbers associated with an DALI address. If a device has a level under 0xFF for a given scene, it will be listed in the response here. If the device has NO scenes, the answer will be NO\_ANSWER (0xA2)

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x14	Command
3	0x0F	Address (15)
4	0x00	-
5	0x00	-
6	0x00	-
7	0x1F	Checksum

### Response with Scene Numbers

This response contains three Scene numbers.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3	0x04	DALI Scene 4
4	0x07	DALI Scene 7
5	0x08	DALI Scene 8
6	0xA9	Checksum

## QUERY\_SCENE\_LEVELS\_BY\_ADDRESS

**Frame Type:** *Basic*

Query DALI Levels associated with an DALI address. All 16 scene values for a given control gear will be responded with. If a control gear has a value of 0xFF for the scene, it won't react (is not part of) that scene.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x02	Seq. Num
2	0x1E	Command
3	0x02	Address (2)
4	0x00	-
5	0x00	-
6	0x00	-
7	0x1A	Checksum

### Response with Scene Levels

This response contains all level values for the 16 dali scenes supported by a control gear.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x10	Data Length (16 bytes)
3	0x83	DALI Scene 0 Level
4	0x89	DALI Scene 1 Level
5	0xF4	DALI Scene 2 Level
6	0xE8	DALI Scene 3 Level
7	0xB8	DALI Scene 4 Level
8	0x08	DALI Scene 5 Level
9	0x49	DALI Scene 6 Level
10	0xA3	DALI Scene 7 Level
11	0xB7	DALI Scene 8 Level
12	0x62	DALI Scene 9 Level
13	0xC3	DALI Scene 10 Level
14	0x6E	DALI Scene 11 Level
15	0x28	DALI Scene 12 Level
16	0xFF	DALI Scene 13 Level (no scene)
17	0x17	DALI Scene 14 Level
18	0xEF	DALI Scene 15 Level
19	0xA9	Checksum

## QUERY\_GROUP\_MEMBERSHIP\_BY\_ADDRESS

Frame Type: *Basic*

Query the groups that a control gear is a member of.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x15	Command
3	0x0F	Address (15)
4	0x00	-
5	0x00	-
6	0x00	-
7	0x1E	Checksum

### Response with bitwise group membership

This response contains a 2 byte bitwise representation of current groups. This particular device is a member of Group 0.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x02	Data Length
3	0x00	Bitwise Membership (Groups 8-15)
4	0x01	Bitwise Membership (Groups 0-7)
5	0xA2	Checksum

## QUERY\_DALI\_ADDRESSES\_WITH\_INSTANCES

**Frame Type:** *Basic*

Query for DALI addresses that have instances associated with them. Due to payload restrictions, the TPI processor can't return 64 results in a single payload, therefore, a start address in `data_lo` must be specified. For example, you might typically run the command with start address 0 and then a further command with start address 60 to check for instances on the final 4 control devices.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x16	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	Start address of list to search (eg. 60)
7	0x12	Checksum

### Response with DALI addresses

This response contains three addresses. You can use [QUERY\\_INSTANCES\\_BY\\_ADDRESS](#) to get the instance information associated with an address returned from this request.

Byte Index	Byte Value	Description
0	0xA1	Response Type ( <code>REPLY_ANSWER</code> )
1	0x00	Seq. Num
2	0x03	Data Length
3	0x41	DALI address 65
4	0x47	DALI address 71
5	0x52	DALI address 82
6	0xA4	Checksum

## QUERY\_DMX\_DEVICE\_NUMBERS

**Frame Type:** *Basic*

DMX Devices are virtual devices configured manually within the controller. These devices have DALI-like metadata associated with them. They have a number that corresponds with a channel number.

Consider sending multiple requests if the first/previous returns with a `Data Length` of 64. The paging value should be larger than the cumulative number of channels previously returned.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x17	Command
3	0x00	-
4	0x00	-
5	0x00	Page Value Hi
6	0x00	Page Value Lo
7	0x13	Checksum

### Response with DMX Device Numbers

This response contains two DMX Device Numbers with a length of 2 bytes each. Number of 7 for the first and Number of 2 for the second.

Byte Index	Byte Value	Description
0	0xA1	Response Type ( <code>REPLY_ANSWER</code> )
1	0x00	Seq. Num
2	0x04	Data Length
3	0x00	DMX Device Number 1 Hi
4	0x07	DMX Device Number 1 Lo
5	0x00	DMX Device Number 2 Hi
6	0x02	DMX Device Number 2 Lo
7	0xA0	Checksum

## QUERY\_DMX\_DEVICE\_BY\_NUMBER

**Frame Type:** *Basic*

Query DMX Device information using a channel number.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x18	Command
3	0x00	-
4	0x00	-
5	0x00	DMX Device Number Hi
6	0x07	DMX Device Number Lo
7	0x1B	Checksum

### Response with DMX Device data

This response contains DMX Device data for DMX Channel 7.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x08	Data Length
3	0x00	DMX Channel Hi
4	0x01	DMX Channel Lo
5	0xFF	Level
6	0x00	Min Level
7	0xFF	Max Level
8	0xFF	Power On Level
9	0x00	DMX Channel Behaviours - (see <i>DMX Channel Behaviours</i> )
10	0x00	Universe number
11	0x57	Checksum

## QUERY\_DMX\_LEVEL\_BY\_CHANNEL

**Frame Type:** *Basic*

Query DMX Channel Level (and mode) by Channel number (1-512). If the mode is `DMX_BEHAVIOUR_TRIGGER` this represents a DMX value input/received. If the mode is `DMX_BEHAVIOUR_OUTPUT` this means the controller is sending this value.

### Warning

A level of `0x00` can be a default value and does not necessarily mean that data has been sent or received yet. A way around this may be to wait for a TPI Event that indicates the DMX should be active.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x19	Command
3	0x00	-
4	0x00	-
5	0x00	DMX Channel Hi
6	0x07	DMX Channel Lo
7	0x1A	Checksum

### Response with DMX Device data

This response contains the level for DMX Channel 7.

Byte Index	Byte Value	Description
0	0xA1	Response Type ( <code>REPLY_ANSWER</code> )
1	0x00	Seq. Num
2	0x02	Data Length
3	0x00	Mode - See <i>DMX Channel Behaviours</i>
4	0xFF	Level (255)
5	0xA3	Checksum

## QUERY\_DMX\_DEVICE\_LABEL\_BY\_NUMBER

**Frame Type:** *Basic*

Query the label for a DMX Device by Channel number (1 - 512). Universe can be specified in the Data Hi byte. Responds with NO\_ANSWER if label not found. Max 64 characters.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x20	Command
3	0x00	-
4	0x00	DMX Universe
5	0x00	DMX Device Channel Number High Byte
6	0x07	DMX Device Channel Number Low Byte
7	0x23	Checksum

### Response with DMX Device data

This response contains the label Foo for DMX Device with a channel of 7.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3 - 5	0x466F6F	Foo
6	0x8B	Checksum

## QUERY\_SCENE\_NUMBERS\_FOR\_GROUP

**Frame Type:** *Basic*

Query the scenes that a group has set up on the controller. Must be a named scene on the cloud. Group number 0-15 is required, not 64-79 as used in other commands.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x1A	Command
3	0x02	Group Number (0-15)
4	0x00	Unused
5	0x00	Unused
6	0x00	Unused
7	0x1C	Checksum

### Bitwise response of scenes that the group 2 is a member of

This response shows that group 2 is currently a member of scene 2 only

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x02	Data Length
3	0x00	Scene Membership (8-15)
4	0x04	Scene Membership (0-7)
5	0xA3	Checksum

## QUERY\_SCENE\_LABEL\_FOR\_GROUP

**Frame Type:** *Basic*

Query the label for a scene and group number combination. Must be set up in the cloud. If scene does not exist, will receive a REPLY\_NO\_ANSWER. Max 64 characters.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x1B	Command
3	0x02	Group Number (0-15)
4	0x02	Scene Number (0-12)
5	0x00	Unused
6	0x00	Unused
7	0x1F	Checksum

### Response of group string

This response contains the label Foo for the scene 2 group 2. Note that as with all label answers, the answer is size dependent on the string length.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3 - 5	0x466F6F	Foo
6	0x8B	Checksum

## QUERY\_CONTROLLER\_VERSION\_NUMBER

Frame Type: *Basic*

Query the 3 byte version number of the controller.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x1C	Command
3	0x00	Unused
4	0x00	Unused
5	0x00	Unused
6	0x00	Unused
7	0x18	Checksum

### Response of controller version

This response shows that the controller is v1.6.255

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3	0x01	Major Version
4	0x06	Minor Version
5	0xFF	Minor Minor Version
6	0x5A	Checksum

## QUERY\_CONTROL\_GEAR\_DALI\_ADDRESSES

Frame Type: *Basic*

Query the control gear addresses present in the database

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x1D	Command
3	0x00	Unused
4	0x00	Unused
5	0x00	Unused
6	0x00	Unused
7	0x19	Checksum

### Bitwise response of control gear present in database

This response shows address 0-9 are present on the controller

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x08	Data Length
3	0xFF	Bitwise Present (CG A0-7)
4	0x03	Bitwise Present (CG A8-15)
5	0x00	Bitwise Present (CG A16-23)
6	0x00	Bitwise Present (CG A24-31)
7	0x00	Bitwise Present (CG A32-39)
8	0x00	Bitwise Present (CG A40-47)
9	0x00	Bitwise Present (CG A48-55)
10	0x00	Bitwise Present (CG A56-63)
11	0x55	Checksum

## DALI\_INHIBIT

**Frame Type:** *Basic*

Inhibit sensors from changing DALI address 7 for 180 seconds.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xA0	Command
3	0x07	Address
4	0x00	-
5	0x00	Time Hi
6	0xB4	Time Lo
7	0x17	Checksum

### Response

Just OK. No extra data.

Byte Index	Byte Value	Description
0	0xA0	Response Type (OK)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

## DALI\_SCENE

**Frame Type:** *Basic*

Call a DALI Scene on an address. Use 0xFF to broadcast the scene across all addresses. Most likely you just want to call a scene on a particular group by adding 64 to the group number for Address.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xA1	Command
3	0xFF	Address
4	0x00	-
5	0x00	-
6	0x01	Scene number
7	0x5B	Checksum

### Response

Command has successfully sent and no answer was received in response to the command (this is expected behaviour for a command). A better response would be OK (0xA0) but we must maintain backwards compatibility.

Byte Index	Byte Value	Description
0	0xA2	Response Type (NO_ANSWER)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA2	Checksum

## DALI\_ARC\_LEVEL

**Frame Type:** *Basic*

Call a DALI Level on an address. Levels can be called on groups by adding 64 to the group number.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xA2	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x7F	level (127)
7	0xD8	Checksum

### Response

Command has successfully sent and no answer was received in response to the command (this is expected behaviour for a command). A better response would be OK (0xA0) but we must maintain backwards compatibility.

Byte Index	Byte Value	Description
0	0xA2	Response Type (NO_ANSWER)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA2	Checksum

## DALI\_ON\_STEP\_UP

**Frame Type:** *Basic*

Call a DALI On and Step Up on an address. If a device is off, it will turn it on. If a device is on, it will step up.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xA3	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xA6	Checksum

### Response

Command has successfully sent and no answer was received in response to the command (this is expected behaviour for a command). A better response would be OK (0xA0) but we must maintain backwards compatibility.

Byte Index	Byte Value	Description
0	0xA2	Response Type (NO_ANSWER)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA2	Checksum

## DALI\_STEP\_DOWN\_OFF

**Frame Type:** *Basic*

Call a DALI Down and Off on an address. If a device is at min, it will turn off. If a device is not yet at min, it will step down.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xA4	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xA1	Checksum

### Response

Command has successfully sent and no answer was received in response to the command (this is expected behaviour for a command). A better response would be OK (0xA0) but we must maintain backwards compatibility.

Byte Index	Byte Value	Description
0	0xA2	Response Type (NO_ANSWER)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA2	Checksum

## DALI\_UP

**Frame Type:** *Basic*

Call a DALI Up on an address.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xA5	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xA0	Checksum

### Response

Command has successfully sent and no answer was received in response to the command (this is expected behaviour for a command). A better response would be OK (0xA0) but we must maintain backwards compatibility.

Byte Index	Byte Value	Description
0	0xA2	Response Type (NO_ANSWER)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA2	Checksum

## DALI\_DOWN

**Frame Type:** *Basic*

Call a DALI Down on an address.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xA6	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xA3	Checksum

### Response

Command has successfully sent and no answer was received in response to the command (this is expected behaviour for a command). A better response would be OK (0xA0) but we must maintain backwards compatibility.

Byte Index	Byte Value	Description
0	0xA2	Response Type (NO_ANSWER)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA2	Checksum

## DALI\_RECALL\_MAX

**Frame Type:** *Basic*

Call a DALI Recall Max on an address.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xA7	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xA2	Checksum

### Response

Command has successfully sent and no answer was received in response to the command (this is expected behaviour for a command). A better response would be OK (0xA0) but we must maintain backwards compatibility.

Byte Index	Byte Value	Description
0	0xA2	Response Type (NO_ANSWER)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA2	Checksum

## DALI\_RECALL\_MIN

**Frame Type:** *Basic*

Call a DALI Recall Min on an address.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xA8	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xAD	Checksum

### Response

Command has successfully sent and no answer was received in response to the command (this is expected behaviour for a command). A better response would be OK (0xA0) but we must maintain backwards compatibility.

Byte Index	Byte Value	Description
0	0xA2	Response Type (NO_ANSWER)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA2	Checksum

## DALI\_OFF

**Frame Type:** *Basic*

Call a DALI Off on an address.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xA9	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xAC	Checksum

### Response

Command has successfully sent and no answer was received in response to the command (this is expected behaviour for a command). A better response would be OK (0xA0) but we must maintain backwards compatibility.

Byte Index	Byte Value	Description
0	0xA2	Response Type (NO_ANSWER)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA2	Checksum

## DALI\_QUERY\_LEVEL

**Frame Type:** *Basic*

Query the Arc Level for a DALI address. Dali level can be 0-254. The additional value of 255 represents as mixed levels. If the address does not exist in the database (or the group has no devices) the response will be 0. This is to bias any resulting decision to send commands to this unknown target as turning the light ON.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xAA	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xAF	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x01	Data Length
3	0xFE	Level (254)
4	0xA0	Checksum

## DALI\_QUERY\_CONTROL\_GEAR\_STATUS

Frame Type: *Basic*

Query the Status for control gear addresses 0-63. Note that 64-79 can be used for groups 0-15 and will produce a set bit if ANY of the group members have the bit set. Similarly, 127/255 can be used for broadcast status but this is only supported as of 1.9.180.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xAB	Command
3	0x01	Address (0 - 63 for CG0 - 63, 64 - 79 for G0-15, 127/255 for broadcast)
4	0x00	-
5	0x00	-
6	0x00	-
7	0xAE	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA1	Response Type ( <i>REPLY_ANSWER</i> )
1	0x00	Seq. Num
2	0x01	Data Length
3	0x04	Status (See <i>DALI Status Masks</i> )
4	0xA4	Checksum

## DALI\_QUERY\_CG\_TYPE

**Frame Type:** *Basic*

Query control gear device type information for a DALI address 0-63. does not work for groups or broadcast target. If the device does not exist, will return all zero.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xAC	Command
3	0x01	Address (0-63)
4	0x00	-
5	0x00	-
6	0x00	eDALI Byte
7	0xA9	Checksum

### Response

Response data is 4 bytes/32 bits long in little endian format. In the example, the CG would have device type 0 (FLOURESCENT) and 8 (COLOUR CONTROL)

#### Warning

Some programming languages/runtimes ([luajit for example](#)) do not reliably perform bitwise operations on numbers >24bits due to using floating point numbers for all numbers. Bitwise operations risk overwriting the [sign and exponent information](#). You may run into these issues when attempting to deal with this result as a single 32bit integer.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x04	Data Length
3	0x01	CG Device Type 0-7 membership
4	0x01	CG Device Type 8-15 membership
5	0x00	CG Device Type 16-23 membership
6	0x00	CG Device Type 24-31 membership
7	0xA5	Checksum

## DALI\_QUERY\_LAST\_SCENE

**Frame Type:** *Basic*

Query the last heard Scene for an address. Note that any changes to a single dali device that are done through group or broadcast scene commands do change the last heard scene for the dali address of the single device too. For example, if A10 is member of G0 and we sent a scene command to G0, A10 will show the same last heard scene as G0 (64).

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xAD	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xA8	Checksum

### Response with Scene Number

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x01	Data Length
3	0x07	Last Heard Scene Number
4	0xA0	Checksum

## DALI\_QUERY\_LAST\_SCENE\_IS\_CURRENT

Frame Type: *Basic*

Query if the last heard scene is the current active scene.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xAE	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xAB	Checksum

### Response with Boolean

0x01 indicates the True condition - the last heard scene is the currently active scene. 0x00 indicates the False condition, which is likely caused by an Arc Level or being issued to the address after the last Scene command.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x01	Data Length
3	0x01	Last Heard Scene is Current Scene (boolean)
4	0xA0	Checksum

## DALI\_QUERY\_MIN\_LEVEL

Frame Type: *Basic*

Query Minimum Level for an Address.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xAF	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xAA	Checksum

### Response with Minimum Level

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x01	Data Length
3	0x01	Min Level (1)
4	0xA1	Checksum

## DALI\_QUERY\_MAX\_LEVEL

Frame Type: *Basic*

Query Maximum Level for an Address.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xB0	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xB5	Checksum

### Response with Maximum Level

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x01	Data Length
3	0xFE	Max Level (254)
4	0x5E	Checksum

## DALI\_QUERY\_FADE\_RUNNING

Frame Type: *Basic*

Query Fade running on an address.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xB1	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xB4	Checksum

### Response with Boolean

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x01	Data Length
3	0x00	Fade is running (False)
4	0xA0	Checksum

## DALI\_ENABLE\_DAPC\_SEQ

**Frame Type:** *Basic*

Begin a DALI Direct Arc Power Control (DAPC) Sequence.

DAPC allows overriding of the fade rate. This allows levels to be immediately set. A DAPC sequence will be continued for 250 milliseconds. If no Arc-levels are received for 250 milliseconds then the DAPC sequence ends and fade rates will apply again.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xB2	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xB7	Checksum

### Response

Command has successfully sent and no answer was received in response to the command (this is expected behaviour for a command). A better response would be OK (0xA0) but we must maintain backwards compatibility.

Byte Index	Byte Value	Description
0	0xA2	Response Type (NO_ANSWER)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA2	Checksum

## QUERY\_DALI\_EAN

**Frame Type:** *Basic*

Query for a Control Device or Control Gear European Article Number (EAN) also known as a GTIN.

It's important to note that for Control Devices, `Address` must be offset by + 64. For Control Gear the normal address between 0 and 63 is used.

### Request

This is a request for the control gear EAN at DALI address 1.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xB8	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xBD	Checksum

### Response with EAN

Example contains the EAN for the product described as a zencontrol Wireless PIR sensor with relay and 2 inputs. This data converted to a decimal integer is 6971103532931.

Byte Index	Byte Value	Description
0	0xA1	Response Type ( <code>REPLY_ANSWER</code> )
1	0x00	Seq. Num
2	0x06	Data Length (always 6 for EAN)
3	0x65	-
4	0x71	-
5	0x62	-
6	0x65	-
7	0x78	-
8	0x03	-
9	0xCE	Checksum

## QUERY\_DALI\_SERIAL

**Frame Type:** *Basic*

Query for a Control Device or Control Gear serial number.

It's important to note that for Control Devices, `Address` must be offset by + 64. For Control Gear the normal address between 0 and 63 is used.

### Request

This is a request for the control gear serial number at DALI address 1. Dali serial numbers are 8 bytes in size.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xB9	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xBC	Checksum

### Response with Serial Number

Example contains the serial number for the Control Gear with DALI address 1. The example shows a serial number of 0x12345678912345 or 5124095575401285

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x08	Data Length (always 8 for serial number)
3	0x12	MSB Serial Number
4	0x34	-
5	0x56	-
6	0x78	-
7	0x91	-
8	0x23	-
9	0x34	-
10	0x45	LSB Serial Number
11	0x27	Checksum

## QUERY\_VIRTUAL\_INSTANCES

**Frame Type:** *Basic*

Query Virtual Instances and their types.

Virtual Instances / Virtual Switches are a paid addon. See [Licenses](#) for more information.

### Request

Query to see the virtual instances on this controller. If there are no instances, response will be NO\_ANSWER. A paging value of the number of instances already received can go in `data_mid` and `data_lo` bytes, though this is unlikely to be needed as no controller currently implements so many virtual instances that it would be necessary.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xB6	Command
3	0x00	-
4	0x00	-
5	0x00	Paging value Hi
6	0x00	Paging value Lo
7	0xB7	Checksum

### Response with Virtual Instance metadata

Responses return 0 or more 2 byte Virtual Instance descriptions.

This example shows a single instance returned. Multiple instances will result in a `Data Length` that will be a multiple of 2 and the fields will repeat in a predictable pattern.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x02	Data Length (divide by 2 to get number of Instances)
3	0x00	Virtual Instance Number
4	0x01	Instance Type. (See <a href="#">Instance Types</a> )
5	0xA3	Checksum

## VIRTUAL\_INSTANCE

**Frame Type:** *Basic*

Trigger a binary action on a Virtual Instance given a virtual instance number and an action.

Virtual Instances / Virtual Switches are a paid addon. See [Licenses](#) for more information.

If the instance does not exist, response will be ERROR with INVALID\_ARGS.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xB3	Command
3	0x01	Virtual Instance number
4	0x00	-
5	0x00	-
6	0x01	Instance Action - See <a href="#">Instance Binary States</a>
7	0xB6	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA0	Response Type (OK)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

## DALI\_CUSTOM\_FADE

**Frame Type:** *Basic*

Run a fade to a level on a DALI address with a custom fade time in seconds. If target is a group or broadcast target, the member devices in the target MUST have the same arc value or unexpected results will occur. If a lighting command is sent to the same target, the custom fade will be stopped.

Whilst this feature could theoretically handle constituent targets at different levels, it must be respected that any difference in arc levels on members of a target means that the fade has to be split into up to 64 parallel fades, which is not easy for dali to be able to service. If you require different levels, dali provides the SCENE concept, which allows you to call a scene, where each device has a stored arc level to internally conduct a fade to.

### Request

Fade to level 0 over 10 seconds.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xB4	Command
3	0x01	Address
4	0x00	Target Level
5	0x00	Seconds Hi Byte
6	0x0A	Seconds Lo Byte (10 Seconds)
7	0xBB	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA0	Response Type (OK)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

## DALI\_GO\_TO\_LAST\_ACTIVE\_LEVEL

Frame Type: *Basic*

Command a DALI Address to go to its "Last Active" level

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xB5	Command
3	0x01	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xB0	Checksum

### Response

Command has successfully sent and no answer was received in response to the command (this is expected behaviour for a command). A better response would be OK (0xA0) but we must maintain backwards compatibility.

Byte Index	Byte Value	Description
0	0xA2	Response Type (NO_ANSWER)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA2	Checksum

## QUERY\_DALI\_INSTANCE\_LABEL

Frame Type: *Basic*

Query the Label for a DALI Instance given a Control Device and an Instance Number. Max 64 characters.

### Request

Query the label for control device address 1 (65 or 0x41), instance 1 (second instance)

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xB7	Command
3	0x41	Control Device Address (64-127 CD0-64)
4	0x00	-
5	0x00	-
6	0x01	Instance number (0-31)
7	0xF3	Checksum

### Response with Label

A label of Foo is returned.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3 - 5	0x466F6F	Foo
6	0x8B	Checksum

## CHANGE\_PROFILE\_NUMBER

**Frame Type:** *Basic*

Request a profile change on the controller. A profile change may not always be successful due as some profiles can't override others. Eg. an Emergency profile can't be overridden by a regular scheduled profile.

Profile numbers are unique across sites, and are up to two-bytes long.

A profile number of 0xFFFF will request a profile change to the profile determined by the schedule.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xC0	Command
3	0x00	-
4	0x00	-
5	0x00	Profile Number Hi
6	0xD1	Profile Number Lo
7	0x15	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA0	Response Type (REPLY_OK)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

A failure to schedule will reply with a response type of REPLY\_ERROR and ERROR\_CMD\_REFUSED as data.

## QUERY\_INSTANCE\_GROUPS

**Frame Type:** *Basic*

Request Group targets associated with an instance. There are always 3 replies to any valid instance.

1. Primary
2. First
3. Second

The Primary group typically represents where the physical device resides.

A group number of 0xFF (255) indicates that no group has been configured.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x21	Command
3	0x64	Address
4	0x00	-
5	0x00	-
6	0x01	Data Lo: Instance Number
7	0x40	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3	0x01	Primary Group
4	0x02	First Group
5	0xFF	Second Group
6	0x5E	Checksum

The Second group has a value of 0xFF, therefore should be ignored.

## QUERY\_DALI\_FITTING\_NUMBER

**Frame Type:** *Basic*

Request the fitting number string (eg. 1.2) for control gear and control devices. Note: does not check for validity. If device is not named or does not exist, you get a default identifier of Controller ID.Dali Address for control gear and Contoller ID.Dali address + 100 for control devices.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x22	Command
3	0x01	Address (0-63 CG, 64-127 CD)
4	0x00	-
5	0x00	-
6	0x00	-
7	0x27	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3	0x31	Data
4	0x2E	Data
5	0x32	Data
6	0xBD	Checksum

The fitting number returned is a string of 1.2.

## QUERY\_DALI\_INSTANCE\_FITTING\_NUMBER

**Frame Type:** *Basic*

Request the fitting number string (eg. 1.2.0) for an instance.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x23	Command
3	0x99	Address (64-127 for CDA0-A64)
4	0x00	-
5	0x00	-
6	0x01	Instance number (0-31)
7	0xBF	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3	0x31	Data
4	0x2E	Data
5	0x32	Data
6	0x8D	Checksum

The fitting number returned is a string of 1.2.

## QUERY\_CONTROLLER\_LABEL

Frame Type: *Basic*

Request the label for the controller. Max 64 characters.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x24	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	-
7	0x20	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3	0x44	Data
4	0x6F	Data
5	0x67	Data
6	0xEE	Checksum

The controller returns a label of Dog.

## QUERY\_CONTROLLER\_FITTING\_NUMBER

**Frame Type:** *Basic*

Request the fitting number string (eg. 90) for the controller itself. This should be the same as the first segment of a Control Device/Control Gear fitting number.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x25	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	-
7	0x21	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x01	Data Length
3	0x31	Data
4	0x91	Checksum

The fitting number returned is a string of 1.

## QUERY\_IS\_DALI\_READY

**Frame Type:** *Basic*

Query whether the DALI line is ready, or has a fault. Will reply REPLY\_OK if DALI ready, or an error otherwise.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x26	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	-
7	0x22	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA3	Response Type (REPLY_ERROR)
1	0x00	Seq. Num
2	0x01	Data Length
3	0x00	Data (ERROR_SHORT_CIRCUIT)
4	0xA2	Checksum

The response indicates DALI is not ready because the Response Type is not 0xA0/REPLY\_OK.

## QUERY\_CONTROLLER\_STARTUP\_COMPLETE

**Frame Type:** *Basic*

Query whether the controller has finish its startup sequence. Will reply `REPLY_OK` if ready, or `REPLY_NO_ANSWER` otherwise. Waiting for the startup sequence to complete is particularly important if you wish to perform queries about DALI. The more devices on a DALI line the longer startup will take to complete. The startup sequence performs DALI queries such as device type, current arc-level, GTIN, serial number, etc. For a line with only a handful of devices expect it to take approximately 1 minute.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x27	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	-
7	0x23	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA0	Response Type ( <code>REPLY_OK</code> )
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

This response indicates that the startup sequence has completed.

## OVERRIDE\_DALI\_BUTTON\_LED\_STATE

**Frame Type:** *Basic*

Override the current DALI push button LED state. The LED is targeted using the associated button DALI address and instance number. The desired LED state (eg. On or Off) is an *Instance Binary State*.

### Request

Set/Override the LED for button at DALI address 112 on Instance Number 1 to On (instance binary state On is 0x02).

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x29	Command
3	0x70	Address
4	0x00	-
5	0x02	Instance Binary State
6	0x01	Instance Number
7	0x5E	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA0	Response Type (REPLY_OK)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

This response indicates that an override command was issued.

## QUERY\_LAST\_KNOWN\_DALI\_BUTTON\_LED\_STATE

**Frame Type:** *Basic*

Query the last known DALI push button LED state. The LED is targeted using the associated button DALI address and instance number. It should be noted that this will only work for led modes where the controller or a TPI caller is managing the LED state. In many cases, the control device itself can manage its own led. The LED state returned (eg. On, Off or Unknown) is an *Instance Binary State*.

### Note

The “last known” LED state may not be the actual physical LED state. To help confirm actual LED state you can query the state of DALI devices that may indicate what the LED state **should** be.

### Request

Query LED for button at DALI address 112 (Control Device Address 48) on Instance Number 1 (the second instance of this device).

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x30	Command
3	0x70	Address
4	0x00	-
5	0x00	-
6	0x01	Instance Number
7	0x45	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x01	Data Length
3	0x01	LED state is <i>Off</i>
4	0xA0	Checksum

This response indicates that the LED state is Off.

## DALI\_STOP\_FADE

**Frame Type:** *Basic*

Sends a DALI STOP FADE dali command to an address/group/broadcast (DAPC level 0xFF). Dali STOP FADE will stop any direct arc or scene fades current on the device.

This command is also able to stop custom/emulated DALI fades from the DALI\_CUSTOM\_FADE command but the target must be the same target as the CUSTOM\_FADE was instigated for. For example, it is not possible stop a CUSTOM\_FADE on a single address which is fading as part of a CUSTOM\_FADE, instigated via a group or broadcast target. This would require subtracting the target from fade and changing to fading individual devices. If you have a group of 20 devices and you instruct the system to subtract one, each direct arc command must now be sent out 19 times (to individual devices). This would likely produced significant degradation in the fade.

### Request

Stop the custom fade on Address 0.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0xC1	Command
3	0x00	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0xC5	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA0	Response Type (REPLY_OK)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

## QUERY\_DALI\_COLOUR\_FEATURES

**Frame Type:** *Basic*

Query DALI Colour Features. Features can also be described as capabilities. The byte returned indicates the colour types that the light is capable of using (eg. tuneable white, RGBWAF, PRIMARY\_N) and the channel count and number of primaries.

### Request

Query a DALI Control Gear with type COLOUR\_CONTROL on Address 0.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x35	Command
3	0x00	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0x31	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x01	Data Length
3	0x83	Data
3	0xA0	Checksum

The response of 0x83 can be represented as 0b10000011. Reading from right to left it can be decoded as the following:

Bits	Position Index	Description
1	0	This light is capable of CIE 1931 XY Coordinates.
1	1	This light is capable of colour temperature (Kelvin) for tuneable white.
000	2 to 4	0b000 is decimal 0. This light has no primaries.
100	5 to 7	0b100 is decimal 4. This light has 4 channels in RGBWAF mode (so it's a RGBW light).

#### **Note**

Some lights may only support tuneable white and no other colour capabilities. Accepting CIE1931 XY does mean it necessarily has full-colour support as colour temperature can also be expressed using XY. If a light supports RGBWAF channels it's reasonable to assume it supports full-colour.

## QUERY\_DALI\_COLOUR\_TEMP\_LIMITS

**Frame Type:** *Basic*

Query DALI Colour Temperature maximum, minimum and step value in Kelvin. Both the Physical limits and the configured limits are returned.

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x38	Command
3	0x00	Address
4	0x00	-
5	0x00	-
6	0x00	-
7	0x3C	Checksum

### Response

The response contains a physical warmest of 1000K, physical coolest of 6000K, configured warmest of 2000K, configured coolest of 6000K and a step value of 500K.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x0A	Data Length
3	0x03	Physical Warmest Upper Byte
4	0xE8	Physical Warmest Lower Byte
5	0x17	Physical Coolest Upper Byte
6	0x70	Physical Coolest Lower Byte
7	0x07	Soft Warmest Upper Byte
8	0xD0	Soft Warmest Lower Byte
9	0x17	Soft Coolest Upper Byte
10	0x70	Soft Coolest Lower Byte
11	0x01	Step Value Upper Byte
12	0xF4	Step Value Lower Byte
13	0x62	Checksum

## SET\_SYSTEM\_VARIABLE

**Frame Type:** *Basic*

Set a system variable. On V2.1 pro controllers, there are 148 system variables. For previous and non pro controllers there are 48 system variables.

System variables in V2.1 are signed 32bit and have magnitude. For compatibility reasons, a new command will be created to take advantage of this. Any usage of this command is limited to 16bit and will set zero magnitude.

### Request

Set system variable 3 to 0xFFFE. The variable number goes in the `address` byte and the value to set is split across `data_mid` and `data_lo`.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x36	Command
3	0x03	Variable Number
4	0x00	-
5	0xFF	Data Mid (Upper byte)
6	0xFE	Data Lo (Lower byte)
7	0x30	Checksum

### Response

Byte Index	Byte Value	Description
0	0xA0	Response Type (REPLY_OK)
1	0x00	Seq. Num
2	0x00	Data Length
3	0xA0	Checksum

## QUERY\_SYSTEM\_VARIABLE

**Frame Type:** *Basic*

Query a system variable. After V2.1, pro controllers have been extended to support 148 system variables. For previous and non pro controllers there are 48 system variables.

System variables in V2.1 are signed 32bit and have magnitude. For compatibility reasons, a new command will be created to take advantage of this.

### Request

Query system variable 3.

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x37	Command
3	0x03	Variable Number
4	0x00	-
5	0x00	-
6	0x00	-
7	0x30	Checksum

### Response

The result of the query to system variable 3 is 0xFFFE.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x02	Data Length
3	0xFF	Data (Hi byte)
4	0xFE	Data (Lo byte)
5	0x5C	Checksum

## QUERY\_SYSTEM\_VARIABLE\_NAME

**Frame Type:** *Basic*

Query the Label for a system variable. Pro controllers only. Max 64 characters.

### Request

Query the label for system variable index 16 (0x10)

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x00	Seq. Num
2	0x42	Command
3	0x10	System variable number (0-147)
4	0x00	-
5	0x00	-
6	0x00	-
7	0x56	Checksum

The controller returns a label of Dog.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x00	Seq. Num
2	0x03	Data Length
3	0x44	Data
4	0x6F	Data
5	0x67	Data
6	0xEE	Checksum

## QUERY\_PROFILE\_INFORMATION

**Frame Type:** *Basic* QUERY\_PROFILE\_NUMBERS

Query of current profile information and numbers. Supersedes *QUERY\_PROFILE\_NUMBERS*

Note: Behaviour contains the following information:

Bit 0 - is disabled (1 for is disabled, 0 enabled)

Bit 1-2 - Profile priority (where 0 - scheduled, 1 - higher, 2 - even higher etc)

### Request

A controller has 4 profiles (0, 100, 200, 400). For the purposes of illustration, behaviours are set to random 8bit numbers but would only have bits 0-2 set in real world.

The current active profile number gives the actual profile the controller is in. This will generally be the scheduled profile but is subject to the controller being in a higher priority profile, preventing the controller from being in the scheduled profile.

The last scheduled profile number is the profile that the controller should be in if it is not in a higher priority profile.

The last overridden profile UTC is the time if a controller entered a non scheduled profile.

The last scheduled profiled UTC is the time of the last scheduled profile change (regardless of whether it has succeeded)

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x46	Seq. Num
2	0x43	Command
3	0x00	-
4	0x00	-
5	0x00	-
6	0x00	-
7	0x01	Checksum

The controller returns a label of Dog.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x46	Seq. Num
2	0x1B	Data Length
3	0x00	Current Active Profile Number (High Byte)
4	0x00	Current Active Profile Number (Low Byte)
5	0x00	Last Scheduled Profile Number (High Byte)
6	0x64	Last Scheduled Profile Number (Low Byte)
7-10	0x22334455	Last Overridden Profile UTC High byte first
11-14	0x44556677	Last Scheduled Profile UTC High byte first
15-16	0x0000	Profile Number 0
17	0xE8	Profile Behaviour
18-19	0x0064	Profile Number 100
20	0xDF	Profile Behaviour
21-22	0x00C8	Profile Number 200
23	0x37	Profile Behaviour
24-25	0x0190	Profile Number 400
26	0xF5	Profile Behaviour
27-28	0x01F4	Profile Number 500
29	0xCB	Profile Behaviour
30	0x6E	Checksum

## QUERY\_COLOUR\_SCENE\_MEMBERSHIP\_BY\_ADDR

**Frame Type:** *Basic*

Query a list of scenes with colour change data for an address. Will respond with NO\_ANSWER if the device does not have the colour control device type or no scenes with data are found

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x66	Seq. Num
2	0x44	Command
3	0x1E	Control Gear address A30
4	0x00	-
5	0x00	-
6	0x00	-
7	0x38	Checksum

### Response with Scenes

This response contains a list of any scene that has colour data.

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x66	Seq. Num
2	0x08	Data Length (number of groups in this case)
3	0x00	Scene 0 has data
4	0x01	Scene 1 has data
5	0x02	Scene 2 has data
6	0x03	Scene 3 has data
7	0x04	Scene 4 has data
8	0x05	Scene 5 has data
9	0x08	Scene 8 has data
10	0x09	Scene 9 has data
11	0xAB	Checksum

## QUERY\_COLOUR\_SCENE\_0\_7\_DATA\_FOR\_ADDR

## QUERY\_COLOUR\_SCENE\_8\_11\_DATA\_FOR\_ADDR

### Frame Type: *Basic*

Query the colour control data for scenes 0-7/8-11. Answer will be NO\_ANSWER if device does not have a colour control device type. Data is in 7 byte segments, which comprise of scene colour type and 6 bytes of data. All scenes are populated, regardless of whether they have data.

Colour type 0x10 = XY, data = High Byte X, Low Byte X, High Byte Y, Low Byte Y, 2 bytes unused data 0xFF

Colour type 0x20 = TC (kelvin), data = High Byte, Low Byte in Kelvin, 4 bytes of unused data 0xFF.

Colour type 0x80 = RGBWAF, data = r, g, b, w, a, f

Colour type 0xFF = Unused scene, data = 6 bytes 0xFF

Colour temperatures are subject to minimum and maximum colour temperatures set on device. Scene values that are given by the responses to this command have already had their scene values constrained by these boundaries.

RGBWAF values of 0xFF indicate “no change” to the current colour.

The response will contain the scene data for scenes 0-7 (0x45) / 8-11 (0x46)

### Request

Byte Index	Byte Value	Description
0	0x04	Start Byte TPI Advanced
1	0x2B	Seq. Num
2	0x45	Command
3	0x1B	Control Gear address A27
4	0x00	-
5	0x00	-
6	0x00	-
7	0x71	Checksum

### Response with Scenes

The response will contain the scene data for scenes 0-7 (0x45) / 8-11 (0x46)

Byte Index	Byte Value	Description
0	0xA1	Response Type (REPLY_ANSWER)
1	0x66	Seq. Num
2	0x08	Data Length (number of groups in this case)
3	0x10	Scene 0 is an XY Scene
4	0x64	Scene 0 X High Byte
5	0x9B	Scene 0 X Low Byte
6	0x81	Scene 0 Y High Byte
7	0xCF	Scene 0 Y Low Byte
8	0xFF	Scene 0 Unused Data
9	0xFF	Scene 0 Unused Data
10	0x20	Scene 1 is a Tc Scene (colour temperature)
11	0xAA	Scene 1 Tc High Byte (kelvin)
12	0xBB	Scene 1 Tc Low Byte (kelvin)
13	0xFF	Scene 1 Unused Data
14	0xFF	Scene 1 Unused Data
15	0xFF	Scene 1 Unused Data
16	0xFF	Scene 1 Unused Data
10	0x80	Scene 2 is an RBWAF Scene
11	0x01	Scene 2 Red
12	0x02	Scene 2 Green
13	0x03	Scene 2 Blue
14	0xFF	Scene 2 White (no change)
15	0x04	Scene 2 Amber
16	0xFF	Scene 2 Freecolour (no change)
...	...	Continued for scenes 3-7
59	0x-	Checksum

## BUTTON\_PRESS\_EVENT and BUTTON\_HOLD\_EVENT

**Frame Type:** *TPI Events Frame*

**Link to Event Types:** *Event Types*

A button press event broadcast over UDP. If event is HOLD\_EVENT, event type will be 0x02.

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x007B	Target - Control Device DALI Address 59 (+64 for Control devices)
10	0x00	Event Type - BUTTON_PRESS_EVENT
11	0x01	Data Length
12	0x05	(Data) Instance number. Useful for identifying the exact button on a keypad
13	0x2D	Checksum

## ABSOLUTE\_INPUT\_EVENT

**Frame Type:** *TPI Events Frame*

**Link to Event Types:** *Event Types*

An absolute input has reported a change in input value. In the example below, Absolute input Control Device A59, instance number 5 has reported 0xAABB for its input value.

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x007B	Target - Control Device DALI Address 59 (+64 for Control devices)
10	0x02	Event Type - ABSOLUTE_INPUT_EVENT
11	0x03	Data Length
12	0x05	(Data) Instance number. Useful for identifying the exact button on a keypad
13	0xAA	Input value high byte
14	0xBB	Input value low byte
15	0x3C	Checksum

## LEVEL\_CHANGE\_EVENT

**Frame Type:** *TPI Events Frame*

**Link to Event Types:** [Event Types](#)

A DALI Level change event on DALI target 59.

Whilst other events use the target to indicate group via the 64-79 range, these are legacy commands and cannot be extended with compromising existing implementations.

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x003B	Target - Actual DALI Address (59) / Group
10	0x03	Event Type - LEVEL_CHANGE_EVENT
11	0x01	Data Length
12	0xFE	(Data) DALI Arc Level
13	0x95	Checksum

## GROUP\_LEVEL\_CHANGE\_EVENT

**Frame Type:** *TPI Events Frame*

**Link to Event Types:** [Event Types](#)

A DALI Level change event on DALI group 6.

Whilst other events use the target to indicate group via the 64-79 range, these are legacy commands and cannot be extended with compromising existing implementations.

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x0006	Target - Group 6
10	0x04	Event Type - GROUP_LEVEL_CHANGE_EVENT
11	0x01	Data Length
12	0xF0	(Data) DALI Arc Level
13	0xA1	Checksum

## SCENE\_CHANGE\_EVENT

**Frame Type:** *TPI Events Frame*

**Link to Event Types:** *Event Types*

Last Heard Scene tracks the most recent scene command sent to the target. This value persists through controller reboots and remains unchanged by other lighting commands. At Scene is a binary flag (1/0) indicating whether the target is still in its last scene state or has received other lighting commands since.

Example:

Send "GO TO SCENE 4" → last\_heard\_scene = 4, at\_scene = 1

Send "OFF" command → last\_heard\_scene = 4, at\_scene = 0

The scene number persists, but the flag shows whether you've deviated from that scene.

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x007B	Target - Control Gear DALI Address 59
10	0x05	Event Type - SCENE_CHANGE_EVENT
11	0x02	Data Length
12	0x05	(Data) Last Heard Scene Number (0-15)
13	0x01	(Data) Target is AT the scene (1/0)
14	0x2A	Checksum

## OCCUPANCY\_EVENT

**Frame Type:** *TPI Events Frame*

**Link to Event Types:** *Event Types*

An occupancy sensor has reported a motion detected event. For individual control devices, we do not provide “not detected” events. If you require this, you must run a timeout from your own software. Please see [GROUP\\_OCCUPANCY\\_EVENT](#) as an alternative if you are only monitoring groups.

In the example CD A59 Instance number 5 has reported motion.

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x007B	Target - Control Device DALI Address 59 (+64 for Control devices)
10	0x06	Event Type - Occupancy detected
11	0x02	Data Length
12	0x05	(Data) Instance number. Useful for identifying the exact sensor
13	0x01	Unneeded data
14	0x29	Checksum

## SYSTEM\_VARIABLE\_CHANGED\_EVENT

**Frame Type:** *TPI Events Frame*

**Link to Event Types:** *Event Types*

A system variable value has changed. In the example, system variable 32 has been changed to the value  $-200$  with magnitude of  $-1$  (implying that the actual system variable value is  $20$  as the value is  $-200 * 10^{-1} = -20$ ). It should be noted expansion of the system variable system to include magnitude is principally targeted at obtaining values of sensors for a variety of applications and that in principle, anything else would likely be best left at zero magnitude.

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x0020	Target - System variable index 32
10	0x07	Event Type - SYSTEM_VARIABLE_CHANGED_EVENT
11	0x05	Data Length
12 - 15	0xFFFFFFFF38	(Data) 1st - 4th byte (big endian). Value of -200
16	0xFF	Magnitude (int8) of $-1$ ( $10^{-1}$ )
17	0x48	Checksum

## COLOUR\_CHANGED\_EVENT

**Frame Type:** *TPI Events Frame*

**Link to Event Types:** *Event Types*

Can support control gear addresses (0-63) and groups (64-79)

Keep in mind, there are multiple types of colour data such as Colour Temperature (in Kelvin) and CIE 1931 XY coordinates. If a fixture is just RGB or RGBW (and not RGBWAF) then the data length will be equal to the number of channels + 1.

A DALI RGBWAF colour change event on DALI target 59.

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x003B	Target - DALI Address (59)
10	0x08	Event Type - Colour Change
11	0x07	Data Length
12	0x80	(Data) DALI RGBWAF - See <i>Colour Type</i>
13	0xFF	R - Red Byte
14	0x00	G - Green Byte
15	0x00	B - Blue Byte
16	0x00	W - White Byte
17	0x00	A - Amber Byte
18	0x00	F - Freecolour Byte
19	0x19	Checksum

A Colour Temperature change event on DALI target 59.

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x003B	Target - DALI Address (59)
10	0x08	Event Type - Colour Change
11	0x03	Data Length
12	0x20	(Data) Colour Mode TC - See <i>Colour Type</i>
13	0xFF	Kelvin - Hi Byte
14	0x00	Kelvin - Lo Byte
15	0xBD	Checksum

## PROFILE\_CHANGED\_EVENT

**Frame Type:** *TPI Events Frame*

**Link to Event Types:** *Event Types*

When a controller profile changes (eg. After Hours) an event will be emitted. This shows a profile change to Profile 15.

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x0000	Target / Unused for Profile Change
10	0x09	Event Type - Profile Event Change
11	0x02	Data Length
12	0x00	(Data) Profile Hi Byte
13	0x0F	Profile Lo Byte
14	0x56	Checksum

## GROUP\_OCCUPANCY\_EVENT

**Frame Type:** *TPI Events Frame*

**Link to Event Types:** *Event Types*

Group occupancy state when a sensor with a group target is triggered (and causes the group to be newly considered occupied) or the amount of time defined as “seconds to unoccupied” has ticked for a group. The default value of “seconds to unoccupied” is 30 seconds but may be changed in the grid control in the Advanced section from V2.2 onwards. Does not send occupied on every trigger from sensor, only on state changes. If a group has already been set to occupied, you will not hear a message until 30 seconds without triggers occurs.

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x0040	Target - Group 0 (+64 for Groups)
10	0x0A	Event Type - Occupancy detected
11	0x02	Data Length
12	0xFF	Instance (unused)
13	0x01	Occupied = 1, Not occupied = 0
14	0xE4	Checksum

## LEVEL\_CHANGE\_EVENT\_V2

**Frame Type:** *TPI Events Frame*

**Link to Event Types:** [Event Types](#)

V2.001.121 or newer controller required. A newer DALI level change event that provides context on whether the target is currently dimming to a value. Supports both address (A0-63 = 0-63) / group targets (G0-G15 = 64-79). To maintain backwards compatibility, is currently sent as well as the original level change events. Please set a filter for the original event, if traffic is a concern.

Example: A59 has begun to dim to 0

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x003B	Target - Actual DALI Address (59) / Group
10	0x0B	Event Type - LEVEL_CHANGE_EVENT_V2
11	0x02	Data Length
12	0xFE	(Data) DALI Arc Level
13	0x00	(Data) Arc Level dimming to
14	0x9E	Checksum

Example: A59 has completed dimming to 0

Byte Index	Byte Value	Description
0 - 1	0x5A43	Literally capitals 'ZC'.
2 - 7	0x7CBACC2F402E	MAC Address
8 - 9	0x003B	Target - Actual DALI Address (59) / Group
10	0x0B	Event Type - LEVEL_CHANGE_EVENT_V2
11	0x02	Data Length
12	0x00	(Data) DALI Arc Level
13	0x00	(Data) Arc Level Dimming to (should be equal to arc for no dim)
14	0x60	Checksum

# Changelog

Date	Min Controller	Change
20/11/2025	v2.2.32	Controller now supports TCP communication
20/11/2025	v2.2.32	Fixed DALI Colour Request Colour Data length (previous size only failed on TCP)
19/09/2025	v2.2.13	Fixed SYSTEM VARIABLE EVENT example event_type (incorrectly 0x5, should be 0x7).
19/09/2025	v2.2.13	Expanded SCENE_CHANGE_EVENT to have AT_SCENE data (packet data is now 2 bytes)