Controller API

Release 6.0

Carallon Ltd

Aug 09, 2023

CONTENTS

Welcome to the API documentation for Mosaic Designer controllers.

If you're new here then you might start with the *introduction*, otherwise you might want to read about *what's new*.

ONE

INTRODUCTION

Mosaic Designer controllers offer *HTTP* and *Lua* APIs providing access to system information, playback functions and trigger operations.

In addition, a small *JavaScript library* is hosted on the controller's web server, which wraps the HTTP requests of the web API and also provides a mechanism to subscribe to the controller's websocket channels via callbacks.

WEB API AUTHENTICATION

If the controller has security setup then some endpoints of the HTTP API and some functions in the JavaScript library will require clients to authenticate in order to authorise the requests.

2.1 Authentication Methods

Two methods for authenticating users of the Web API are supported:

- *Cookie Authentication*: the default when using the API and/or query.js library in a custom web interface.

- *Token Authentication*: used with HTTP API requests, typically when the client is not a web browser.

With both methods, a new token, valid for 5 minutes, is returned from each authenticated request. If the user, or API client, is inactive for longer than 5 minutes then the cookie or token expires, requiring a username and password to be provided again.

2.1.1 Cookie Authentication

Cookie authentication is typically used by the controller's web interface (either the default web interface or a custom web interface in a project).

Cookie authentication works with both the HTTP API and the query.js library.

A cookie is returned by the controller in response to a *POST* request to the `/authenticate` endpoint when the `original_url` is provided as a cookie or a query parameter. This is the endpoint used by the default login page whenever a user signs in.

The cookie is stored by a web browser automatically, and the browser then sends this cookie with subsequent requests to authenticate the user. The response from each authenticated request will update this cookie with a new token, valid for 5 minutes. If no authenticated requests are made for 5 minutes then the token in the cookie will expire and the `/authenticate` endpoint must be used to get a new token.

The cookie can be removed by making a *GET* request to the `/logout` endpoint, which can be done simply by navigating the browser to that endpoint.

Custom Login Page

Normally, a user will sign into the controller using the login page of the default web interface, which is shown if a user tries to visit a page that they don't have access to. In a custom web interface, uploaded as part of a project, a custom login page can be configured with the `LoginFile` directive in the `.webconfig` file of the custom web interface. This custom login page is then shown instead of the default login page when a user tries to visit part of a custom web interface that they don't have access to.

Typically a login page will be an HTML page with a form element containing fields for the username and password. The HTML snippet below can be used to generate a form with these fields:

```html
<form action="/authenticate" method="POST">
  <input type="text" name="user">
  <input type="password" name="password">
  <button type="submit">Submit</button>
</form>
```

The form's action is set to POST the form to the controller's `/authenticate` endpoint. The `original_url` cookie will have been set by the webserver automatically, and will be sent by the browser as part of the POST request. If authentication is successful, the response from the controller will contain a `token` cookie, which the browser will store automatically.

2.1.2 Token Authentication

Token authentication is typically used by the HTTP API in cases where a web browser is not the client. The client requests a Bearer Token with a *POST* request to the controller's `/authenticate` endpoint, providing the username and password, and this token is then used in future requests.

To use the token in a request, set the `Authorization` header value to `Bearer {your token}`, where `{your token}` should be replaced with the value of `token` in the response.

The JSON object in the response from each authenticated request will include a `token` attribute, whose value will be a new token, valid for 5 minutes. If no authenticated requests are made for 5 minutes then the token will expire and the `/authenticate` endpoint must be used to get a new token.

WHAT'S NEW

3.1 v6.0

- Breaking change to HTTP authentication, using new *Authenticate* endpoint.
- Add *Factory Reset* HTTP endpoint.
- Remove `password` from the HTTP *config* response.
- Breaking change to setting colour overrides with new *Override Colour* object in *HTTP* and *JavaScript*.
- New snapshot functionality when setting colour overrides in *HTTP* and *JavaScript*.
- Add *RDM Discovery* HTTP endpoint and *RDM Discovery* JavaScript function.
- Add *RDM Get* HTTP endpoint and *RDM Get* JavaScript function.
- Add *RDM Set* HTTP endpoint and *RDM Set* JavaScript function.
- Add EDN protocols to Lua *disable_output*.

3.2 v5.0

- Added controller propagation to certain HTTP API requests and query.js functions.
- `memory_free` changed to `memory_available` in the HTTP & JavaScript *System* information and in the Lua *System* namespace.
- *get_trigger_number* function added.
- `vlan_tag` property added to Lua *Controller*.
- `is_network_primary` property added to Lua *Controller*.
- `dns_servers` property added to the Lua *System* namespace.

HTTP API

Mosaic controllers provide an HTTP API to query and control the current project and the controller itself.

4.1 Authentication

Mosaic controllers have user accounts, each of which can belong to different security groups, which in turn control access to parts of the HTTP API. The HTTP API has a series of *endpoints* to allow clients to authenticate users with the controller.

4.1.1 Authentication

Authentication reference for the controller HTTP API.

Authenticate

Methods

POST

Accepts form data or JSON to authenticate a user's credentials.

POST /authenticate

The payload, whether form data or JSON, should have the following attributes:

Attribute	Value Type	Description
username	string	The username of the user.
password	string	The user's password.

If the credentials are valid, a JSON web token (JWT) is returned. This token is returned either as a `token` cookie or in a JSON object with a `token` attribute, depending on whether the *original_url* parameter was sent with the request.

To use a token returned in a JSON object to authorise a request, set the `Authorization` header value to `Bearer {your token}`, where `{your token}` should be replaced with the value of `token` in the response from /authenticate.

If the user cannot be authenticated because the username or password are incorrect then a redirect response will be returned, pointing to the value of the `Referer` header in the request.

The response will be a 400 error if either attribute is missing or a value is of an invalid type.

original_url

The `original_url` parameter is typically used when authenticating the user from form data sent from a web page. Its value is set to the path of the page from where the user was redirected to the login page, and its where the response from `/authenticate` will redirect the browser upon successful authentication. It can be sent as a cookie or a query parameter with the `/authenticate` request. Its presence in the request will result in the response from `/authenticate` setting a cookie with the JWT, rather than returning a JSON object containing the JWT.

For example, if an unauthenticated or unauthorised user attempts to access the configuration page of the built-in web interface, they would try to navigate to `/default/config.lsp` but the controller's web server would redirect them to `default/login.lsp` and set the `original_url` cookie to `/default/config.lsp`.

In a custom web interface using `.webconfig` files to configure access control, the `original_url` cookie is automatically set by the web server when redirecting to the login page (which may be a custom login page) when the user attempts to access a restricted page for which they are not authorised.

In both cases, when the login page submits a request to `/authenticate`, the `original_url` cookie will be sent automatically by the browser. A successful response will redirect to the value of `original_url` and store a `token` cookie in the browser with the user's JWT.

Logout

Methods

GET

Ends the user's current session.

`GET /logout`

The request must be authenticated either with a cookie or by sending a valid Bearer token in the `Authorization` header.

If the request is made from a web browser using cookie authentication then the cookie will be deleted from the browser by the response. The web browser will reload the page from which the request was made if the `Referer` header is set.

4.2 Querying and Controlling

The endpoints provided in the HTTP API for querying and controlling the controller and its current project are detailed in the following sections:

4.2.1 Beacon

Methods

POST

Toggle beacon mode on the controller.

`POST /api/beacon`

In beacon mode, a controller will flash its LEDs or it screen continuously.

4.2.2 Channel / Park

Methods

POST

Park an output channel or channels at a specified level.

`POST /api/channel`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`universe`	string	See *Universe Key String Format*	`"dmx:1"`
`channels`	string	Comma separated list of channel numbers.	`"1-3,5"`
`level`	integer	Level to set the channel(s) to: 0-255.	`128`

DELETE

Unpark an output channel or channels.

`DELETE /api/channel`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`universe`	string	See *Universe Key String Format*	`"dmx:1"`
`channels`	string	Comma separated list of channel numbers.	`"1-3,5"`

Universe Key String Format

A universe key string takes the form:

- `protocol:index` for protocols `dmx`, `pathport`, `sacn`, `art-net`;
- `protocol:kinetPowerSupplyNum:kinetPort` for protocol `kinet`;
- `protocol:remoteDeviceType:remoteDeviceNum` for protocol `rio-dmx`;
- `protocol:remoteDeviceType:remoteDeviceNum:port` for protocols `edn`, `edn-spi`.

Where:

- `kinetPowerSupplyNum` is an integer;
- `kinetPort` is an integer;
- `remoteDeviceType` can be `rio08`, `rio44` or `rio80`, `edn10` or `edn20`;
- `remoteDeviceNum` is an integer;
- `port` is an integer.

For example:

- `"dmx:1"`
- `"rio-dmx:rio44:1"`

4.2.3 Cloud

Methods

GET

Returns the state of connectivity to the cloud remote management system.

`GET /api/cloud`

Returns a JSON object with the following attributes:

Attribute	Value Type	Description
`connected`	boolean	Whether or not the system is currently connected to the cloud
`connecting`	boolean	Whether or not the system is currently in the process of connecting to the cloud

POST

Allows configuration of the parameters for connection to the cloud.

`POST /api/cloud`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description
`action`	string	Either `set_device_key` or `clear_device_key`
`cloud_device_key`	string	Only required for `set_device_key` - the string to set as the key.

4.2.4 Command

Methods

POST

Run a Lua script or pass a command to the command line parser on the controller.

Note: The Command Line Parser must be enabled in the web interface settings of the current project, else this endpoint will not be available.

`POST /api/cmdline`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description
`input`	string	The script to parse or run.

For example:

```
{
  "input": "tl = 1 get_timeline(tl):start()"
}
```

Response

If the Command Line Parser is enabled in the web interface settings of the current project then a 200 status code will be returned, along with the text `Executed` if the script was executed successfully. If an error occurred when attempting to run the script then the error string will be returned.

The response will be `501 Not Implemented` if the Command Line Parser is not enabled, or `400 Bad Request` if no project is loaded.

4.2.5 Config

Methods

POST

Edits the configuration of the controller.

`POST /api/config`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`ip`	string	Optional. Set the controller's IP address (management interface)	`"192.168.1.3"`
`subnet_mask`	string	Optional. Set the controller's subnet mask (management interface)	`"255.255.255.0"`
`gateway`	string	Optional. Set the controller's gateway address (management interface)	`"192.168.1.1"`
`dhcp_enabled`	boolean	Optional. Set whether the controller is assigned its IP address automatically by DHCP	`true`
`name_server_1`	string	Optional. Set the primary name server address	`"192.168.1.1"`
`name_server_2`	string	Optional. Set the secondary name server address	`"8.8.8.8"`
`http_port`	integer	Optional. Set the port opened for HTTP access to the controller's web server	`80`
`https_port`	integer	Optional. Set the port opened for HTTPS access to the controller's web server	`443`
`year`	integer	Optional. Set the year of the current date on the controller's clock	`2021`
`month`	integer	Optional. Set the month of the current date on the controller's clock (1-12)	`4`
`day`	integer	Optional. Set the day of the current date on the controller's clock (1-31)	`25`
`hour`	integer	Optional. Set the hour component of the current time on the controller's clock (0-23)	`13`
`minute`	integer	Optional. Set the minute component of the current time on the controller's clock (0-59)	`21`
`second`	integer	Optional. Set the second component of the current time on the controller's clock (0-59)	`46`
`watchdog_enabled`	boolean	Optional. Set whether the controller's hardware watchdog is enabled	`true`
`log_level`	integer	Optional. Set the level of verbosity of the controller's log (1-5)	`3`
`syslog_enabled`	boolean	Optional. Set whether the controller will send its log to a syslog server	`false`
`syslog_ip`	string	Optional. Set the IP address of a third party syslog server	`"192.168.1.2"`
`ntp_enabled`	boolean	Optional. Set whether the controller will fetch the current time automatically from an NTP server	`true`
`ntp_ip`	string	Optional. Set the IP address of a third party NTP server	`"192.168.1.1"`

If the response status code is 200 (OK), the response body will be a JSON object with a `restart` attribute. The value of `restart` is boolean. If `true`, the controller will reset itself imminently in order to apply the changes.

GET

Returns information about the queried controller's configuration.

`GET /api/config`

Returns a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`ip`	string	Controller IP address (management interface)	`"192.168.1.3"`
`subnet_mask`	string	Controller subnet mask (management interface)	`"255.255.255.0"`
`gateway`	string	Gateway address (management interface)	`"192.168.1.1"`
`dhcp_enabled`	boolean	Whether the controller is assigned its IP address automatically by DHCP	`true`
`name_server_1`	string	Primary name server address	`"192.168.1.1"`
`name_server_2`	string	Secondary name server address	`"8.8.8.8"`
`http_port`	integer	Port opened for HTTP access to the controller's web server	`80`
`https_port`	integer	Port opened for HTTPS access to the controller's web server	`443`
`year`	integer	Year of the current date, according to the controller's clock	`2021`
`month`	integer	Month of the current date, according to the controller's clock (1-12)	`4`
`day`	integer	Day of the current date, according to the controller's clock (1-31)	`25`
`hour`	integer	Hour component of the current time, according to the controller's clock (0-23)	`13`
`minute`	integer	Minute component of the current time, according to the controller's clock (0-59)	`21`
`second`	integer	Second component of the current time, according to the controller's clock (0-59)	`46`
`watchdog_enabled`	boolean	Whether the controller's hardware watchdog is enabled	`true`
`log_level`	integer	Level of verbosity of the controller's log (1-5)	`3`
`syslog_enabled`	boolean	Whether the controller is sending its log to a syslog server	`false`
`syslog_ip`	string	IP address of a third party syslog server	`"192.168.1.2"`
`ntp_enabled`	boolean	Whether the controller is fetching current time automatically from an NTP server	`true`
`ntp_ip`	string	IP address of a third party NTP server	`"192.168.1.1"`

4.2.6 Content Targets

Note: Atlas/Atlas Pro only

Methods

POST

Control a content target; currently the only supported action is to master the intensity of a content target (applied as a multiplier to output levels).

```
POST /api/content_target
```

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`action`	string	The action to perform on the content target. Currently only `master_intensity` is supported.	`"master_intensity"`
`type`	string	Optional. Type of content target (only relevant on Atlas Pro): `primary`, `secondary`, `target_3`, `target_4`, `target_5`, `target_6`, `target_7`, `target_8`. Defaults to `primary`.	`"secondary"`
`level`	float or string containing a bounded integer	Master intensity level to set on the content target	`0.5 or "50:100"`
`fade`	float	Optional. Fade time to apply the intensity change, in seconds.	`2.0`
`delay`	float	Optional. Time to wait before applying the intensity change, in seconds.	`2.0`

GET

Returns information about the current state of all Content Targets in the project.

```
GET /api/content_target
```

Returns a JSON object with a single `content_targets` attribute, which has an array value. Each item in the array is a Content Target object with the following attributes:

Attribute	Value Type	Description	Value Example
`name`	string	Content target name	`"Primary"`
`level`	integer	Current intensity master level of the content target, 0-100	`100`

4.2.7 Controller

Methods

GET

Returns data about the controllers in the project.

`GET /api/controller`

Returns a JSON object with a single `controllers` attribute, which has an array value. Each item in the array is a Controller object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	integer	Controller number	1
`type`	string	Controller type, e.g. "MSC" or "MTPC"	"MSC"
`name`	string	Controller user name, or the default name if none is set	`"Controller 1"`
`serial`	string	Serial number of the controller	`"009060"`
`ip_address`	string	IP address of the controller if the controller is discovered; empty if the controller is not discovered or is the queried controller	`"192.168.1.3"` or `""`
`online`	boolean	Whether the controller is detected as online on the local network	`true`
`is_network_primary`	boolean	Whether the controller is set as the network primary in the project	`true`

4.2.8 DALI

If the project uses DALI, the DALI API call can be used to get the status of connected DALI ballasts, and to allow external systems to mark DALI issues as fixed.

Methods

GET

Returns information about connected DALI devices on a particular interface - see *DALI Interface* to retrieve a list of interfaces.

`GET /api/dali?interface=interface_num`

`interface_num` is an integer referring to a specific interface.

Returns a JSON object with the following attributes:

Attribute	Value Type	Description
`online`	boolean	Whether or not the interface is currently online
`schedule`	object	A *DALI Schedule* object
`power`	object	A *DALI Power* object
`errors`	array of objects	An array of *DALI Error* objects
`ballast_status`	array of objects	An array of *DALI Ballast Status* objects

POST

Allows marking of a DALI error as fixed, or refresh of the DALI data.

`POST /api/dali`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description
`interface`	integer	The interface on which to perform the reset.
`address`	integer	The DALI short address on which to perform the reset.
`action`	string	Either `mark_fixed` or `refresh`.

4.2.9 DALI Interface

The DALI Interface API allows retrieval of a list of DALI interfaces in the system.

Methods

GET

Returns an array of DALI interfaces

`GET /api/dali_interfaces`

Returns an array of JSON objects with the following attributes:

Attribute	Value Type	Description
`id`	integer	The ID of the interface
`name`	string	The assigned string name of the interface

4.2.10 Factory Reset

Reset the controller to its factory settings, including all network settings and user accounts.

HTTP

POST

`POST /api/factory_reset`

4.2.11 Group

Note: Not applicable to Atlas/Atlas Pro

Methods

POST

Control a group; currently the only supported action is to master the intensity of a group (applied as a multiplier to output levels). Action will propagate to all controllers in a project.

`POST /api/group`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`action`	string	The action to perform on the group. Currently only `master_intensity` is supported.	`"master_intensity"`
`num`	integer	Group number. Group 0 means the *All Fixtures* group.	`1`
`level`	float or string containing a bounded integer	Master level to set on the group	`0.5` or `"50:100"`
`fade`	float	Optional. Fade time to apply the intensity change, in seconds.	`2.0`
`delay`	float	Optional. Time to wait before applying the intensity change, in seconds.	`2.0`

GET

Returns data about the fixture groups in the project.

`GET /api/group[?num=groupNumbers]`

`num` can be used to filter which groups are returned and is expected to be either a single number or a string expressing the required groups, e.g. `"1,2,5-9"`.

Note: Group 0 will return data about the *All Fixtures* group.

Returns a JSON object with a single `groups` attribute, which has an array value. Each item in the array is a Group object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	integer	Group number (only included for user-created groups)	`1`
`name`	string	Group name	`"Group 1"`
`level`	integer	Group master level, 0-100	`100`

4.2.12 Input

Methods

GET

Returns the status of digital & analogue inputs on the queried controller.

`GET /api/input`

Returns a JSON object with the following attributes:

Attribute	Value Type	Description
gpio	array	Array of Input objects; returned when queried controller is MSC or MTPC + TPC-RIO
dmxIn	object	DMX Input object; returned when DMX input is configured on the queried controller

The Input object has the following properties:

Attribute	Value Type	Description	Value Example
input	integer	Input number	1
type	string	`Analog`, `Digital`, or `Contact Closure`	`"Contact Closure"`
value	integer or boolean	Value type depends on input type - `Analog` inputs return an integer, 0-100; other types return a bool.	`true`

The DMX Input object has the following properties:

Attribute	Value Type	Description	Value Example
error	string	If DMX input is configured but no DMX is received	`"No DMX received"`
dmxInFrame	array	Array of channel values	`[0,0,0,0,0,0,0,0, 0,255,255,255... 255,0,255]`
dmxInSourceCount	integer	The number of sources - will be 1 except for sACN.	1
dmxInProtocol	string	`dmx`, `art-net` or `sacn`	`"dmx"`

4.2.13 Log

Methods

GET

Returns the log from the controller.

`GET /api/log`

Returns a JSON object with the following attributes:

Attribute	Value Type	Description
`log`	string	The whole log from the controller

4.2.14 Lua Variable

Methods

GET

Returns the current value of specified Lua variables.

`GET /api/lua?variables=luaVariables`

`luaVariables` is expected to be a string or comma-separated list of strings, where each string is a Lua variable name.

Returns a JSON object with the Lua variables and their values as its key/value pairs - the Lua variable names are the keys.

For example, in a project that creates variables called `bob` and `alice`, `GET /api/lua?variables=bob,alice` could return a JSON object as follows:

```
{
  "alice": 1234,
  "bob": "a string variable"
}
```

4.2.15 Output

Methods

POST

Enable/disable the output of a selected protocol from the controller. Action will propagate to all controllers in a project.

`POST /api/output`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`protocol`	string	Protocol to disable. Options: `dmx`, `pathport`, `sacn`, `art-net`, `kinet`, `rio-dmx`, `edn`, `edn-spi`.	`"parthport"`
`action`	string	Whether to `enable` or `disable` output via the protocol.	`"disable"`

GET

Returns the lighting levels being output by the queried controller.

`GET /api/output?universe=universeKey`

`universeKey` is a string; see *Universe Key String Format*.

For example: * `GET /api/output?universe=dmx:1` * `GET /api/output?universe=rio-dmx:rio44:1`

If the queried controller is an MSC 1, the universe is DMX 2, DMX Proxy has been enabled for a MTPC in the project and the MTPC is offline then this request will return a JSON object with the following attributes:

Attribute	Value Type	Value Example
`proxied_tpc_name`	string	`"Controller 2""`

Otherwise a JSON object with the following attributes is returned:

Attribute	Value Type	Description	Value Example
`channels`	array	Array of integer (0-255) channel levels	`[0,0,0,0,0,0,0,0,` `0,255,255,255...` `255,0,255]`
`disabled`	bool	Whether the output has been disabled by a Trigger Action	`false`

Universe Key String Format

A universe key string takes the form:

- `protocol:index` for protocols `dmx`, `pathport`, `sacn`, `art-net`;

- `protocol:kinetPowerSupplyNum:kinetPort` for protocol `kinet`;

- `protocol:remoteDeviceType:remoteDeviceNum` for protocol `rio-dmx`;

- `protocol:remoteDeviceType:remoteDeviceNum:port` for protocols `edn`, `edn-spi`.

Where:

- `kinetPowerSupplyNum` is an integer;

- `kinetPort` is an integer;

- `remoteDeviceType` can be `rio08`, `rio44` or `rio80`, `edn10` or `edn20`;

- `remoteDeviceNum` is an integer;

- `port` is an integer.

For example:

- `"dmx:1"`

- `"rio-dmx:rio44:1"`

4.2.16 Override

Methods

PUT

Set the Intensity, Red, Green, Blue levels for a fixture or group. Action will propagate to all controllers in a project.

`PUT /api/override`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`target`	string	What the override should be applied to: `group`, `fixture`.	`"group"`
`num`	integer	Group or fixture number, depending on `target`. Group 0 means the *All Fixtures* group.	`1`
`intensity`	integer or string	Optional. Either an integer (0-255) representing the intensity to set as part of override **or** the string `"snapshot"` to capture the current intensity of the fixture(s) and set this as the override value. Intensity override will not be changed if this attribute isn't provided.	`128`
`colour`	*Override Colour* or string	Optional. Specifies the colour to set as part of the override. Either an *Override Colour* or the string `"snapshot"` to capture the current colour of the fixture(s) and set this as the override.	
`temperature`	integer or string	Optional. Either an integer (0-255) representing the temperature component to set as part of override **or** the string `"snapshot"` to capture the current temperature component of the fixture(s) and set this as the override value. Temperature override will not be changed if this attribute isn't provided.	`128`
`fade`	float	Optional. Fade time to apply the override change, in seconds.	`2.0`
`path`	string	Optional. Crossfade path to use when applying the override: `Default`, `Linear`, `Start`, `End`, `Braked`, `Accelerated`, `Damped`, `Overshoot`, `Col At Start`, `Col At End`, `Int At Start`, `Int At End`, `Colour First`, `Intensity First`	`"Braked"`

Override Colour

The value of the `colour` attribute in a PUT override request is a JSON object, specifying colour as *either RGB* or *Hue/Saturation* values.

RGB

Colour as RGB for `colour` in an override *PUT* request:

Attribute	Value Type	Description	Value Example
red	integer	Optional. Red component to set as part of override: 0-255. Red override will not be changed if this attribute isn't provided.	255
green	integer	Optional. Green component to set as part of override: 0-255. Green override will not be changed if this attribute isn't provided.	255
blue	integer	Optional. Blue component to set as part of override: 0-255. Blue override will not be changed if this attribute isn't provided.	255

Hue/Saturation

Colour as hue/saturation for `colour` in an override *PUT* request:

Attribute	Value Type	Description	Value Example
hue	integer	Hue component to set as part of override: 0-255.	0
saturation	integer	Saturation component to set as part of override: 0-255.	255

Note: Both `hue` and `saturation` are required for the request to be valid.

DELETE

Release any overrides on fixtures or groups. Action will propagate to all controllers in a project.

`DELETE /api/override`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
target	string	What the overrides should be cleared on: `group`, `fixture`.	`"group"`
num	integer	Optional. Group or fixture number, depending on `target`. If not provided, `target` is ignored and all overrides are cleared.	1
fade	float	Optional. Fade time in which to release overrides, in seconds.	`2.0`

4.2.17 Project

Methods

GET

Returns data about the current project.

```
GET /api/project
```

Returns a JSON object with the following attributes:

Attribute	Value Type	Value Example
`name`	string	`"Help Project"`
`author`	string	`"Contoso"`
`filename`	string	`"help_project_v1.pd2"`
`unique_id`	string	`"{6b48627a-1d5e-4b2f-81e2-481e092a6a79}"`
`upload_date`	string	`"2017-01-30T15:19:08"`

4.2.18 Project File

The controller allows you to upload or download the current project file, allowing the project in use to be switched out.

Methods

GET

Downloads the currently running project file.

```
GET /api/project/file
```

Returns the project file (as type application/vnd.pharos).

POST

Uploads a project file, which will trigger the controller to switch to the new file.

> **Warning:** The file to be uploaded **must** be exported from Designer for the project using the *Export Project For Upload* button in Designer under the *Network* tab. You can **not** load a saved Designer project file directly.

```
POST /api/project/file
```

Uploads a project file to the controller. The body of the request should be the exported project file as binary data.

Note that the `Content-Type` header should be set to `application/vnd.pharos`; and the `Content-Length` header should be set to the size of the project file.

4.2.19 Protocol

Methods

GET

Returns all the universes in the project on the queried controller.

`GET /api/protocol`

Returns a JSON object with a single `outputs` attribute, which has an array value. Each item in the array is a Protocol object with the following attributes:

Attribute	Value Type	Description	Value Example
`type`	integer	Protocol type; possible types are: DMX (1), Path-port (2), Art-Net (4), KiNET (8), sACN (16), DVI (32), RIO DMX (64), EDN DMX (128), EDN SPI (256)	1
`name`	string	Protocol name	`"DMX"`
`disabled`	boolean	Whether the output has been disabled by a Trigger Action	`false`
`universes`	array	Array of Universe objects (see table below)	`[{"key":{"index":1}, "name":"1"}, {"key":{"index":2}, "name":"2"}]`
`dmx_proxy`	object	DMX Proxy object, if applicable (see table below)	`{"ip_address":"192. 168.1.17", "name":"Controller 1"}`

Each Universe object has the following properties:

Attribute	Value Type	Description	Value Example
`name`	string	A simplistic version of the universe name, which for most protocols is simply the index number	`"1"`
`key`	object	Universe Key object (see table below)	`{"index":1}`

Each DMX Proxy object has the following properties:

Attribute	Value Type	Description	Value Example
`name`	string	Name of the controller that is outputting this universe by proxy	`"Controller 1"`
`ip_address`	string	IP address of the controller that is outputting this universe by proxy	`"192.168.1.17"`

The properties of the Universe Key object depend on the type.

For DMX, Pathport, sACN and Art-Net:

Attribute	Value Type	Value Example
`index`	integer	1

For KiNET:

Attribute	Value Type	Value Example
`kinet_port`	integer	1
`kinet_power_supply_num`	integer	1

For RIO DMX:

Attribute	Value Type	Description	Value Example
`remote_device_num`	integer	Remote device number (address)	1
`remote_device_type`	integer	Value can be 101 (RIO 80), 102 (RIO 44) or 103 (RIO 08)	101

For EDN:

Attribute	Value Type	Description	Value Example
`remote_device_num`	integer	EDN number (address)	1
`remote_device_type`	integer	Value can be 109 (EDN 20) or 110 (EDN 10)	110
`port`	integer	Number of EDN output port	1

4.2.20 RDM Discovery

Methods

POST

Request to start a full RDM discovery. A `202` response will be returned if the request has been successfully queued. Results are available via a websocket subscription (see *subscribe_rdm_discovery*).

`POST /api/rdm/discovery`

Payload is a JSON object with a single `universe` attribute, which can either be a string in the *Universe Key String Format* or an *RDM Universe Key* object.

For example, to start a full discovery on DMX universe 2, the request payload could be:

```
{
  "universe": "dmx:2"
}
```

or, alternatively:

```
{
  "universe": {
    "protocol": 1,
    "index": 2
  }
}
```

To start RDM discovery on the first port of the EDN 20 with number 4 in the project, the request payload could be:

```
{
  "universe": "edn:edn20:4:1"
}
```

or, alternatively:

```
{
  "universe": {
    "protocol": 128,
    "remote_device_type": 109
  }
}
```

PUT

Request to start an RDM discovery update, which is faster if a full RDM discovery has already been performed with a *POST* request. A `202` response will be returned if the request has been successfully queued. Results are available via a websocket subscription (see *subscribe_rdm_discovery*).

`PUT /api/rdm/discovery`

Payload is a JSON object with a single `universe` attribute, which can either be a string in the *Universe Key String Format* or an object with the same attributes as for the *POST* request.

GET

Returns the cached results of the last RDM discovery operation.

`GET /api/rdm/discovery?universe=universeId`

`universe` specifies which output universe to fetch cached RDM discovery data for. Its value is a string in the *Universe Key String Format*.

Returns a JSON object with a `devices` attribute, which has an array value. Each item in the array is an *RDM Device Info* object.

Universe Key String Format

A universe key string for RDM takes the form:

- `protocol:index` for protocols `dmx` and `art-net`;
- `protocol:remoteDeviceType:remoteDeviceNum:port` for protocol `edn`.

Where:

- `remoteDeviceType` can be `edn10` or `edn20`;
- `remoteDeviceNum` is an integer;
- `port` is an integer.

For example:

- `"dmx:1"`
- `"edn:edn20:1:5"`

4.2.21 RDM Get

Methods

POST

Request to start an RDM Get operation. A `202` response will be returned if the request has been successfully queued. Results are available via a websocket subscription (see *subscribe_rdm_get_set*).

`POST /api/rdm/get`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`universe`	string in *Universe Key String Format* or *RDM Universe Key*	The universe on which to perform the RDM Get operation.	`"dmx:2"`
`destination_uid`	string	Format is `{manuId}:{deviceId}(:{subId})` where {manuId} is a padded unsigned hexadecimal integer of width 4, lowercase, e.g. `072c`; {deviceId} is a padded unsigned hexadecimal integer of width 8, lowercase, e.g. `0004fe02`; {subId} is an optional unsigned decimal integer.	`"072c:0004fe02"`
`pid`	string	RDM PID for the Get operation. Can be one of the *Supported RDM PIDs* or the raw PID value as a hex string, e.g. `"FF"`.	`"DEVICE_INFO"`
`meta`	object	Optional. Metadata for the PID, i.e. query params (see *Meta*).	
`max_rx_length`	integer	Optional. Expected length of the response data. Only relevant if a raw PID value has been provided for `pid`. If not provided then the controller must wait for a timeout before handling a response to ensure all response data has been received from the device.	

Meta

STATUS_MESSAGES

For the STATUS_MESSAGES PID, the `meta` object should have the following parameters:

Attribute	Value Type	Description
status_type	integer	Type of status messages to retrieve. Set to STATUS_NONE (`0x00`) to establish whether a device is present on the network without retrieving any status message data from the device.

PARAMETER_DESCRIPTION

For the PARAMETER_DESCRIPTION PID, the `meta` object should have the following parameters:

Attribute	Value Type	Description
pid_requested	integer	The manufacturer-specific PID for which a description is requested. Range 0x8000 to 0xFFDF.

DMX_PERSONALITY_DESCRIPTION

For the DMX_PERSONALITY_DESCRIPTION PID, the `meta` object should have the following parameters:

Attribute	Value Type	Description
personality_requested	integer	Index of the requested personality.

SLOT_DESCRIPTION

For the SLOT_DESCRIPTION PID, the `meta` object should have the following parameters:

Attribute	Value Type
slot_number_requested	integer

SENSOR_DEFINITION and SENSOR_VALUE

For the SENSOR_DEFINITION and SENSOR_VALUE PIDs, the `meta` object should have the following parameters:

Attribute	Value Type
sensor_number_requested	integer

Universe Key String Format

A universe key string for RDM takes the form:

- `protocol:index` for protocols `dmx` and `art-net`;
- `protocol:remoteDeviceType:remoteDeviceNum:port` for protocol `edn`.

Where:

- `remoteDeviceType` can be `edn10` or `edn20`;
- `remoteDeviceNum` is an integer;
- `port` is an integer.

For example:

- `"dmx:1"`
- `"edn:edn20:1:5"`

Supported RDM PIDs

The following PIDs are directly supported for RDM Get operations:

- `COMMS_STATUS`
- `STATUS_MESSAGES`
- `SUPPORTED_PARAMETERS`
- `PARAMETER_DESCRIPTION`
- `DEVICE_INFO`
- `DEVICE_MODEL_DESCRIPTION`
- `MANUFACTURER_LABEL`
- `DEVICE_LABEL`
- `FACTORY_DEFAULTS`
- `SOFTWARE_VERSION_LABEL`
- `BOOT_SOFTWARE_VERSION_ID`
- `BOOT_SOFTWARE_VERSION_LABEL`
- `DMX_PERSONALITY`
- `DMX_PERSONALITY_DESCRIPTION`
- `DMX_START_ADDRESS`
- `SLOT_INFO`
- `SLOT_DESCRIPTION`
- `SENSOR_DEFINITION`
- `SENSOR_VALUE`
- `LAMP_HOURS`
- `LAMP_STATE`

4.2.22 RDM Set

Methods

POST

Request to start an RDM Set operation. A `202` response will be returned if the request has been successfully queued. Results are available via a websocket subscription (see *subscribe_rdm_get_set*).

`POST /api/rdm/set`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`universe`	string in *Universe Key String Format* or *RDM Universe Key*	The universe on which to perform the RDM Set operation.	`"dmx:2"`
`destination_uid`	string	Format is `{manuId}:{deviceId}(:{subId})` where `{manuId}` is a padded unsigned hexadecimal integer of width 4, lowercase, e.g. `072c`; `{deviceId}` is a padded unsigned hexadecimal integer of width 8, lowercase, e.g. `0004fe02`; `{subId}` is an optional unsigned decimal integer.	`"072c:0004fe02"`
`pid`	string	RDM PID for the Set operation. Can be one of the *Supported RDM PIDs* or the raw PID value as a hex string, e.g. `"FF"`.	`"DEVICE_INFO"`
`meta`	object	Optional. Metadata for the PID, i.e. query params (see *Meta*).	
`max_rx_length`	integer	Optional. Expected length of the response data. Only relevant if a raw PID value has been provided for `pid`. If not provided then the controller must wait for a timeout before handling a response to ensure all response data has been received from the device.	

Meta

DEVICE_LABEL

For the `DEVICE_LABEL` PID, the `meta` object should have the following parameters:

Attribute	Value Type	Description
`label`	string	Ascii text label for the device. Up to 32 characters.

IDENTIFY_DEVICE

For the `IDENTIFY_DEVICE` PID, the `meta` object should have the following parameters:

Attribute	Value Type	Description
enable	boolean	Whether to enable/disable IDENTIFY_DEVICE mode over RDM.

DMX_START_ADDRESS

For the `DMX_START_ADDRESS` PID, the `meta` object should have the following parameters:

Attribute	Value Type	Description
start_address	integer	DMX start address to set on the device.

DMX_PERSONALITY

For the `DMX_PERSONALITY` PID, the `meta` object should have the following parameters:

Attribute	Value Type	Description
personality	integer	Index of the personality to set as current.

SENSOR_VALUE

For the `SENSOR_VALUE` PID, the `meta` object should have the following parameters:

Attribute	Value Type	Description
sensor_number	integer	Sensor number to reset.

LAMP_HOURS

For the `LAMP_HOURS` PID, the `meta` object should have the following parameters:

Attribute	Value Type	Description
lamp_hours	integer	Starting value to set on the device's lamp hours counter.

LAMP_STATE

For the `LAMP_STATE` PID, the `meta` object should have the following parameters:

Attribute	Value Type	Description
lamp_state	integer	Operating state to set the lamp to.

Raw

Where a raw PID value has been provided for `pid`, the `meta` object should have a single `raw` attribute with a string value. This value will be the base64-encoded string containing parameters for the Set command.

Universe Key String Format

A universe key string for RDM takes the form:

- `protocol:index` for protocols `dmx` and `art-net`;

- `protocol:remoteDeviceType:remoteDeviceNum:port` for protocol `edn`.

Where:

- `remoteDeviceType` can be `edn10` or `edn20`;

- `remoteDeviceNum` is an integer;

- `port` is an integer.

For example:

- `"dmx:1"`

- `"edn:edn20:1:5"`

Supported RDM PIDs

The following PIDs are directly supported for RDM Set operations:

- `COMMS_STATUS`

- `DEVICE_LABEL`

- `FACTORY_DEFAULTS`

- `IDENTIFY_DEVICE`

- `DMX_START_ADDRESS`

- `DMX_PERSONALITY`

- `SENSOR_VALUE`

- `LAMP_HOURS`

- `LAMP_STATE`

4.2.23 Remote Device

Methods

GET

Returns data about all the remote devices in the project.

`GET /api/remote_device`

Returns a JSON object with a single `remote_devices` attribute, which has an array value. Each item in the array is a Remote Device object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	integer	Remote device number (address)	1
`type`	string	`RIO 08`, `RIO 44`, `RIO 80`, `BPS`, `BPI`, `RIO A`, or `RIO D`	`"RIO 44"`
`serial`	array	Array of serial numbers (as strings) of all discovered devices matching the address and type	`["001234", "005678"]`
`outputs`	array	Array of Output objects (see table below); only returned for RIO 44 and RIO 08 on the queried controller	`[{"output":1, "value":true}, {"output":2, "value":true}, {"output":3, "value":true}, {"output":4, "value":true}]`
`inputs`	array	Array of Input objects (see table below); only returned for RIO 44 and RIO 80 on the queried controller	`[{"input":1, "type":"Contact Closure", "value":true}, {"input":2, "type":"Contact Closure", "value":true}, {"input":3, "type":"Contact Closure", "value":true}, {"input":4, "type":"Contact Closure", "value":true}]`
`online`	boolean	Whether the remote device is detected as being online on the local network	`true`

The Output JSON object has the following attributes:

Attribute	Value Type	Description	Value Example
`output`	integer	Number of the output, as labelled on the remote device	1
`state`	boolean	`true` means the output is on, `false` means it is off	`true`

The Input JSON object has the following attributes:

Attribute	Value Type	Description	Value Example
`input`	integer	Number of the input, as labelled on the remote device	1
`type`	string	`Analog`, `Digital`, or `Contact Closure`	`""Digital"`
`value`	integer or boolean	Value type depends on input type - `Analog` inputs return an integer, 0-255; other types return a bool.	`true`

4.2.24 Replication

Methods

GET

Returns data about the install replication.

`GET /api/replication`

Returns a JSON object with the following attributes:

Attribute	Value Type	Value Example
`name`	string	`"Help Project"`
`unique_id`	string	`"{6b48627a-1d5e-4b2f-81e2-481e092a6a79}"`

4.2.25 Hardware Reset

Methods

POST

Reboot the controller.

`POST /api/reset`

4.2.26 Scene

Methods

POST

Control a scene in the project.

Action will propagate to all controllers in a project.

`POST /api/scene`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`action`	string	The action to perform on the scene(s): `start`, `release`, `toggle`	`"start"`
`num`	integer	The number of the scene to perform the action on. If not present, the action will be applied to all scenes in the project; omitting this attribute is valid for `release`.	`1`
`fade`	number	Optional. The fade time to apply to a `release` action, in seconds, or the scene release that results from a `toggle` action. If not provided, the default release fade time will be used.	`2.0`
`group`	string	Optional. Scene group name: A through H. Prepend the group name with ! to apply the action to all groups *except* the specified group, e.g. `!A`. This attribute is valid for a `release` action without a specified `num`, meaning *release all scenes*.	`"B"`

For example, to start a scene 2, the request payload is:

```
{
  "action": "start",
  "num": 2
}
```

To release scene 2 in 3.5 seconds, the request payload would be:

```
{
  "action": "release",
  "num": 2,
  "fade": 3.5
}
```

To toggle scene 2, and release it in 2 seconds if it's already been started, the request payload would be:

```
{
  "action": "toggle",
  "num": 2,
  "fade": 2.0
}
```

To release all scenes in 2 seconds, the request payload would be:

```
{
  "action": "release",
  "fade": 2.0
}
```

To release all scenes except those in group B in 2 seconds, the request payload would be:

```
{
  "action": "release",
  "group": "!B",
  "fade": 2.0
}
```

GET

Returns data about the scenes in the project and their state on the controller.

`GET /api/scene[?num=sceneNumbers]`

`num` can be used to filter which scenes are returned and is expected to be either a single number or a string expressing the required scenes, e.g. `"1,2,5-9"`.

Returns a JSON object with a single `scenes` attribute, which has an array value. Each item in the array is a Scene object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	integer	Scene number	1
`name`	string	Scene name	`"Scene 1"`
`state`	string	`none`, `started`	`"none"`
`onstage`	boolean	Whether the scene is affecting output of any fixtures	`true`

4.2.27 System

Methods

GET

Returns data about the controller.

`GET /api/system`

Returns a JSON object with the following attributes:

Attribute	Value Type	Value Example
`hardware_type`	string	"MSC"
`channel_capacity`	integer	512
`serial_number`	string	`"006321"`
`memory_total`	string	`"12790Kb"`
`memory_used`	string	`"24056Kb"`
`memory_available`	string	`"103884Kb"`
`storage_size`	string	`"1914MB"`
`bootloader_version`	string	`"0.9.0"`
`firmware_version`	string	`"2.8.0"`
`reset_reason`	string	`"Software Reset"`
`last_boot_time`	string	`"01 Jan 2017 09:09:38"`
`ip_address`	string	`"192.168.1.3"`
`subnet_mask`	string	`"255.255.255.0"`
`broadcast_address`	string	`"192.168.1.255"`
`default_gateway`	string	`"192.168.1.3"`

4.2.28 Temperature

Methods

GET

Returns data about the controller's temperature.

`GET /api/temperature`

Returns a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`sys_temp`	number	Only for MSC X and Atlas/Atlas Pro	`40.2`
`core1_temp`	number	Only for MSC X and Atlas/Atlas Pro	`44`
`core2_temp`	number	Only for MSC X rev 1	`44.1`
`ambient_temp`	number	Only for MTPC, MSC X rev 1	`36.9`
`cc_temp`	number	Only for MSC X rev 2 and Atlas/Atlas Pro	`44.1`
`gpu_temp`	number	Only for Atlas/Atlas Pro	`38.2`

4.2.29 Text Slots

Methods

PUT

Set the value of a text slot used in the project, which will propagate to all controllers in a project.

`PUT /api/text_slot`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`name`	string	Text slot name	`"myTextSlot"`
`value`	string	New value for the text slot.	`"Hello World!"`

GET

Returns data about the text slots in the project and their current values.

`GET /api/text_slot[?names=slotNames]`

`slotNames` can be used to filter which test slots are returned and is expected to be either a single string or an array of strings.

Returns a JSON object with a single `text_slots` attribute, which has an array value. Each item in the array is a Text Slot object with the following attributes:

Attribute	Value Type	Value Example
`name`	string	`"text"`
`value`	string	`"example"`

4.2.30 Time

Methods

GET

Returns data about the time stored in the controller.

`GET /api/time`

Returns a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`datetime`	string	Controller's local time as a string	`"01 Feb 2017 13:44:42"`
`local_time`	integer	Controller's local time in milliseconds	`1485956682`
`uptime`	integer	Milliseconds since last boot	`493347`

4.2.31 Timeline

Methods

POST

Control a timeline in the project. Action will propagate to all controllers in a project.

`POST /api/timeline`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
action	string	The action to perform on the timeline(s): start, release, toggle, pause, resume, set_rate, set_position	"start"
num	integer	The number of the timeline to perform the action on. If not present, the action will be applied to all timelines in the project; omitting this attribute is valid for release, pause and resume.	1
fade	number	Optional. The fade time to apply to a release action, in seconds, or the timeline release that results from a toggle action. If not provided, the default release fade time will be used.	2.0
group	string	Optional. Timeline group name: A, B, C, D, E, F, G or H. Prepend the group name with ! to apply the action to all groups *except* the specified group, e.g. !A. This attribute is valid for a release action without a specified num, meaning *release all timelines*.	"B"
rate	string	Required for a set_rate action; invalid otherwise. Value should be a string containing a floating point number or a bounded integer, where 1.0 means the timeline's default rate.	"0.1" or "10:100"
position	string	Required for a set_position action; invalid otherwise. Value should be a string containing a floating point number or a bounded integer, representing a fraction of the timeline length.	"0.1" or "10:100"

For example, to start a timeline 2, the request payload is:

```
{
  "action": "start",
  "num": 2
}
```

To release timeline 2 in 3.5 seconds, the request payload would be:

```
{
  "action": "release",
  "num": 2,
  "fade": 3.5
}
```

To toggle timeline 2, and release it in 2 seconds if it's running, the request payload would be:

```
{
  "action": "toggle",
  "num": 2,
  "fade": 2.0
}
```

To pause timeline 4, the request payload is:

```
{
  "action": "pause",
```

```
  "num": 4
}
```

To resume timeline 4, the request payload is:

```
{
  "action": "resume",
  "num": 4
}
```

To pause all timelines, the request payload is:

```
{
  "action": "pause"
}
```

To resume all timelines, the request payload is:

```
{
  "action": "resume"
}
```

To release all timelines in 2 seconds, the request payload would be:

```
{
  "action": "release",
  "fade": 2.0
}
```

To release all timelines except those in group B in 2 seconds, the request payload would be:

```
{
  "action": "release",
  "group": "!B",
  "fade": 2.0
}
```

To set the rate of timeline 5 to half the default rage, the request payload would be:

```
{
  "action": "set_rate",
  "num": 5,
  "rate": "0.5"
}
```

To set the position of timeline 1 to a third of the way through, the request payload would be:

```
{
  "action": "set_rate",
  "num": 1,
  "position": "1:3"
}
```

GET

Returns data about the timelines in the project and their state on the controller.

`GET /api/timeline[?num=timelineNumbers]`

`num` can be used to filter which timelines are returned and is expected to be either a single number or a string expressing the required timelines, e.g. `"1,2,5-9"`.

Returns a JSON object with a single `timelines` attribute, which has an array value. Each item in the array is a Timeline object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	integer	Timeline number	1
`name`	string	Timeline name	`"Timeline 1"`
`group`	string	Timeline group name (A through H or empty string)	`"A"`
`length`	integer	Timeline length, in milliseconds	`10000`
`source_bus`	string	`internal`, `timecode_1` ... `timecode_6`, `audio_1` ... `audio_4`	`"internal"`
`timecode_format`	string	Incoming timecode format on source bus	`"SMPTE30"`
`audio_band`	integer	0 is volume band	`0`
`audio_channel`	string	`left`, `right` or `combined`	`"combined"`
`audio_peak`	boolean	The Peak setting of the timeline, if set to an audio time source	`false`
`time_offset`	integer	1/1000 of a second	`5000`
`state`	string	`none`, `running`, `paused`, `holding_at_end` or `released`	`"running"`
`onstage`	boolean	Whether the timeline is affecting output of any fixtures	`true`
`position`	integer	1/1000 of a second	`10000`
`priority`	string	`high`, `above_normal`, `normal`, `below_normal` or `low`	`"normal"`
`custom_properties`	object	Object properties and property values correspond to custom property names and values	`{}`

4.2.32 Trigger

Methods

POST

Fire a trigger in the project.

`POST /api/trigger`

Payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	User number of the trigger to fire.	2
var	string	Optional. Comma-separated to pass into the trigger.	e.g. a string "Foo"; integers 2,4,5; multiple strings '"string1", "string2", "string3"'
conditions	boolean	Optional. Whether to test the trigger's conditions before deciding to run its actions. Defaults to true.	true

GET

Returns the triggers in the project.

GET /api/trigger?[type=triggerType]

triggerType is expected to be a string and can be used to filter the type of trigger returned. For example, "Timeline Started" would return only Timeline Started triggers in the project.

Returns a JSON object with a single triggers attribute, which has an array value. Each item in the array is a Trigger object with the following attributes:

Attribute	Value Type	Description	Value Example
type	string	Trigger type	"Startup"
num	integer	Trigger user number	1
name	string	User-defined trigger name	"Initialise"
group	string	Trigger group colour as a hex colour string	"#e18383"
description	string	User-defined description of trigger	""
trigger_text	string	Generated description of when the trigger will run, based on its properties	"At startup"
conditions	array	Array of Condition objects (see below)	[{"text":"Before 12:00:00 every day"}]
actions	array	Array of Action objects (see below)	[{"text":"Start Timeline 1"}]

The Condition and Action objects have the following properties:

Attribute	Value Type	Description	Value Example
text	string	Generated description of the condition or action, based on its properties	"Start Timeline 1"

4.2.33 User

This allows user accounts on the controller to be added, modified, or removed.

Methods

POST

`POST /api/user`

Add a new user. The payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`session_password`	string	The password for the current session.	`"my_password"`
`username`	string	The name of the new user to add	`"bob"`
`password`	string	The new user's password.	`"bobs_password"`
`access`	array of strings	The access level(s) to grant the new user. Includes Admin, Control and Status.	`["Control", "Status"]`

PUT

`PUT /api/user`

Update a user account with a new password and/or access groups. The payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`session_password`	string	The password for the current session.	`"my_password"`
`"username"`	string	The name of the user to modify	`"bob"`
`password`	string	The user's updated password.	`"bobs_password"`
`access`	array of strings	The access level(s) to grant the user. Includes Admin, Control and Status.	`["Control", "Status"]`

DELETE

`DELETE /api/user`

Update a user account with a new password and/or access groups. The payload is a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`session_password`	string	The password for the current session.	`"my_password"`
`username`	string	The name of the user to delete	`"bob"`

4.2.34 User Groups

These methods allow discovery of the user and guest groups on the controller.

Methods

GET

`GET /api/user_groups`

Get the list of available user groups. Returns a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
user_groups	array of strings	The list of available groups.	["Admin", "Control", "Status"]

GET

`GET /api/guest_groups`

Get the list of available guest groups. Returns a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
guest_groups	array of strings	The list of available guest groups.	["Foo", "Bar"]

4.2.35 HTTP API Objects

Reference for objects used in the controller HTTP API.

DALI Power

The DALI power object has the following attributes:

Parameter	Value Type	Description	Value Example
dali_bus_uptime	integer	The amount of time the DALI bus has been up, in minutes	368
power_failures	array of date-time	A list of the time and dates of recent power failures	["01 Feb 2017 13:44:42", "30 Nov 2022 08:33:01"]

DALI Error

The DALI error object has the following attributes:

Parameter	Value Type	Description	Value Example
address	integer	The DALI bus address of the device with the error	12
test	string	The test that discovered the error	"Function"
error	string	A description of the DALI error	"Battery Duration"
fixed	boolean	Whether the error has been fixed. Once fixed, the error remains in the list until it is retested.	true

DALI Schedule

The DALI ballast status object has the following attributes:

Parameter	Value Type	Description	Value Example
next_function_test	date-time	The next date and time automated function test will occur	"01 Feb 2017 13:44:42"
next_duration_test	date-time	The next date and time automated duration test will occur	"01 Feb 2017 13:44:42"
prev_function_test	date-time	The previous date and time automated function test occurred	"01 Feb 2017 13:44:42"
prev_duration_test	date-time	The previous date and time automated duration test occurred	"01 Feb 2017 13:44:42"

DALI Ballast Status

The DALI ballast status object has the following attributes:

Parameter	Value Type	Description	Value Example
address	integer	The ballast address	12
user_name	string	The user assigned name of the ballast	"Center Room"
status	string	A string representing the current status of the ballast	"Lamp Failure"
actual_level	integer	The current actual output level of the ballast	128
battery_level	integer	For emergency ballasts only - the level of the battery reported	12
battery_charged	boolean	Whether or not the battery is charged	True
lamp_emergency_hours	integer	How many hours the fixture has been on in emergency state	12
lamp_total_hours	integer	How many hours the fixture has been on in total	400
last_status_check	date/time	The last date and time the ballast status was checked	0

RDM Device Info

Where an RDM Device Info object is returned from an API request, it will have the following attributes:

Parameter	Value Type	Description	Value Example
uid	string	Format is {manuId}:{deviceId}(:{subId}) where {manuId} is a padded unsigned hexadecimal integer of width 4, lowercase, e.g. 072c; {deviceId} is a padded unsigned hexadecimal integer of width 8, lowercase, e.g. 0004fe02; {subId} is an optional unsigned decimal integer.	"072c:0004fe02"
rdm_protocol_version	integer	16 bit value encoding the major version in the most significant byte and the minor version in the least significant byte. The current standard v1.0 is therefore 0x0100.	0x0100
device_model_id	integer	Device model ID of the Root Device or the Sub-Device. Must be unique within the products of a manufacturer.	1836
product_category	integer	16 bit value encoding the coarse category in the upper eight bits and the (optional) fine category in lower eight bits, e.g. 0x0100 is PRODUCT_CATEGORY_FIXTURE with no fine category.	0x0100
software_version_id	integer	Software version ID for the device, which is a 32-bit value determined by the manufacturer. It may use any encoding scheme such that the controller may identify devices containing the same software versions. Any devices from the same manufacturer with differing software will not report the same software version ID.	
dmx512_footprint	integer (0-512)	The DMX footprint of the device - the number of consecutive DMX slots required to patch the device. If the device is a sub-device, then the value is the DMX footprint for that sub-device. If the device is the root device, it is the footprint for the root device itself.	3
dmx512_personality	integer	16 bit field, encoding the current personality in the upper 8 bits and the total number of personalities supported by the device in the lower 8 bits.	0x0102
dmx512_start_address	integer	The DMX start address of the device, or 0xffff if the device has a DMX footprint of zero.	7
sub_device_count	integer	Number of sub devices represented by the root device. This value is always the same regardless of whether the device is the root device or a sub-device.	0
sensor_count	integer	Number of available sensors in a root device or sub-device. For sub-devices, this value is identical for any sub-device owned by the same root device. When a device or sub-device is fitted with a single sensor, it will return a value of 0x01 for the sensor count. This sensor would then be addressed as sensor number 0x00 when using the other sensor-related parameter messages.	0

RDM Universe Key

Used to specify the target universe for RDM operations. It is a JSON object with the following attributes:

Attribute	Value Type	Description
`protocol`	integer	Output protocol (see *Enumerated Protocols*).
`index`	integer	Only required for protocols `DMX` and `ART-NET`.
`remote_device_num`	integer	Only required for protocol `EDN`. The remote device number of the EDN node.
`remote_device_type`	integer	Only required for protocol `EDN`. The type of EDN as defined in *Enumerated EDN Device Types*.
`port`	integer	Only required for protocol `EDN`. The port on the EDN.

Enumerated Protocols

Constants for protocols are defined in query.js as follows:

Name	Value
`DMX`	1
`PATHPORT`	2
`ARTNET`	4
`KINET`	8
`SACN`	16
`DVI`	32
`RIO_DMX`	64
`EDN`	128

Enumerated EDN Device Types

Constants for EDN types are defined in query.js as follows:

Name	Value
`EDN20`	109
`EDN10`	110

JAVASCRIPT QUERY LIBRARY

Mosaic controllers provide a JavaScript library, accessible at `/default/js/query.js`. Controller projects may have a custom web interface, whose source files may include this library to provide convenient access to the controller HTTP API through JavaScript callbacks and to real time status updates through *websocket subscriptions*.

5.1 Including the Library

The query.js library may be included within the <head> in any HTML file within a custom web interface created for a Mosaic Designer project as follows:

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, user-
↪scalable=yes">
        <!--Include query.js library-->
        <script type="text/javascript" src="/default/js/query.js" defer></script>
    </head>
    <body>
    <!-- etc. -->
    </body>
</html>
```

5.2 Event Handlers

Functions are provided in the library to set event handlers.

- `set_success_handler(success)` - function passed as `success` will be called when a websocket connection is successfully established with the controller and when a response is received to an HTTP API request.

- `set_error_handler(error)` - function passed as `error` will be called when a websocket connection cannot be established with the controller and when an error is encountered as part of making an HTTP API request.

- `set_restart_handler(restart)` - function passed as `restart` will be called when the controller has restarted, at which point any users must authenticate again.

- `set_redirect_handler(redirect)` - function passed as `redirect` will be called when a request is unauthorized. The function will be passed the url of the default login page as a string, and may choose to return this (the default behaviour) or return the path of a custom login page.

For example:

```
Query.set_redirect_handler((suggestion) => {
  console.log("Suggested redirect: " + suggestion)
  return "/custom-login.html"
})
```

5.3 Querying and Controlling

The functions provided in query.js for querying and controlling the controller and its current project are in the following sections:

5.3.1 Beacon

Functions

toggle_beacon

Toggle beacon mode on the controller.

`toggle_beacon(callback)`

In beacon mode, a controller will flash its LEDs or it screen continuously.

5.3.2 Channel / Park

Functions

park_channel

Park an output channel or channels at a specified level.

`park_channel(params, callback)`

`params` is expected to be an object with the same attributes as the HTTP *POST* request.

unpark_channel

Unpark an output channel or channels.

`unpark_channel(params, callback)`

`params` is expected to be an object with the same attributes as the HTTP *DELETE* request.

5.3.3 Command

Functions

run_command

Run a Lua script or pass a command to the command line parser on the controller.

Note: The Command Line Parser must be enabled in the web interface settings of the current project, else this function will not be available.

`run_command(params, callback)`

`params` is expected to be an object with the same attributes as the HTTP *POST* request.

Returns `Executed` if the script was executed successfully or an error string if not.

5.3.4 Config

Functions

edit_config

Edits the configuration of the controller.

`edit_config(params, callback)`

`params` is expected to be an object with the same attributes as the HTTP *POST* request.

The `callback` function will be passed the same object as is received from the HTTP *POST* request.

get_config

Returns information about the queried controller's configuration.

`get_config(callback)`

Returns an object with the same attributes as in the HTTP *GET* response.

For example:

```
Query.get_config(config => {
  let year = config.year
})
```

5.3.5 Content Targets

Note: Atlas/Atlas Pro only

Functions

master_content_target_intensity

`master_content_target_intensity(params, callback)`

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`type`	string	Optional. Type of content target (only relevant on Atlas Pro): `primary`, `secondary`, `target_3`, `target_4`, `target_5`, `target_6`, `target_7`, `target_8`. Defaults to `primary`.	`"secondary"`
`level`	float or string containing a bounded integer	Master level to set on the group	`0.5` or `"50:100"`
`fade`	float	Optional. Fade time to apply the intensity change, in seconds.	`2.0`
`delay`	float	Optional. Time to wait before applying the intensity change, in seconds.	`2.0`

get_content_target_info

`get_content_target_info(callback)`

Returns an object with a single `content_targets` attribute, which has an array value. Each item in the array is a Content Target object with the same attributes as in the HTTP *GET* response.

For example:

```
Query.get_content_target_info(c => {
  let level = c.content_targets[0].level // level of primary content target
})
```

5.3.6 Controller

Functions

get_controller_info

`get_controller_info(callback)`

Returns an object with a single `controllers` attribute, which has an array value. Each item in the array is a Controller object with the same attributes as in the HTTP *GET* response.

For example:

```
Query.get_controller_info(data => {
    for(index in data.controllers) {
        console.log("Controller " + index + " name is " + data.controllers[index].name);
    }
});
```

Will print out the name of each controller to the console.

5.3.7 Group

Note: Not applicable to Atlas/Atlas Pro

Functions

master_intensity

`master_intensity(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	integer	Group number. Group 0 means the *All Fixtures* group.	1
`level`	float or string containing a bounded integer	Master level to set on the group	`0.5 or "50:100"`
`fade`	float	Optional. Fade time to apply the intensity change, in seconds.	`2.0`
`delay`	float	Optional. Time to wait before applying the intensity change, in seconds.	`2.0`

For example:

```
// Master group 1 to 50% in 3 seconds
Query.master_intensity({
  "num":1,
  "level":"50:100",
  "fade":3
}, result => {
  // Check for error
})
```

get_group_info

Returns data about the fixture groups in the project.

```
get_group_info(callback[, num])
```

Returns an object with a single `groups` attribute, which has an array value. Each item in the array is a Group object with the same attributes as in the HTTP *GET* response.

`num` can be used to filter which groups are returned and is expected to be a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	string **or** integer	Define the numbers of the group that should be returned	`"1,2,5-9"` **or** 5

Note: Group 0 will return data about the *All Fixtures* group.

For example:

```
Query.get_group_info(g => {
  let name = g.groups[0].name // name of the first group returned
}, {"num":"2-4"})
```

5.3.8 Input

There's no function in the JavaScript Query library to get the digital & analogue inputs at the moment.

5.3.9 Log

There's no function in the JavaScript Query library to get the log at the moment.

5.3.10 Lua Variable

Functions

get_lua_variables

Returns the current value of specified Lua variables.

```
get_lua_variables(luaVariables, callback)
```

Returns an object with the requested Lua variables and their values as key/value pairs, in the same manner as the HTTP *GET* request.

`luaVariables` can be a string or an array of strings, where each string is a Lua variable name. The Lua variable must be directly accessible from the Lua global table.

For example:

```lua
--[[ Lua definitions ]]--
foo = 'spam'
bar = {
    a = 'ham',
    b = 100
}
local baz = 'eggs'
```

```javascript
/* Javascript Query */
Query.get_lua_variables(["foo","bar"], v => {
  let foo = v.foo // foo contains "spam"
  console.log(typeof foo) // Output: "string"
  let bar = v.bar // bar contains a javascript object { a: "ham", b: 100 }
  console.log(typeof bar) // Output: "object"
  console.log(typeof bar.a) // Output: "string"
  console.log(typeof bar.b) // Output: "number"
})

// Invalid query, `a` is a child of `bar` and not directly accessible from the global␣
↪table
Query.get_lua_variables(["bar.a"], v => {})

// Invalid query, `baz` is scoped locally, and inaccessible from the global table
Query.get_lua_variables(["baz"], v => {})
```

5.3.11 Output

Functions

disable_output

Disable the output of a specified protocol from the controller. Propagates to all controllers in a project.

```
disable_output(params, callback)
```

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`protocol`	string	Protocol to disable. Options: `dmx`, `pathport`, `sacn`, `art-net`, `kinet`, `rio-dmx`, `edn`, `edn-spi`.	`"parthport"`

enable_output

Enable the output of a specified protocol from the controller. Propagates to all controllers in a project.

```
enable_output(params, callback)
```

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`protocol`	string	Protocol to enable. Options: `dmx`, `pathport`, `sacn`, `art-net`, `kinet`, `rio-dmx`, `edn`, `edn-spi`.	`"parthport"`

get_output

Returns the lighting levels being output by the queried controller.

```
get_output(universeKey, callback)
```

Returns an object with the same attributes as in the HTTP *GET* response.

`universeKey` can be a string (see *Universe Key String Format*) or it can be an object with the following attributes:

Attribute	Value Type	Description
`protocol`	integer	Output protocol (see *Enumerated Protocols*)
`index`	integer	Required unless `protocol` is KINET, RIO_DMX or EDN
`kinet_power_supply_num`	integer	Only required if `protocol` is KINET
`kinet_port`	integer	Only required if `protocol` is KINET
`remote_device_type`	integer	Only required if `protocol` is RIO_DMX or EDN (see *Enumerated Remote Device Types*)
`remote_device_num`	integer	Only required if `protocol` is RIO_DMX or EDN
`port`	integer	Only required if `protocol` is EDN

For example:

```javascript
Query.get_output({
    protocol: KINET,
    kinet_port: 1,
    kinet_power_supply_num: 1
  }, u => {
  console.log(u)
  }
)

Query.get_output({
    protocol: DMX,
    index: 1
  }, u => {
    console.log(u)
  }
)

Query.get_output("dmx:1", u => {
```

```
    console.log(u)
})
```

Universe Key String Format

A universe key string takes the form:

- `protocol:index` for protocols `dmx`, `pathport`, `sacn`, `art-net`;
- `protocol:kinetPowerSupplyNum:kinetPort` for protocol `kinet`;
- `protocol:remoteDeviceType:remoteDeviceNum` for protocol `rio-dmx`;
- `protocol:remoteDeviceType:remoteDeviceNum:port` for protocols `edn`, `edn-spi`.

Where:

- `kinetPowerSupplyNum` is an integer;
- `kinetPort` is an integer;
- `remoteDeviceType` can be `rio08`, `rio44` or `rio80`, `edn10` or `edn20`;
- `remoteDeviceNum` is an integer;
- `port` is an integer.

For example:

- `"dmx:1"`
- `"rio-dmx:rio44:1"`

Enumerated Protocols

Constants for protocols are defined in query.js as follows:

Name	Value
DMX	1
PATHPORT	2
ARTNET	4
KINET	8
SACN	16
DVI	32
RIO_DMX	64
EDN	128

Enumerated Remote Device Types

Constants for RIO types are defined in query.js as follows:

Name	Value
RIO80	101
RIO44	102
RIO08	103

Constants for EDN types are defined in query.js as follows:

Name	Value
EDN20	109
EDN10	110

5.3.12 Override

Functions

set_group_override

Set the Intensity, Red, Green, Blue levels for a group. Propagates to all controllers in a project.

`set_group_override(params, callback)`

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Group or fixture number, depending on target. Group 0 means the *All Fixtures* group.	1
intensity	integer or string	Optional. Either an integer (0-255) representing the intensity to set as part of override **or** the string "snapshot" to capture the current intensity of the fixture(s) and set this as the override value. Intensity override will not be changed if this attribute isn't provided.	128
colour	*Override Colour* or string	Optional. Specifies the colour to set as part of the override. Either an *Override Colour* or the string "snapshot" to capture the current colour of the fixture(s) and set this as the override. JSON object with the same attributes as the HTTP *PUT* request.	
temperature	integer or string	Optional. Either an integer (0-255) representing the temperature component to set as part of override **or** the string "snapshot" to capture the current temperature component of the fixture(s) and set this as the override value. Temperature override will not be changed if this attribute isn't provided.	128
fade	float	Optional. Fade time to apply the override change, in seconds.	2.0
path	string	Optional. Crossfade path to use when applying the override: Default, Linear, Start, End, Braked, Accelerated, Damped, Overshoot, Col At Start, Col At End, Int At Start, Int At End, Colour First, Intensity First	"Braked"

clear_group_overrides

Release any overrides on a group, or all groups. Propagates to all controllers in a project.

clear_group_overrides(params, callback)

params is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Optional. Group number. If not provided, all overrides are cleared.	1
fade	float	Optional. Fade time in which to release overrides, in seconds.	2.0

set_fixture_override

Set the Intensity, Red, Green, Blue levels for a fixture. Propagates to all controllers in a project.

`set_fixture_override(params, callback)`

`params` is expected to be an object with the same attributes as for *set_group_override*.

clear_fixture_overrides

Release any overrides on a fixture, or all fixtures. Propagates to all controllers in a project.

`clear_fixture_overrides(params, callback)`

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	integer	Optional. Fixture number. If not provided, all overrides are cleared.	1
`fade`	float	Optional. Fade time in which to release overrides, in seconds.	`2.0`

clear_all_overrides

Release all overrides. Propagates to all controllers in a project.

`clear_all_overrides(params, callback)`

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`fade`	float	Optional. Fade time in which to release overrides, in seconds.	`2.0`

5.3.13 Project

Functions

get_project_info

Returns data about the current project.

`get_project_info(callback)`

Returns an object with the same attributes as in the HTTP *GET* response.

For example:

```
Query.get_project_info(project => {
  const author = project.author
})
```

5.3.14 Protocol

Functions

get_protocols

Returns all the universes in the project on the queried controller.

`get_protocols(callback)`

Returns an object with a single `outputs` attribute, which has an array value. Each item in the array is a Protocol object with the same attributes as in the HTTP *GET* response.

For example:

```
Query.get_protocols(p => {
  const protocol_name = p.outputs[0].name // name of the first protocol
})
```

5.3.15 RDM Discovery

Functions

start_rdm_discovery

Request to start a full RDM discovery. Results are available via *subscribe_rdm_discovery*.

`start_rdm_discovery(params, callback)`

`params` is expected to be an object with the same attributes as the HTTP *POST* request.

5.3.16 RDM Get

Functions

start_rdm_get

Request to start an RDM Get operations. Results are available via *subscribe_rdm_get_set*.

`start_rdm_get(params, callback)`

`params` is expected to be an object with the same attributes as the HTTP *POST* request.

5.3.17 RDM Set

Functions

start_rdm_set

Request to start an RDM Set operations. Results are available via *subscribe_rdm_get_set*.

`start_rdm_set(params, callback)`

`params` is expected to be an object with the same attributes as the HTTP *POST* request.

5.3.18 Remote Device

Functions

get_remote_device_info

Returns data about all the remote devices in the project.

`get_remote_device_info(callback)`

Returns an object with a single `remote_devices` attribute, which has an array value. Each item in the array is a Remote Device object with the same attributes as in the HTTP *GET* response.

For example:

```
Query.get_remote_device_info(r => {
  const type = r.remote_devices[0].type // type of the first remote device
})
```

5.3.19 Replication

Functions

get_replication

Returns data about the install replication.

`get_replication(callback)`

Returns an object with the same attributes as in the HTTP *GET* response.

5.3.20 Scene

Functions

start_scene

`start_scene(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Scene number	5

For `callback` please see *JavaScript Command Callback*.

release_scene

`release_scene(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Scene number	5
fade	float	Optional. Release fade time in seconds. If not provided, the default fade time will be used.	2.0

For `callback` please see *JavaScript Command Callback*.

toggle_scene

`toggle_scene(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Scene number	5
fade	float	Optional. The release fade time in seconds to apply if the toggle action results in the scene being released. If not provided, the default fade time will be used.	2.0

For `callback` please see *JavaScript Command Callback*.

release_all_scenes

`release_all_scenes(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`fade`	float	Optional. Release fade time in seconds. If not provided, the default fade time will be used.	`2.0`
`group`	string	Optional. Scene group name: `A` through `H`. Prepend the group name with `!` to apply the action to all groups *except* the specified group, e.g. `!A`.	`"B"`

For `callback` please see *JavaScript Command Callback*.

release_all

Release all timelines and scenes. Propagates to all controllers in a project.

`release_all(params, callback)`

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`fade`	float	Optional. Release fade time in seconds. If not provided, the default fade time will be used.	`2.0`
`group`	string	Optional. Timeline/Scene group name: `A` through `H`. Prepend the group name with `!` to apply the action to all groups *except* the specified group, e.g. `!A`.	`"B"`

For `callback` please see *JavaScript Command Callback*.

get_scene_info

Returns data about the scenes in the project and their state on the controller.

`get_scene_info(callback[, num])`

Returns an object with a single `scenes` attribute, which has an array value. Each item in the array is a Scene object with the same attributes as in the HTTP GET response.

`num` can be used to filter which scenes are returned and is expected to be a JSON object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	string **or** integer	Define the numbers of the scene that should be returned	`"1,2,5-9"` **or** `5`

For example:

```
Query.get_scene_info(s => {
  let name = s.scenes[0].name // name of the first scene returned
}, {"num":"1,2-5"})
```

JavaScript Command Callback

Functions in the JavaScript API that perform actions on the controller, e.g. `start_timeline`, have an optional `callback` argument. This expects a function, which is called when a response to the underlying HTTP API request is received. Its argument, if non-null, is the response body. If the content type of the response was `"application/json"` then the argument will be an object - the result of parsing the body as JSON.

5.3.21 System

Functions

get_system_info

`get_system_info(callback)`

Returns an object with the same attributes as in the HTTP *GET* response.

For example:

```
Query.get_system_info(system => {
  const capacity = system.channel_capacity
})
```

5.3.22 Temperature

Functions

get_temperature

`get_temperature(callback)`

Returns an object with the same attributes as in the HTTP *GET* response.

For example:

```
Query.get_temperature(temp => {
  const ambient = temp.ambient_temp
})
```

5.3.23 Text Slots

Functions

set_text_slot

Set the value of a text slot used in the project, which will propagate to all controllers in a project.

`set_text_slot(params, callback)`

`params` is expected to be an object with the same attributes as the HTTP *PUT* request.

get_text_slot

Returns data about the text slots in the project and their current values.

`get_text_slot(callback[, filter])`

Returns an object with a single `text_slots` attribute, which has an array value. Each item in the array is a Text Slot object with the same attributes as in the HTTP *GET* response.

`filter` can be used to filter which text slots are returned and is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`names`	string **or** array	Define the names of the text slots that should be returned, either as a single string or an array of strings	`["test_slot1", "anotherSlot"]` **or** `"test_slot1"`

For example:

```
Query.get_text_slot(t => {
  let value = t.text_slots[0].value // value of the first text slot returned
}, {"names":["test_slot1","test_slot2"]})
```

5.3.24 Time

Functions

get_current_time

`get_current_time(callback)`

Returns an object with the same attributes as in the *GET* GET response.

For example:

```
Query.get_current_time(time => {
  const uptime = time.uptime
})
```

5.3.25 Timeline

Functions

start_timeline

`start_timeline(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Timeline number	5

For `callback` please see *JavaScript Command Callback*.

release_timeline

`release_timeline(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Timeline number	5
fade	float	Optional. Release fade time in seconds. If not provided, the default fade time will be used.	2.0

For `callback` please see *JavaScript Command Callback*.

toggle_timeline

`toggle_timeline(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Timeline number	5
fade	float	Optional. The release fade time in seconds to apply if the toggle action results in the timeline being released. If not provided, the default fade time will be used.	2.0

For `callback` please see *JavaScript Command Callback*.

pause_timeline

`pause_timeline(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Timeline number	5

For `callback` please see *JavaScript Command Callback*.

resume_timeline

`resume_timeline(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Timeline number	5

For `callback` please see *JavaScript Command Callback*.

pause_all

Pause all timelines in the project which are currently running. Propagates to all controllers in a project.

`pause_all(callback)`

For `callback` please see *JavaScript Command Callback*.

resume_all

Resume all timelines in the project which are currently paused. Propagates to all controllers in a project.

`resume_all(callback)`

For `callback` please see *JavaScript Command Callback*.

release_all_timelines

`release_all_timelines(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`fade`	float	Optional. Release fade time in seconds. If not provided, the default fade time will be used.	`2.0`
`group`	string	Optional. Timeline group name: `A` through `H`. Prepend the group name with `!` to apply the action to all groups *except* the specified group, e.g. `!A`.	`"B"`

For `callback` please see *JavaScript Command Callback*.

release_all

Release all timelines and scenes. Propagates to all controllers in a project.

`release_all(params, callback)`

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`fade`	float	Optional. Release fade time in seconds. If not provided, the default fade time will be used.	`2.0`
`group`	string	Optional. Timeline/Scene group name: `A` through `H`. Prepend the group name with `!` to apply the action to all groups *except* the specified group, e.g. `!A`.	`"B"`

For `callback` please see *JavaScript Command Callback*.

set_timeline_rate

`set_timeline_rate(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	integer	Timeline number	5
`rate`	string	A string containing a floating point number or a bounded integer, where 1.0 means the timeline's default rate.	`"0.1"` or `"10:100"`

For `callback` please see *JavaScript Command Callback*.

set_timeline_position

`set_timeline_position(params, callback)`

Propagates to all controllers in a project.

`params` is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	integer	Timeline number	5
`position`	string	A string containing a floating point number or a bounded integer, representing a fraction of the timeline length.	`"0.1"` or `"10:100"`

For `callback` please see *JavaScript Command Callback*.

get_timeline_info

`get_timeline_info(callback[, num])`

Returns data about the timelines in the project and their state on the controller.

Returns an object with a single `timelines` attribute, which has an array value. Each item in the array is a Timeline object with the same attributes as in the HTTP GET response.

`num` can be used to filter which timelines are returned and is expected to be an object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	string **or** integer	Define the numbers of the timeline that should be returned	`"1,2,5-9"` **or** 5

For example:

```
Query.get_timeline_info(t => {
  let name = t.timelines[0].name // name of the first timeline returned
}, {"num":"1-4"})
```

JavaScript Command Callback

Functions in the JavaScript API that perform actions on the controller, e.g. `start_timeline`, have an optional `callback` argument. This expects a function, which is called when a response to the underlying HTTP API request is received. Its argument, if non-null, is the response body. If the content type of the response was `"application/json"` then the argument will be an object - the result of parsing the body as JSON.

5.3.26 Trigger

Functions

fire_trigger

```
fire_trigger(params, callback)
```

`params` is expected to be an object with the same attributes as the HTTP *POST* request.

get_trigger_info

```
get_trigger_info(callback[, type])
```

Returns an object with a single `triggers` attribute, which has an array value. Each item in the array is a Trigger object with the same attributes as in the HTTP *GET* response.

`type` is expected to be a string and can be used to filter the type of trigger returned. For example, `"Timeline Started"` would return only Timeline Started triggers in the project.

For example:

```
Query.get_trigger_info(t => {
  let name = t.triggers[0].name // name of first startup trigger returned
}, "Startup")
```

5.4 Subscriptions

Websocket subscriptions allow data to be pushed to the web client whenever there is a change within the project. The query.js library includes *functions* with callbacks to subscribe to each channel and return any data received.

5.4.1 Websocket Subscriptions

Websocket subscriptions allow data to be pushed to the web client whenever there is a change within the project. The query.js library includes functions with callbacks to subscribe to each channel and return any data received.

Functions

subscribe_timeline_status

Subscribe to changes in timeline status.

```
subscribe_timeline_status(callback)
```

The `callback` is called each time a timeline changes state on the controller. Each time it is passed an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Timeline number	1
state	string	The new state of the timeline: `none`, `running`, `paused`, `holding_at_end`, `released`	`"running"`
onstage	boolean	Whether the timeline is currently affecting the output of any fixtures in the project.	`true`
position	integer	Current time position of the timeline playback, in milliseconds	`5000`

For example:

```
Query.subscribe_timeline_status(t => {
  alert(t.num + ": " + t.state)
})
```

subscribe_scene_status

Subscribe to changes in scene status.

subscribe_scene_status(callback)

The `callback` is called each time a scene changes state on the controller. Each time it is passed an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Scene number	1
state	string	The new state of the scene: `none`, `started`, `released`	`"started"`
onstage	boolean	Whether the scene is currently affecting the output of any fixtures in the project.	`true`

For example:

```
Query.subscribe_scene_status(s => {
  alert(s.num + ": " + s.state)
})
```

subscribe_group_status

Subscribe to changes in group level, as set by the Master Intensity action.

subscribe_group_status(callback)

The `callback` is called each time the group master level changes on the controller. Each time it is passed an object with the following attributes:

Attribute	Value Type	Description	Value Example
num	integer	Group number	1
name	string	Group name	`"Group 1"`
level	integer	New master intensity level of the group: 0-255	128

For example:

```
Query.subscribe_group_status(g => {
  alert(g.num + ": " + g.level)
})
```

subscribe_remote_device_status

Subscribe to changes in remote device online/offline status.

`subscribe_remote_device_status(callback)`

The `callback` is called each time the remote device online/offline status changes. Each time it is passed an object with the following attributes:

Attribute	Value Type	Description	Value Example
`num`	integer	Remote device number	1
`type`	string	Type of remote device: `RIO 80`, `RIO 44`, `RIO 08`, `BPS`, `RIO A`, `RIO D`, `EDN 20`, `EDN 10`	`"RIO 80"`
`online`	boolean	New online state of the remote device	`true`
`serial`	string	Remove device serial number	`"001001"`

For example:

```
Query.subscribe_remote_device_status(r => {
  alert(r.num + ": " + (r.online ? "online" : "offline"))
})
```

subscribe_beacon

Subscribe to changes in the device beacon.

`subscribe_beacon(callback)`

The `callback` is called each time the controller beacon status changes. Each time it is passed an object with the following attributes:

Attribute	Value Type	Description	Value Example
`on`	boolean	New beacon status	`true`

For example:

```
Query.subscribe_beacon(b => {
  alert(b.on ? "Beacon turned on" : "Beacon turned off")
})
```

subscribe_lua

The receiver for the `push_to_web()` Lua function.

```
subscribe_lua(callback)
```

The `callback` is called each time a script on the controller calls the `push_to_web()` function. Each time it is passed an object with a single attribute - the name or key string passed as the first argument to `push_to_web()`. The value of this attribute is the second argument passed to `push_to_web()`, converted to a string.

For example, if a project needs to send a touch slider level to the web interface, it might have the following in a trigger Lua script:

```lua
level = getMySliderLevel() -- user-defined function to get the current slider level
push_to_web("slider_level", level) -- invoke callbacks on subscribers
```

If `level` is equal to e.g. 56 then the object passed the JavaScript callback will be:

```json
{
  "slider_level": "56"
}
```

And the subscription could be setup as follows:

```javascript
Query.subscribe_lua(l => {
 key = Object.keys(l)[0] // "slider_level" in the above example
 value = l.key            // "56" in the above example
 alert(key + ": " + value)
})
```

subscribe_rdm_discovery

Subscribe for results from RDM discovery operations.

```
subscribe_rdm_discovery(callback)
```

The callback is called every time an RDM device is found during an RDM discovery operation, and to announce when the RDM discovery operation is finished or has been cancelled. The callback is passed an object with the following attributes:

Attribute	Value Type	Description
message_type	string	Categorises the message, defining what `data` is present, if any (see below).
universe	string	The universe on which the RDM operation is acting, in the *Universe Key String Format*.
data	object	Optional. Data appropriate for the message type.

Device found

`"message_type" : "device_found"`

The `data` object will have the following attributes:

Attribute	Value Type	Description
device_info	*RDM Device Info*	RDM device info from the discovered device.
fixture_num	integer	User number of the fixture in the project with the same DMX address and footprint as the discovered device, or *null* if there is no matching fixture in the project.

Discovery finished

`"message_type" : "finished"`

The `data` object will not be present, or will be empty.

Discovery cancelled

`"message_type" : "cancelled"`

The `data` object will have the following attributes:

Attribute	Value Type	Description
error	string	A description of why the discovery was cancelled.

subscribe_rdm_get_set

Subscribe for results from RDM Get and Set operations.

`subscribe_rdm_get_set(callback)`

The callback is called to provide the response from RDM Get and Set operations, and to announce when the RDM operation is finished or has been cancelled. The callback is passed an object with the following attributes:

Attribute	Value Type	Description
message_type	string	Categorises the message, defining what `data` is present, if any (see below).
universe	string	The universe on which the RDM operation is acting, in the *Universe Key String Format*.
device_id	string	Format is {manuId}:{deviceId}(:{subId}) where {manuId} is a padded unsigned hexadecimal integer of width 4, lowercase, e.g. `072c`; {deviceId} is a padded unsigned hexadecimal integer of width 8, lowercase, e.g. `0004fe02`; {subId} is an optional unsigned decimal integer.
pid	string	RDM PID as a human-readable string, e.g. `DEVICE_INFO`, or a string containing the hex representation of the enum value of the PID as defined by the RDM standard, e.g. `"c1"`.
data	object	Optional. Data appropriate for the message type.

Get Finished

`"message_type" : "get_finished"`

The GET operation indicated by the PID has finished. No `data` object is expected.

Set Finished

`"message_type" : "set_finished"`

The SET operation indicated by the PID has finished. No `data` object is expected.

Get/Set result error

`"message_type" : "result_error"`

The `data` object will have the following attributes:

Attribute	Value Type	Description
error	string	Description of the error with the response.

Get/Set operation cancelled

`"message_type" : "get_cancelled" "message_type" : "set_cancelled"`

The `data` object will have the following attributes:

Attribute	Value Type	Description
error	string	Description of why the operation was cancelled.

Get/Set Result

`"message_type" : "result"`

Provides the results of the operation, parsed from the response from the device. The `data` object will be appropriate for the PID. If `pid` is a human-readable string, e.g. `DEVICE_INFO` then `data` is described under *RDM PID result data*. Otherwise, if `pid` is the hex representation of the enum value of a PID, then `data` will have one key, `raw`, the value of which will be the base64-encoded raw payload data received from the device.

RDM PID result data

When the object passed to the `subscribe_rdm_get_set` callback has `"message_type": "result"` and where `pid` is a human-readable string, e.g. `DEVICE_INFO`, the format of the `data` object is described in one of the following sections.

Get Communication Status (COMMS_STATUS)

Following a successful GET operation for `COMMS_STATUS`, the `data` object in the `subscribe_rdm_get_set` callback argument will have the following attributes, which map to the attributes of the same names in the RDM specification for this response:

- `short_message` - number (16 bit)

- `length_mismatch` - number (16 bit)

- `checksum_fail` - number (16 bit)

Get Status Messages (STATUS_MESSAGES)

Following a successful GET operation for `STATUS_MESSAGES`, the `data` object in the `subscribe_rdm_get_set` callback argument will have a `status_messages` attribute with an array value, the items of which will each have the following attributes, which map to the attributes of the same names in the RDM specification for this response:

- `sub_device_id` - number (16 bit)

- `status_type` - number (8 bit)

- `status_message_id` - number (16 bit)

- `data_value_1` - number (16 bit)

- `data_value_2` - number (16 bit)

Get Supported Parameters (SUPPORTED_PARAMETERS)

Following a successful GET operation for `SUPPORTED_PARAMETERS`, the `data` object in the `subscribe_rdm_get_set` callback argument will have a `supported_parameters` attribute with an array value. The array will contain numbers, corresponding to the 16 bit parameter IDs supported by the RDM device, as described in the RDM specification.

Get Parameter Description (PARAMETER_DESCRIPTION)

Following a successful GET operation for `PARAMETER_DESCRIPTION`, the `data` object in the `subscribe_rdm_get_set` callback argument will have the following attributes, which map to the attributes of the same names in the RDM specification for this response:

- `pid_requested` - number (16 bit)

- `pdl_size` - number (8 bit)

- `data_type` - number (8 bit)

- `command_class` - number (8 bit)

- `type` - number (8 bit)

- `unit` - number (8 bit)

- `prefix` - number (8 bit)

- `min_valid_value` - number (32 bit)

- `max_valid_value` - number (32 bit)

- `default_value` - number (32 bit)

- `description` - string (ASCII, max 32 characters)

Get Device Info (DEVICE_INFO)

Following a successful GET operation for `DEVICE_INFO`, the `data` object in the `subscribe_rdm_get_set` callback argument will have the following attributes, which map to the attributes of the same names in the RDM specification for this response:

- `rdm_protocol_version` - number (16 bit)

- `device_model_id` - number (16 bit)

- `product_category` - number (16 bit)

- `software_version_id` - number (32 bit)

- `dmx512_footprint` - number (16 bit)

- `dmx512_personality` - number (16 bit)

- `start_address` - number (16 bit)

- `sub_device_count` - number (16 bit)

- `sensor_count` - number (8 bit)

Get Device Model Description (DEVICE_MODEL_DESCRIPTION)

Following a successful GET operation for `DEVICE_MODEL_DESCRIPTION`, the `data` object in the `subscribe_rdm_get_set` callback argument will have a `model_description` attribute with a string value. The string will be the ASCII model description, 0-32 characters, as described in the RDM specification.

Get Manufacturer Label (MANUFACTURER_LABEL)

Following a successful GET operation for `MANUFACTURER_LABEL`, the `data` object in the `subscribe_rdm_get_set` callback argument will have a `manufacturer_label` attribute with a string value. The string will be the ASCII manufacturer description, 0-32 characters, as described in the RDM specification.

Get/Set Device Label (DEVICE_LABEL)

Following a successful GET operation for `DEVICE_LABEL`, the `data` object in the `subscribe_rdm_get_set` callback argument will have a `device_label` attribute with a string value. The string will be the current ASCII device label, 0-32 characters, as described in the RDM specification.

No `data` is expected in the response for a SET operation.

Get/Set Factory Defaults (FACTORY_DEFAULTS)

Following a successful GET operation for `FACTORY_DEFAULTS`, the `data` object in the `subscribe_rdm_get_set` callback argument will have a `factory_defaults` attribute with a boolean value, indicating whether the device is currently set to is factory defaults.

No `data` is expected in the response for a SET operation.

Get Software Version Label (SOFTWARE_VERSION_LABEL)

Following a successful GET operation for `SOFTWARE_VERSION_LABEL`, the `data` object in the `subscribe_rdm_get_set` callback argument will have a `software_version_label` attribute with a string value. The string will be the ASCII software version label, 0-32 characters, as described in the RDM specification.

Get Boot Software Version ID (BOOT_SOFTWARE_VERSION_ID)

Following a successful GET operation for `BOOT_SOFTWARE_VERSION_ID`, the `data` object in the `subscribe_rdm_get_set` callback argument will have a `boot_software_version_id` attribute with a 32 bit number value, as described in the RDM specification.

Get Boot Software Version Label (BOOT_SOFTWARE_VERSION_LABEL)

Following a successful GET operation for `BOOT_SOFTWARE_VERSION_LABEL`, the `data` object in the `subscribe_rdm_get_set` callback argument will have a `boot_software_version_label` attribute with a string value. The string will be the ASCII boot version label, 0-32 characters, as described in the RDM specification.

Get/Set DMX512 Personality (DMX_PERSONALITY)

Following a successful GET operation for `DMX_PERSONALITY`, the `data` object in the `subscribe_rdm_get_set` callback argument will have the following attributes, which map to the attributes of the same names in the RDM specification for this response:

- `current_personality` - number (8 bit)
- `num_personalities` - number (8 bit)

No `data` is expected in the response for a SET operation.

Get DMX512 Personality Description (DMX_PERSONALITY_DESCRIPTION)

Following a successful GET operation for `DMX_PERSONALITY_DESCRIPTION`, the `data` object in the `subscribe_rdm_get_set` callback argument will have the following attributes, which map to the attributes of the same names in the RDM specification for this response:

- `personality_requested` - number (8 bit)
- `dmx512_slots_required` - number (16 bit)
- `description` - string (ASCII, 0-32 characters)

Get/Set DMX512 Starting Address (DMX_START_ADDRESS)

Following a successful GET operation for `DMX_START_ADDRESS`, the `data` object in the `subscribe_rdm_get_set` callback argument will have a `dmx512_address` attribute with a 16 bit number value, as described in the RDM specification.

No `data` is expected in the response for a SET operation.

Get Slot Info (SLOT_INFO)

Following a successful GET operation for `SLOT_INFO`, the `data` object in the `subscribe_rdm_get_set` callback argument will have a `slot_info` attribute with an array value, the items of which will each have the following attributes, which map to the attributes of the same names in the RDM specification for this response:

- `slot_offset` - number (16 bit)
- `slot_type` - number (8 bit)
- `slot_label_id` - number (16 bit)

Get Slot Description (SLOT_DESCRIPTION)

Following a successful GET operation for `SLOT_DESCRIPTION`, the `data` object in the `subscribe_rdm_get_set` callback argument will have the following attributes, which map to the attributes of the same names in the RDM specification for this response:

- `slot_number_requested` - number (16 bit)
- `description` - string (ASCII, 0-32 characters)

Get Sensor Definition (SENSOR_DEFINITION)

Following a successful GET operation for `SENSOR_DEFINITION`, the `data` object in the `subscribe_rdm_get_set` callback argument will have the following attributes, which map to the attributes of the same names in the RDM specification for this response:

- `sensor_number_requested` - number (8 bit)
- `type` - number (8 bit)
- `unit` - number (8 bit)
- `prefix` - number (8 bit)
- `range_minimum_value` - number (16 bit)
- `range_maximum_value` - number (16 bit)
- `normal_minimum_value` - number (16 bit)
- `normal_maximum_value` - number (16 bit)
- `recorded_value_support` - number (8 bit)
- `description` - string (ASCII, 0-32 characters)

Get/Set Sensor (SENSOR_VALUE)

Following a successful GET or SET operation for `SENSOR_VALUE`, the `data` object in the `subscribe_rdm_get_set` callback argument will have the following attributes, which map to the attributes of the same names in the RDM specification for this response:

- `sensor_number_requested` - number (8 bit)

- `present_value` - number (16 bit)

- `lowest_detected_value` - number (16 bit)

- `highest_detected_value` - number (16 bit)

- `recorded_value` - number (16 bit)

Get/Set Lamp Hours (LAMP_HOURS)

Following a successful GET or SET operation for `LAMP_HOURS`, the `data` object in the `subscribe_rdm_get_set` callback argument will have the following attributes, which map to the attributes of the same names in the RDM specification for this response:

- `lamp_hours` - number (32 bit)

Get/Set Lamp State (LAMP_STATE)

Following a successful GET or SET operation for `LAMP_STATE`, the `data` object in the `subscribe_rdm_get_set` callback argument will have the following attributes, which map to the attributes of the same names in the RDM specification for this response:

- `lamp_state` - number (8 bit)

Universe Key String Format

A universe key string for RDM takes the form:

- `protocol:index` for protocols `dmx` and `art-net`;

- `protocol:remoteDeviceType:remoteDeviceNum:port` for protocol `edn`.

Where:

- `remoteDeviceType` can be `edn10` or `edn20`;

- `remoteDeviceNum` is an integer;

- `port` is an integer.

For example:

- `"dmx:1"`

- `"edn:edn20:1:5"`

LUA API

Mosaic controllers offer a Lua API providing access to system information, playback functions and trigger operations.

6.1 Adjustment Target

Note: Only supported on Atlas Pro.

An `Adjustment` object is returned from *get_adjustment*.

6.1.1 Properties

Property	Value Type
`rotation_offset`	float
`x_position_offset`	float
`y_position_offset`	float

For example:

```
target = get_adjustment(1)
r_offset = target.rotation_offset
```

6.1.2 Member functions

The following are member functions of `Adjustment` objects.

transition_rotation

`transition_rotation([angle[, count[, period[, delay[, useShortestPath]]]]])`

Applies a rotation to the adjustment target according to the parameters:

Parameter	Value Type	Description	Value Example
`angle`	float	Optional. Angle of rotation to transition to, in degrees. Defaults to zero.	`90.0`
`count`	integer	Number of times to repeat the rotation transformation.	`1`
`period`	integer	The period of the rotation, in seconds - the time to perform one count of the transformation.	`2`
`delay`	integer	Time to wait before starting the rotation, in seconds.	`0`

transition_x_position

`transition_x_position([x_offset[, count[, period[, delay]]]])`

Moves the adjustment target along the x axis according to the parameters:

Parameter	Value Type	Description	Value Example
`x_offset`	float	Optional. Offset to apply to the x position. Defaults to 0.	`25.0`
`count`	integer	Number of times to repeat the x translation.	`1`
`period`	integer	The period of the translation, in seconds - the time to perform one count of the transformation.	`2`
`delay`	integer	Time to wait before starting the translation, in seconds.	`0`

transition_y_position

`transition_y_position([x_offset[, count[, period[, delay]]]])`

Moves the adjustment target along the y axis according to the parameters:

Parameter	Value Type	Description	Value Example
`y_offset`	float	Optional. Offset to apply to the y position. Defaults to 0.	`25.0`
`count`	integer	Number of times to repeat the y translation.	`1`
`period`	integer	The period of the translation, in seconds - the time to perform one count of the transformation.	`2`
`delay`	integer	Time to wait before starting the translation, in seconds.	`0`

6.2 BPS

A BPS object is returned from *get_bps*.

6.2.1 Member functions

The following are member functions of BPS objects.

get_state

get_state(buttonNum)

Returns the state of the button with integer number `buttonNum`, which can be one of the constants `RELEASED`, `PRESSED`, `HELD` or `REPEAT`.

For example:

```
bps = get_bps(1)
btn = bps:get_state(1)
```

set_led

set_led(button, effect[, intensity[, fade]])

Set the effect and intensity of a BPS button LED according to the parameters:

Parameter	Value Type	Description	Value Example
`button`	integer (1-8)	Number of the BPS button to set an effect on	1
`effect`	integer	Integer value of constants: `OFF`, `ON`, `SLOW_FLASH`, `FAST_FLASH`, `DOUBLE_FLASH`, `BLINK`, `PULSE`, `SINGLE`, `RAMP_ON`, `RAMP_OFF`	`SLOW_FLASH`
`intensity`	integer (0-255)	Optional. Intensity level to set on the LED. If this parameter is not specified, full intensity will be set on the LED.	255
`fade`	float	Optional. Fade time to apply the override change, in seconds.	2.0

For example:

```
-- Set button 1 on BPS 1 to Fast Flash at full intensity
get_bps(1):set_led(1,FAST_FLASH,255)
```

6.3 Content Target

Note: Only supported on Atlas and Atlas Pro.

A `ContentTarget` object is returned from *get_content_target*.

6.3.1 Properties

Property	Value Type	Description
`master_intensity_level`	*Variant*	
`rotation_offset`	float	Atlas Pro only
`x_position_offset`	float	Atlas Pro only
`y_position_offset`	float	Atlas Pro only

For example, on a Atlas:

```
target = get_content_target(1)
current_level = target.master_intensity_level
```

And on a Atlas Pro:

```
target = get_content_target(1, PRIMARY)
current_angle = target.rotation_offset
```

6.3.2 Member functions

The following are member functions of `ContentTarget` objects.

set_master_intensity

set_master_intensity(level[, fade[, delay]])

Masters the intensity of the content target according to the parameters:

Parameter	Value Type	Description	Value Example
`level`	float (0.0-1.0) or integer (0-255)	Master level to set on the content target.	`0.5` or `128`
`fade`	float	Optional. Fade time to apply the intensity change, in seconds.	`2.0`
`delay`	float	Optional. Time to wait before applying the intensity change, in seconds.	`3.0`

For example, on a Atlas:

```
-- Master the primary content target in composition 1 to 50% (128/255 = 0.5) in 3 seconds
get_content_target(1):set_master_intensity(128,3)
```

Or on a Atlas Pro:

```
-- Master the secondary content target in composition 2 to 100% in 2.5 seconds
get_content_target(2, SECONDARY):set_master_intensity(255,2.5)
```

transition_rotation

Note: Only supported on Atlas Pro.

---

transition_rotation([angle[, count[, period[, delay[, useShortestPath]]]]])

Applies a rotation to the content target according to the parameters:

Parameter	Value Type	Description	Value Example
angle	float	Optional. Angle of rotation to transition to, in degrees. Defaults to zero.	90.0
count	integer	Number of times to repeat the rotation transformation.	1
period	integer	The period of the rotation, in seconds - the time to perform one count of the transformation.	2
delay	integer	Time to wait before starting the rotation, in seconds.	0

transition_y_position

transition_y_position([y_offset[, count[, period[, delay]]]])

Moves the content target along the y axis according to the parameters:

Parameter	Value Type	Description	Value Example
y_offset	float	Optional. Offset to apply to the y position. Defaults to 0.	25.0
count	integer	Number of times to repeat the y translation.	1
period	integer	The period of the translation, in seconds - the time to perform one count of the transformation.	2
delay	integer	Time to wait before starting the translation, in seconds.	0

6.4 Controller

A Controller object is returned from e.g. *get_current_controller*.

6.4.1 Properties

Property	Value Type	Description	Value Example
number	integer	Controller number	1
name	string	Controller name	"Controller 1"
vlan_tag	string	VLAN tag number as a string. "None" if there is no tag set	"65535"
is_network_primary	boolean	Whether this controller is set as the Network Primary in the project	true

For example:

```
cont = get_current_controller()
name = cont.name
```

6.5 DateTime

A `DateTime` object is returned from e.g. *System* properties.

6.5.1 Properties

Property	Value Type	Value Example
year	integer	2022
month	integer	12
monthday	integer	3
time_string	string	"11:35:32"
date_string	string	"03 Dec 2022"
weekday	integer (0 => Sunday)	0
hour	integer	11
minute	integer	35
second	integer	32
utc_timestamp	integer	1670045912

6.6 Group

A `Group` object is returned from *get_group*.

6.6.1 Properties

Property	Value Type	Description	Value Example
`name`	string	Group name	`"Group 1"`
`master_intensity_level`	*Variant*	The intensity level that this group is currently being mastered to	

For example:

```
grp = get_group(1)
name = grp.name
```

6.6.2 Member functions

The following are member functions of `Group` objects.

set_master_intensity

```
set_master_intensity(level[, fade[, delay]])
```

Masters the intensity of the group according to the parameters:

Parameter	Value Type	Description	Value Example
`level`	float (0.0-1.0) or integer (0-255)	Master level to set on the group	`0.5` or `128`
`fade`	float	Optional. Fade time to apply the intensity change, in seconds	`2.0`
`delay`	float	Optional. Time to wait before applying the intensity change, in seconds	`3.0`

For example:

```
-- Master group 1 to 50% (128/255 = 0.5) in 3 seconds
get_group(1):set_master_intensity(128,3)
```

6.7 Location

A `Location` object is returned from *get_location*.

6.7.1 Properties

Property	Value Type	Value Example
`lat`	float	`51.512`
`long`	float	`-0.303`

For example:

```
lat = get_location().lat
```

6.8 Override

An `Override` object is returned from *get_fixture_override* and *get_group_override*.

6.8.1 Member functions

The following are member functions of `Override` objects.

set_irgb

```
set_irgb(intensity, red, green, blue, [fade, [path]])
```

Overrides the intensity, red, green and blue levels for the fixture or group according to the parameters:

Parameter	Value Type	Description	Value Example
`intensity`	integer (0-255)	Intensity level to set as an override.	`128`
`red`	integer (0-255)	Red level to set as an override.	`128`
`green`	integer (0-255)	Green level to set as an override.	`128`
`blue`	integer (0-255)	Blue level to set as an override.	`128`
`fade`	float	Optional. Fade time to apply the override change, in seconds.	`2.0`
`path`	string	Optional. Crossfade path to use when applying the override: `Default`, `Linear`, `Start`, `End`, `Braked`, `Accelerated`, `Damped`, `Overshoot`, `Col At Start`, `Col At End`, `Int At Start`, `Int At End`, `Colour First`, `Intensity First`	`"Linear"`

For example:

```
-- Get override for fixture 22
override = get_fixture_override(22)
-- Set the override colour to red (and full intensity)
override:set_irgb(255, 255, 0, 0)
```

set_intensity

`set_intensity(intensity, [fade, [path]])`

Overrides the intensity level for the fixture or group according to the parameters:

Parameter	Value Type	Description	Value Example
`intensity`	integer (0-255)	Intensity level to set as an override.	`128`
`fade`	float	Optional. Fade time to apply the override change, in seconds.	`2.0`
`path`	string	Optional. Crossfade path to use when applying the override: `Default`, `Linear`, `Start`, `End`, `Braked`, `Accelerated`, `Damped`, `Overshoot`, `Col At Start`, `Col At End`, `Int At Start`, `Int At End`, `Colour First`, `Intensity First`	`"Linear"`

For example:

```
-- Get override for group 3
override = get_group_override(3)
-- Set the intensity to 50% in 2 seconds
override:set_intensity(128, 2.0)
```

set_red

`set_red(red, [fade, [path]])`

Overrides the red level for the fixture or group according to the parameters:

Parameter	Value Type	Description	Value Example
`red`	integer (0-255)	Red level to set as an override.	`128`
`fade`	float	Optional. Fade time to apply the override change, in seconds.	`2.0`
`path`	string	Optional. Crossfade path to use when applying the override: `Default`, `Linear`, `Start`, `End`, `Braked`, `Accelerated`, `Damped`, `Overshoot`, `Col At Start`, `Col At End`, `Int At Start`, `Int At End`, `Colour First`, `Intensity First`	`"Linear"`

set_green

`set_green(green, [fade, [path]])`

Overrides the green level for the fixture or group according to the parameters:

Parameter	Value Type	Description	Value Example
green	integer (0-255)	Green level to set as an override.	128
fade	float	Optional. Fade time to apply the override change, in seconds.	2.0
path	string	Optional. Crossfade path to use when applying the override: Default, Linear, Start, End, Braked, Accelerated, Damped, Overshoot, Col At Start, Col At End, Int At Start, Int At End, Colour First, Intensity First	"Linear"

set_blue

set_blue(blue, [fade, [path]])

Overrides the blue level for the fixture or group according to the parameters:

Parameter	Value Type	Description	Value Example
blue	integer (0-255)	Blue level to set as an override.	128
fade	float	Optional. Fade time to apply the override change, in seconds.	2.0
path	string	Optional. Crossfade path to use when applying the override: Default, Linear, Start, End, Braked, Accelerated, Damped, Overshoot, Col At Start, Col At End, Int At Start, Int At End, Colour First, Intensity First	"Linear"

set_temperature

set_temperature(temperature, [fade, [path]])

Overrides the temperature level for the fixture or group according to the parameters:

Parameter	Value Type	Description	Value Example
temperature	integer (0-255)	Temperature level to set as an override.	128
fade	float	Optional. Fade time to apply the override change, in seconds.	2.0
path	string	Optional. Crossfade path to use when applying the override: Default, Linear, Start, End, Braked, Accelerated, Damped, Overshoot, Col At Start, Col At End, Int At Start, Int At End, Colour First, Intensity First	"Linear"

clear

clear([fade])

Removes any override on the fixture or group. Optionally specify a `fade` time in seconds as a float, e.g. `2.0`.

For example:

```
-- Clear the override on fixture 1
get_fixture_override(1):clear()
```

See also: *clear_all_overrides*.

6.9 Project

A `Project` object is returned from *get_current_project*.

6.9.1 Properties

Property	Value Type	Value Example
name	string	"Help Project"
author	string	"Contoso"
filename	string	"help_project_v1.pd2"
unique_id	string	"{6b48627a-1d5e-4b2f-81e2-481e092a6a79}"

For example:

```
project_name = get_current_project().name
```

6.10 Network 2

Information about the controller's second network interface is available in the `protocol_interface` namespace. In trigger action scripts the `protocol_interface` namespace is added directly to the environment; in IO modules it is in the `controller` namespace, i.e. `controller.protocol_interface`.

6.10.1 Properties

The `protocol_interface` namespace has the following properties:

Property	Value Type	Value Example
has_interface	boolean	true
is_up	boolean	true
ip_address	string	"192.168.1.12"
subnet_mask	string	"255.255.255.0"
gateway	string	"192.168.1.1"

For example:

```
if protocol_interface.has_interface == true then
  ip = protocol_interface.ip_address
end
```

6.11 Replication

A `Replication` object is returned from *get_current_replication*.

6.11.1 Properties

Property	Value Type	Value Example
name	string	"Help Project"
unique_id	string	"{6b48627a-1d5e-4b2f-81e2-481e092a6a79}"

For example:

```
rep_name = get_current_replication().name
```

6.12 RIO

A `RIO` object is returned from *get_rio*.

For example:

```
rio = get_rio(RIO44, 1)
input = rio:get_input(1)
output_state = rio:get_output(1)
```

6.12.1 Member functions

The following are member functions of `RIO` objects.

get_input

`get_input(inputNum)`

Returns the state of the input with integer number `inputNum` as a boolean if the input is set to Digital or Contact Closure, or an integer if the input is set to Analog.

For example:

```
rio = get_rio(RIO44, 3)
input = rio:get_input(1)
```

get_output

`get_output(outputNum)`

Returns the state of the output with integer number `outputNum` as a boolean.

For example:

```
rio = get_rio(RIO44, 2)
output_state = rio:get_output(1)
```

set_output

`set_output(outputNum, state)`

Sets the output of a RIO to on or off according to the parameters:

Parameter	Value Type	Description	Value Example
`outputNum`	integer (1-8)	Number of the RIO output to change the state of. Range depends on type of RIO.	1
`state`	boolean or integer	State to set the output to. Can be any of: `0`, `1`, `true`, `false`, `ON` or `OFF`	OFF

6.13 Scene

A Scene object is returned from *get_scene*.

6.13.1 Properties

Property	Value Type	Description	Value Example
`name`	string	Scene name	`"Scene 1"`
`group`	string	Scene group name (A through H or empty string)	`"A"`
`state`	integer	Integer value of constants: `Scene.NONE`, `Scene.STARTED` or `Scene.RELEASED`	1
`onstage`	boolean	Whether the scene is affecting output of any fixtures	`false`
`custom_properties`	table	Table keys and values correspond to custom property names and values	

For example:

```
scn = get_scene(1)
name = scn.name
state = scn.state
```

6.13.2 Member functions

The following are member functions of `Scene` objects.

start

```
start()
```

Starts the scene. For example:

```
-- start scene 1
get_scene(1):start()
```

release

```
release([fade])
```

Releases the scene. Optionally specify a `fade` time in seconds as a float, e.g. `2.0`.

For example:

```
-- release scene 3 with a fade of 1 second
get_scene(3):release(1.0)
```

toggle

```
toggle([fade])
```

Toggles the playback of the scene - if it's running, release it; if it's not running, start it. Optionally specify a release `fade` time in seconds as a float, e.g. `2.0`.

For example:

```
-- toggle scene 2, releasing in time 3 secs if it's running
get_scene(2):release(3.0)
```

6.14 System

In trigger action scripts the `system` namespace is added directly to the environment; in IO modules it is in the `controller` namespace, i.e. `controller.system`.

6.14.1 Properties

The `system` namespace has the following properties:

Property	Value Type	Value Example
hardware_type	string	"lpc"
channel_capacity	integer	512
serial_number	string	"006321"
memory_total	string	"12790Kb"
memory_used	string	"24056Kb"
memory_available	string	"103884Kb"
storage_size	string	"1914MB"
bootloader_version	string	"0.9.0"
firmware_version	string	"2.8.0"
reset_reason	string	"Software Reset"
last_boot_time	*DateTime*	
ip_address	string	"192.168.1.3"
subnet_mask	string	"255.255.255.0"
broadcast_address	string	"192.168.1.255"
default_gateway	string	"192.168.1.3"
dns_servers	table of strings	"1.1.1.1","1.0.0.1"

For example:

```
capacity = system.channel_capacity

boot_time = system.last_boot_time.time_string
```

6.15 Temperature

A `Temperature` object is returned from *get_temperature*.

6.15.1 Properties

Property	Value Type	Description	Value Example
sys_temp	number	Only for MSC X and Atlas/Atlas Pro	40.2
core1_temp	number	Only for MSC X and Atlas/Atlas Pro	44
core2_temp	number	Only for MSC X rev 1	44.1
ambient_temp	number	Only for MTPC, MSC X rev 1	36.9
cc_temp	number	Only for MSC X rev 2 and Atlas/Atlas Pro	44.1
gpu_temp	number	Only for Atlas/Atlas Pro	38.2

For example:

```
temp = get_temperature()
log(temp.ambient_temp)
```

6.16 Time

Information about the controller's clock is available in the `time` namespace. In trigger action scripts the `time` namespace is added directly to the environment; in IO modules it is in the `controller` namespace, i.e. `controller.time`.

6.16.1 Properties

The `time` namespace has the following properties:

Property	Value Type	Value Example
`is_dst`	boolean	`true`
`gmt_offset`	integer (minutes)	`-300` 300 Minutes (5 hours) behind

6.16.2 Functions

The `time` namespace has the following functions, which each return a *DateTime* object:

- `get_current_time()`
- `get_sunrise()`
- `get_sunset()`
- `get_civil_dawn()`
- `get_civil_dusk()`
- `get_nautical_dawn()`
- `get_nautical_dusk()`
- `get_new_moon()`
- `get_first_quarter()`
- `get_full_moon()`
- `get_third_quarter()`

For example:

```
current_hour = time.get_current_time().hour
```

6.17 Timeline

A `Timeline` object is returned from *get_timeline*.

6.17.1 Properties

Property	Value Type	Description	Value Example
name	string	Timeline name	`"Timeline 1"`
group	string	Timeline group name (A through H or empty string)	`"A"`
length	integer	Timeline length, in milliseconds	`10000`
source_bus	integer	Integer value of constants: `DEFAULT`, `TCODE_1` … `TCODE_6`, `AUDIO_1` … `AUDIO_4`	`1`
timecode_format	string	Incoming timecode format on source bus	`"SMPTE30"`
audio_band	integer	0 is equivalent to the constant: `VOLUME`	`0`
audio_channel	integer	Integer value of constants: `LEFT`, `RIGHT` or `COMBINED`	`1`
audio_peak	boolean	The Peak setting of the timeline, if set to an audio time source	`false`
time_offset	integer	Milliseconds	`5000`
state	integer	Integer value of the state - see *Timeline States* below for definitions	`1`
onstage	boolean	Whether the timeline is affecting output of any fixtures	`true`
position	integer	Milliseconds	`5000`
priority	integer	Integer value of constants: `HIGH_PRIORITY`, `ABOVE_NORMAL_PRIORITY`, `NORMAL_PRIORITY`, `BELOW_NORMAL_PRIORITY` or `LOW_PRIORITY`	`0`
custom_properties	table	Table keys and values correspond to custom property names and values	

For example:

```
tl = get_timeline(1)
name = tl.name
state = tl.state

if (tl.source_bus == TCODE_1) then
  -- do something
end
```

Timeline States

A timeline will be in one of the following states:

- `Timeline.NONE`

  – The timeline has never been run (since the last reset of the controller).

- `Timeline.RUNNING`

  – The timeline is running (although might not be actively controlling outputs - see the `onstage` property).

- `Timeline.PAUSED`

  – The timeline has been paused by another action.

- `Timeline.HOLDING_AT_END`

  – The timeline has reached the end, and is holding.

- `Timeline.RELEASED`

  – The timeline has been run and has now been released.

6.17.2 Member functions

The following are member functions of `Timeline` objects.

start

`start()`

Starts the timeline. For example:

```
-- start timeline 1
get_timeline(1):start()
```

release

`release([fade])`

Releases the timeline. Optionally specify a `fade` time in seconds as a float, e.g. `2.0`.

For example:

```
-- release timeline 3
get_timeline(3):release(1.0)
```

toggle

```
toggle([fade])
```

Toggles the playback of the timeline - if it's running, release it; if it's not running, start it. Optionally specify a release fade time in seconds as a float, e.g. `2.0`.

For example:

```
-- toggle timeline 2, releasing in time 3 secs if it's running
get_timeline(2):release(3.0)
```

pause

```
pause()
```

Pauses the timeline.

resume

```
resume()
```

Resumes the timeline.

set_rate

```
set_rate(rate)
```

Sets the rate of playback of the timeline. Set the `rate` as a float or an integer with range, e.g. `0.1` or `Variant(10, 100)` would set the rate to 10% of normal speed.

For example:

```
-- set the rate of timeline 1 to 20% of normal speed
get_timeline(1):set_rate(0.2)
-- set the rate of timeline 2 to 30% of normal speed
get_timeline(2):set_rate(Variant(30,100))
```

set_position

```
set_position(position)
```

Jumps the position of playback of the timeline. Set the `position` as a float or an integer with range, e.g. `0.1` or `Variant(10, 100)` would set the position to 10% of the timeline length.

For example:

```
-- set the position of timeline 1 to 50% of timeline length
get_timeline(1):set_position(0.5)
-- set the position of timeline 2 to 20% of timeline length
get_timeline(2):set_position(Variant(2,10))
```

set_default_source

Set the time source for the timeline to the default.

For example:

```
get_timeline(1):set_default_source()
```

set_timecode_source

```
set_timecode_source(timecodeBus[, offset])
```

Set a timecode source for the timeline according to the parameters:

Parameter	Value Type	Description	Value Example
`timecodeBus`	integer	Integer value of constants: `TCODE_1` ... `TCODE_6`	`TCODE_1`
`offset`	integer	Optional offset to apply to the timecode, in milliseconds	`1000`

set_audio_source

```
set_audio_source(audioBus, band, channel[, peak])
```

Set a audio band as the time source for the timeline according to the parameters:

Parameter	Value Type	Description	Value Example
`audioBus`	integer	Integer value of constants: `AUDIO_1` ... `AUDIO_4`	`AUDIO_1`
`band`	integer	The audio band to sample (number of bands depends on audio source configuration; 0 => volume)	`0`
`channel`	integer	Integer value of constants: `LEFT`, `RIGHT` or `COMBINED`	`LEFT`
`peak`	boolean	Optional. Whether to use the peak levels from the audio band as the time source input (default false)	`false`

6.18 Universe

A `Universe` object is returned from e.g. *get_dmx_universe*.

6.18.1 Member functions

The following are member functions of `Universe` objects.

get_channel_value

`get_channel_value(channel)`

Gets the current level of a channel in the universe, where `channel` is the integer channel number (1-512).

For example:

```
uni = get_dmx_universe(1) -- get DMX Universe 1
level = uni:get_channel_value(1) -- get channel 1 from the returned universe
```

park

`park(channel, value)`

Parks an output channel at a given value according to the parameters:

Parameter	Value Type	Description	Value Example
`channel`	integer (1-512)	Number of the output channel	1
`value`	integer (0-255)	Level to set the channel to	128

For example:

```
-- Park channel 4 of DMX universe 1 at 128 (50%)
get_dmx_universe(1):park(4,128)
```

unpark

`unpark(channel)`

Clears the parked value on an output channel, where `channel` is the integer channel number (1-512).

For example:

```
 -- Unpark channel 4 of DMX universe 1
 -- (it will go back to normal output levels)
get_dmx_universe(1):unpark(4)
```

6.19 Variant

6.19.1 Introduction

Within Lua Scripting (as with other scripting languages) it is possible to store data within a named location (variable).

Lua typically doesn't differentiate between the contents of a variable (unlike some programming languages) and the type (integer, string, boolean) of the variable can change at any time.

Mosaic has added an object to the scripting environment called a `Variant`, which can be used to contain the data with an assignment as to the type of data that is contained. This means that a single `Variant` can be utilised and handled differently depending on the data that is contained and how it is being used.

6.19.2 Definition

Properties

A `Variant` object has the following properties:

Property	Description
`integer`	Get or set an integer data type
`range`	Get or set the range of an integer data type
`real`	Get or set a real data type (number with decimal point)
`string`	Get or set a string data type
`ip_address`	Get or set an IP address data type

Member functions

Constructor

`Variant()`

Create new variant.

is_integer

Returns `true` or `false` to show whether the stored data has an integer representation.

is_string

Returns `true` or `false` to show whether the stored data has a string representation.

is_ip_address

Returns `true` or `false` to show whether the stored data has an IP address representation.

6.19.3 Usage

```
Variant(value, range)
```

Defining a variant

Within your Lua script you can create a Variant with the following syntax:

```
var = Variant() -- where var is the name of the variant.
```

Variant types

Integer

An integer variant can be used to store a whole number:

```
var = Variant() -- where var is the name of the variant

var.integer = 123 -- set var to an integer value of 123

log(var.integer) -- get the integer value stored in var

log(var.real) -- get the integer value stored in var and convert it to a float

log(var.string) -- get the integer value stored in var and convert it to a string
```

As shown in the example code, above, the `integer` property of a `Variant` can be used to either get or set the value of the `Variant` as an integer (whole number).

```
var:is_integer() -- returns a boolean if the variant contains an integer
```

Range

An integer can be stored with an optional range parameter:

```
var = Variant() -- where var is the name of the variant

var.integer = 123 -- set var to an integer value of 123

var.range = 255 -- set the range of var to be 255
```

This can be used to calculate fractions and/or to define that a `Variant` is a 0-1, 0-100 or 0-255 value.

The range of a `Variant` should be set if you intend to use the `Variant` to set an intensity or colour value.

Some captured variables have a range attribute, and this is indicated in the log like this:

```
Trigger 7 (Ethernet Input): Captured 3 variables
Captured variables
   1 - Integer: 100 of 255
```

Real

A real `Variant` can be used to store a floating point (decimal) number.

```
var = Variant() -- where var is the name of the variant.

var.real = 12.3 -- set var to an integer value of 12.3

log(var.real) -- get the integer value stored in var
```

As shown in the example code, above, the `real` property of a `Variant` can be used to either get or set the value of the `Variant` as a real number.

String

A string `Variant` can be used to store a string of ASCII characters.

```
var = Variant() -- where var is the name of the variant

var.string = "example" -- set var to a string value of "example"

log(var.string) -- get the string value stored in var
```

As shown in the example code, above, the `string` property of a `Variant` can be used to either get or set the value of the `Variant` as a string.

```
var:is_string() -- returns a boolean if the variant contains a string
```

IP address

```
var = Variant() -- where var is the name of the variant

var.ip_address = "192.168.1.23" -- set var to the IP Address 192.168.1.23 or -1062731497

log(var) -- get the stored data ("192.168.1.23")

log(var.ip_address) -- get the stored IP Address (-1062731497)

log(var.string) -- get the stored IP Address and convert it to a string ("192.168.1.23")

log(var.integer) -- get the stored IP Address and convert it to an integer (-1062731497)
```

As shown in the example code, above, the `ip_address` property of a `Variant` can be used to either get or set the value of the `Variant` as an IP Address.

As a setter, you can pass a dotted decimal string (e.g. "192.168.1.23" or the integer representation -1062731497).

```
var:is_ip_address() -- returns a boolean if the variant contains a IP Address
```

Shorthand

A `Variant` can also be defined using a shorthand:

```
var = Variant(128,255) -- create variable var as an integer (128) with range 0-255

var = Variant(128) -- create variable var as a real number (128.0)

var = Variant(12.3) -- create variable var as a real number (12.3)

var = Variant("text") -- create variable var as a string ("text")
```

Note: There isn't a shorthand for IP Addresses.

6.19.4 Default variants

Some script functions return a `Variant`, including *get_trigger_variable*. For example:

```
get_trigger_variable(1).integer
```

The `master_intensity_level` properties of *Group* and *Content Target* are both Variants:

```
get_group(1).master_intensity_level.integer

get_group(1).master_intensity_level.range

get_content_target(1).master_intensity_level.integer

get_content_target(1).master_intensity_level.range
```

6.20 Standard Libraries

The following standard Libraries are imported

- Basic library
- Package library
- String manipulation
- Basic UTF-8 support
- Table manipulation
- Mathematical functions
- Input and output

6.21 Functions

The following functions are available in trigger action scripts and in IO modules. In trigger action scripts they are added directly to the environment; in IO modules they are available in the `controller` namespace.

6.21.1 Queries

get_current_project

Returns a *Project* object.

For example:

```
project_name = get_current_project().name
```

get_current_replication

Returns a *Replication* object.

For example:

```
rep_name = get_current_replication().name
```

get_location

Returns a *Location* object.

For example:

```
lat = get_location().lat
```

get_timeline

get_timeline(timelineNum)

Returns a single *Timeline* object for the timeline with user number `timelineNum`.

For example:

```
tl = get_timeline(1)
name = tl.name
state = tl.state

if (tl.source_bus == TCODE_1) then
  -- do something
end
```

get_scene

get_scene(sceneNum)

Returns a single *Scene* object for the scene with user number sceneNum.

For example:

```
scn = get_scene(1)
name = scn.name
state = scn.state
```

get_group

get_group(groupNum)

Returns a single *Group* object for the group with user number groupNum.

For example:

```
grp = get_group(1)
name = grp.name
```

Note: Passing 0 as groupNum will return *Group* for the *All Fixtures* group. This can also be used on Atlas family projects to master the intensity of the entire unit.

---

get_fixture_override

get_fixture_override(fixtureNum)

Returns an *Override* object for the fixture with user number fixtureNum.

For example:

```
-- Get override for fixture 22
override = get_fixture_override(22)
-- Set the override colour to red (and full intensity)
override:set_irgb(255, 255, 0, 0)
```

get_group_override

get_group_override(groupNum)

Returns an *Override* object for the group with user number groupNum.

Note: Passing 0 as groupNum will return an *Override* for the *All Fixtures* group.

For example:

```
-- Get override for group 3
override = get_group_override(3)
-- Set the intensity to 50% in 2 seconds
override:set_intensity(128, 2.0)
```

get_current_controller

Returns the *Controller* that the script is being executed on.

For example:

```
cont = get_current_controller()
name = cont.name
```

get_network_primary

Returns the *Controller* in the project that is set as the *network primary*.

is_controller_online

is_controller_online(controllerNum)

Returns true if the controller with user number `controllerNum` has been discovered, or false otherwise.

For example:

```
if (is_controller_online(2)) then
  log("Controller 2 is online")
else
  log("Controller 2 is offline")
end
```

get_temperature

Returns a *Temperature* object with measurements from the controller's temperature sensors.

For example:

```
temp = get_temperature()
log(temp.ambient_temp)
```

get_rio

get_rio(type, num)

Returns a *RIO* object representing a RIO matching the parameters:

- `type` can be one of the constants RIO80, RIO44 or RIO80.

- `num` is the remote device number within the Designer project.

For example:

```
rio = get_rio(RIO44, 1)
input = rio:get_input(1)
output_state = rio:get_output(1)
```

Note: The constants for `type` are in the `controller` namespace within IO modules, e.g. `controller.RIO44`.

---

get_bps

get_bps(num)

Returns a *BPS* object with remote device number `num`.

For example:

```
bps = get_bps(1)
btn = bps:get_state(1)
```

get_text_slot

get_text_slot(slotName)

Returns the value of the text slot with name `slotName`. If no such text slot exists in the project then an empty string will be returned.

For example:

```
log(get_text_slot("my text slot"))
```

get_dmx_universe

get_dmx_universe(idx)

Returns a *Universe* object for the DMX universe with number `idx`.

For example:

```
uni = get_dmx_universe(1) -- get DMX Universe 1
level = uni:get_channel_value(1) -- get channel 1 from the returned universe
```

get_artnet_universe

get_artnet_universe(idx)

Returns a *Universe* object for the Art-Net universe with number `idx`.

get_pathport_universe

get_pathport_universe(idx)

Returns a *Universe* object for the Pathport universe with number `idx`.

get_sacn_universe

get_sacn_universe(idx)

Returns a *Universe* object for the sACN universe with number `idx`.

get_kinet_universe

get_kinet_universe(power_supply_num, port_num)

Returns a *Universe* object for the KiNET power supply port matching the parameters:

- `power_supply_num` is the KiNET power supply number in the project.

- `port_num` is the port number of the KiNET power supply.

get_edn_universe

get_edn_universe(remote_device_type, remote_device_num, port_num)

Returns a *Universe* object for the EDN output matching the parameter:

- `remote_device_type` is be one of the constants `EDN10` or `EDN 20`.

- `remote_device_num` is the remote device number of the EDN in the project.

- `port_num` is the DMX output port number of the EDN.

Note: The constants for `remote_device_type` are in the `controller` namespace within IO modules, e.g. `controller.EDN20`.

---

get_input

get_input(idx)

Returns the state of the controller's input numbered `idx` as a boolean (for digital inputs) or an integer (for analog inputs, 0-100).

For example:

```lua
in1 = get_input(1)

if in1 == true then
  log("Input 1 is digital and high")
elseif in1 == false then
  log("Input 1 is digital and low")
else
  log("Input 1 is analog at " .. in1)
end
```

get_dmx_input

`get_dmx_input(channel)`

Returns the value of the DMX `channel` number as an integer. If no DXM input is detected then `nil` will be returned.

get_trigger_variable

`get_trigger_variable(idx)`

Returns the trigger variable at index `idx` as a *Variant*.

For example:

```lua
-- Use with a Touch Colour Move Trigger
red = get_trigger_variable(1).integer
green = get_trigger_variable(2).integer
blue = get_trigger_variable(3).integer

-- Use with Serial Input "<s>\r\n"
input = get_trigger_variable(1).string
```

get_trigger_number

`get_trigger_number()`

Returns the number of the trigger that ran this script. Will return `nil` if called from another context.

get_resource_path

`get_resource_path(filename)`

Returns the path to the resource file, where `filename` is the name of a file on the controller's internal storage.

For example:

```lua
dofile(get_resource_path("my_lua_file.lua"))
```

get_content_target

Note: Only supported on Atlas and Atlas Pro.

On a Atlas: `get_content_target(compositionNum)`

On a Atlas Pro: `get_content_target(compositionNum, type)`

Returns a *Content Target* object representing the Content Target in the project that matches the parameters:

- `compositionNum` is the user number of the composition containing the desired Content Target.
- `type` describes the Content Target type and can be one of the constants `PRIMARY`, `SECONDARY` or `TARGET_3` ... `TARGET_8`.

Note: The constants for `type` are in the `controller` namespace within IO modules, e.g. `controller.TARGET_5`.

Will return `nil` if no matching Content Target exists in the project.

For example, on a Atlas:

```
target = get_content_target(1)
current_level = target.master_intensity_level
```

And on a Atlas Pro:

```
target = get_content_target(1, PRIMARY)
current_angle = target.rotation_offset
```

get_adjustment

Note: Only supported on Atlas Pro.

`get_adjustment(num)`

Returns an *Adjustment Target* object representing the Adjustment Target in the project with the integer user number `num`:

Will return `nil` if no matching Adjustment Target exists in the project.

For example:

```
target = get_adjustment(1)
target:transition_x_position(10,1,5) -- Move 10 pixels right in 5 seconds
target:transition_y_position(10,1,5) -- Move 10 pixels down in 5 seconds
target:transition_rotation(90,1,5) -- Rotate by 90 degrees in 5 seconds
```

get_log_level

Returns the current log level of the controller, which can be one of the following constants:

- `LOG_DEBUG`
- `LOG_TERSE`
- `LOG_NORMAL`
- `LOG_EXTENDED`
- `LOG_VERBOSE`
- `LOG_CRITICAL`

Note: These constants are in the `controller` namespace within IO modules, e.g. `controller.LOG_NORMAL`.

get_syslog_enabled

Returns true if Syslog is enabled, or false otherwise.

get_syslog_ip_address

Returns the IP address of the Syslog server as a string.

get_ntp_enabled

Returns true if NTP is enabled.

get_ntp_ip_address

Returns the IP address of the NTP server as a string.

6.21.2 Actions

log

```
log([level, ]message)
```

Write a message to the controller's log according to the parameters:

Parameter	Value Type	Description	Value Example
`level`	Integer value of constants: `LOG_DEBUG`, `LOG_TERSE`, `LOG_NORMAL`, `LOG_EXTENDED`, `LOG_VERBOSE`, `LOG_CRITICAL`; defaults to `LOG_NORMAL`	Optional. The log level to apply to the message.	`LOG_VERBOSE`
`message`	string	The message to add to the log.	`"Your log message"`

For example:

```
log(LOG_CRITICAL, "This is a critical message!") -- logs a message at Critical log level
log("This is a normal message.") -- logs a message at Normal log level.
```

set_log_level

`set_log_level(log_level)`

Changes the log level of the controller, showing more or less detailed information, where `log_level` is an integer value of the constants:

- `LOG_DEBUG` (5)
- `LOG_TERSE` (4)
- `LOG_NORMAL` (3)
- `LOG_EXTENDED` (2)
- `LOG_VERBOSE` (1)
- `LOG_CRITICAL` (0)

pause_all

Pause all timelines in the project.

resume_all

Resume all timelines in the project.

release_all

`release_all([fade,] [group])`

Release all timelines and scenes in the project.

Note:

You can provide:

- No arguments - this will release all with the default fade time.

- A fade time, which will be used to release all.

- Or, both a fade time and a group.

Parameter	Value Type	Description	Value Example
`fade`	float	Optional. Release fade time in seconds. If not provided, the default fade time will be used.	`2.0`
`group`	string	Optional. Group name: `A` through `H`. Prepend the group name with `!` to apply the action to all groups *except* the specified group, e.g. `!A`.	`"B"`

release_all_timelines

`release_all_timelines([fade,] [group])`

Release all timelines in the project.

Note:

You can provide:

> - No arguments - this will release all with the default fade time.
>
> - A fade time, which will be used to release all.
>
> - Or, both a fade time and a group.

Parameter	Value Type	Description	Value Example
`fade`	float	Optional. Release fade time in seconds. If not provided, the default fade time will be used.	`2.0`
`group`	string	Optional. Group name: `A` through `H`. Prepend the group name with `!` to apply the action to all groups *except* the specified group, e.g. `!A`.	`"B"`

release_all_scenes

`release_all_scenes([fade,] [group])`

Release all scenes in the project.

Note:

You can provide:

> - No arguments - this will release all with the default fade time.
>
> - A fade time, which will be used to release all.
>
> - Or, both a fade time and a group.

Parameter	Value Type	Description	Value Example
`fade`	float	Optional. Release fade time in seconds. If not provided, the default fade time will be used.	`2.0`
`group`	string	Optional. Group name: `A` through `H`. Prepend the group name with `!` to apply the action to all groups *except* the specified group, e.g. `!A`.	`"B"`

clear_all_overrides

```
clear_all_overrides([fade])
```

Removes all overrides from all fixtures and groups. Optionally specify a `fade` time in seconds as a float, e.g. `2.0`.

enqueue_trigger

```
enqueue_trigger(num[,var...])
```

Queue trigger number `num` to be fired on the next controller playback refresh. The trigger's conditions will be tested. Optional variables `var` can be passed in as additional arguments.

For example:

```
-- enqueue trigger 2, passing in three variables: 255, 4.0 and "string"
enqueue_trigger(2,255,4.0,"string")
```

enqueue_local_trigger

```
enqueue_local_trigger(num[,var...])
```

Same behaviour as for *enqueue_trigger* but the trigger `num` will only be queued on the controller that ran the function - the trigger will not propagate to other controllers in the project.

force_trigger

```
force_trigger(num[,var...])
```

Queue trigger number `num` to be fired on the next controller playback refresh without testing the trigger's conditions - the trigger actions will always run. Optional variables `var` can be passed in as additional arguments.

For example:

```
-- force the execution of trigger 2's actions
-- pass in three variables: 255, 4.0 and "string"
force_trigger(2,255,4.0,"string")
```

force_local_trigger

```
force_local_trigger(num[,var...])
```

Same behaviour as for *force_trigger* but the trigger `num` will only be queued on the controller that ran the function - the trigger will not propagate to other controllers in the project.

set_text_slot

set_text_slot(name, value)

Set the value of the text slot named `name` in the project to `value`, for example:

```
-- Set "My slot" to value "Hello world!"
set_text_slot("My slot", "Hello world!")
```

set_control_value

set_control_value(name, [index,] value[, emitChange])

Set the value on a Touch Slider or Colour Picker according to the parameters:

Parameter	Value Type	Description	Value Example
name	string	The Key of the Touch Control.	slider001
index	integer (1-3)	Optional. Axis of movement - a slider has 1; a colour picker has 3. Will default to 1 if this parameter isn't specified.	1
value	integer (0-255)	New value to set.	128
emitChange	boolean	Optional. Whether to fire associated triggers as a result of the control value change. Defaults to `false`.	true

For example:

```
-- Set slider001 to half (and don't fire any associated triggers)
set_control_value("slider001", 128)
-- Set the second axis (green) to full on colour020
set_control_value("colour020", 2, 255)
```

set_control_state

set_control_state(name, state)

Set the state on a Touch control according to the parameters:

Parameter	Value Type	Description	Value Example
name	string	The Key of the Touch Control.	slider001
state	string	The name of the state as defined in the Touch theme.	Green

For example:

```
-- Set slider001 to a state called "Green"
set_control_state("slider001", "Green")
```

set_control_caption

```
set_control_caption(name, caption)
```

Set the caption on a Touch control according to the parameters:

Parameter	Value Type	Description	Value Example
`name`	string	The Key of the Touch Control.	`button001`
`caption`	string	The text to display as the control's caption.	On

For example:

```lua
-- Set button001's caption to "On"
set_control_caption("button001", "On")
```

set_interface_page

```
set_interface_page(number[, transition])
```

Change the current page on the Touch interface according to the parameters:

Parameter	Value Type	Description	Value Example
`number`	integer	Touch interface page to change to.	2
`transition`	integer	Optional page transition. Integer value of constants: `SNAP`, `PAN_LEFT`, `PAN_RIGHT`	`PAN_LEFT`

Note: Must be executed on the MTPC that hosts the interface.

For example:

```lua
-- Change the touch screen interface to page 4 with a snap transition
set_interface_page(4, SNAP)
```

set_interface_enabled

```
set_interface_enabled([enabled])
```

Enable/disable the touchscreen, according to the optional boolean parameter `enabled` (default: `true`).

Note: Must be executed on the MTPC that hosts the interface.

For example:

```lua
-- Disable the touchscreen
set_interface_enabled(false)
```

set_interface_locked

set_interface_locked([lock])

Lock/unlock the touchscreen, according to the optional boolean parameter `lock` (default: `true`).

Note: Must be executed on the MTPC that hosts the interface.

For example:

```
-- Lock the touchscreen
set_interface_locked()
-- Unlock the touchscreen
set_interface_locked(false)
```

push_to_web

push_to_web(name, value)

Sends data as JSON to clients who are subscribed to the relevant websocket channel, e.g. custom web interfaces using *subscribe_lua* in the `query.js` library. The parameters are as follows:

Parameter	Value Type	Description	Value Example
`name`	string	JSON attribute name	`"myVar"`
`value`	*Variant*	Value for the JSON, which will be sent as a string.	`"String value"` or 1234

For example:

```
myData = 1234
-- Will push JSON object {"my_data":"1234"}
push_to_web("my_data", myData)
```

disable_output

disable_output(protocol)

Disables the output of a single protocol from the controller, where `protocol` is the integer value of the constants:

- `DMX`
- `PATHPORT`
- `ARTNET`
- `KINET`
- `SACN`
- `DVI`
- `RIO_DMX`
- `EDN_DMX`
- `EDN_SPI`

For example:

```
-- Disable the DMX output from the controller
disable_output(DMX)
```

enable_output

```
enable_output(protocol)
```

Enables the output of a single protocol from the controller, where `protocol` is the integer value of the constants defined for *disable_output*.

For example:

```
-- Enable the DMX output from the controller
enable_output(DMX)
```

set_timecode_bus_enabled

```
set_timecode_bus_enabled(bus[, enable])
```

Enable or disable a timecode bus, where bus is the integer value of the constants TCODE_1 ... TCODE_6 and `enable` is a boolean, determining whether the bus is enabled (default `true`) or not.