Pharos Designer User Manual

v2.14

Contents

Welcome

Introduction

Welcome and thank you for using version v2.14 of the Pharos Designer software. This release offers some significant improvements and additional functionality over earlier versions, see New In for details.

WARNING: Projects saved with v2.14 can not be opened with earlier versions so please make sure to back up your programming prior to installation.

Help Overview

The Help is split into five main sections: Quick Start, Hardware, Reference, Troubleshooting, and Appendices.

Those of you experimenting with the software for the first time are advised to work through the Quick Start guide to familiarise yourself with the basics of the software. The Reference section gives detailed descriptions of every aspect of the software as well as the configuration of the Pharos Controllers and their accessories. The Troubleshooting section provides help to resolve any problems while the Appendices provide additional useful resources.

If you have a Controller that you wish to connect to and program now then please read the Network section for instructions or follow the Quick Start guide.

Help

This is the PDF version of the on-line Help and it is available in various formats for printing. The on-line version, which has the advantage of being fully searchable and includes animated tutorials, can be opened from within Designer using Help > Contents on the main toolbar.

Support

As with all successful control products, support is crucial and the team at Pharos will do everything possible to ensure that your project is a success.

Please do not hesitate to contact us with your questions, bug reports and suggestions at:

T: +44-(0)20-7471-9449
E: support@pharoscontrols.com

Please also visit our website to keep up to date with the latest product news and software releases: www.pharoscontrols.com.

New in v2.14

RIO D4: a new four-port DALI remote device.

- Each RIO D4 supports four DALI buses, which can be used as outputs for control and for receiving DALI commands as triggers.
- Support for up to 64 DALI ballasts per bus, including discovery, configuration and command.
- Each device can be placed where it is needed and connected to a Pharos Designer controller over an Ethernet network.

New DALI features: for use in conjunction with RIO D4.

- Triggers for DALI Button Event, DALI Illuminance and DALI Occupancy.
- New Set DALI Feedback State Action.

Changes to touch device communication: touch devices are now remote devices.

- TPS, TPS 5 and TPS 8 are categorised as remote devices.
- Touch interfaces are sent from the controller to the touch device, rather than uploaded directly from Pharos Designer.
- Touch Triggers, Conditions and Actions allow you to specify a particular device.
- Please see this article for more information.

Pharos Designer Enhancements: We continue to make improvements throughout the software. Latest changes include:

- Specify playback group in timeline and scene triggers.
- Give timeline presets custom labels.
- View time at end of timeline preset by holding shift while moving the preset.
- Specify a fade time in the Set Screen Brightness action.
- Populate group name with model of first fixture in selection.
- Fixture tooltip settings are now saved in project files.
- The following system limits have been increased:
    - Up to 1000 timelines per project.
    - Up to 1000 scenes per project.
    - Up to 2000 fixture groups per project.
    - TPC, LPC 1, LPC 2 and LPC 4 now support up to 64 remote devices.

Introduction

Pharos is a comprehensive system with sophisticated features that allow you to make advanced shows. The Designer 2 software is the tool provided to configure and program the Pharos Controllers, Remote Devices and Accessories. The Controllers have been designed specifically for the architectural and installation markets and, as opposed to DMX frame store solutions, offer genuine lighting and show control functionality in an install and forget housing.

Lighting is programmed on timelines, with a particular timeline having control data for one, some or all the lighting fixtures being controlled. Multiple timelines are supported and so a single unit can control multiple distinct zones, or more complex presentations can be programmed with external triggers coming from multiple systems.

The software offers powerful functionality with a simple and intuitive graphical user interface. Most operations can be performed with mouse clicks (typically left-click for selection and right-click for context sensitive options & commands) and drag-and-drop. Creating a project is broken down into steps that all have their own tab for an easy step by step process to setup the system.

Modes Overview

Mode tabs down the left hand side allow you to switch between the Modes by left clicking the tab or use the function keys (in brackets) to toggle between them. Tabs can be viewed in different windows using the Tear Off options:

NOTE: Not all modes will be available by default, but will be enabled when relevant. See Project Features for more details

Project (F1)

In Project, you setup the Project settings such as location and time information, along with the project features that will be made available.

See Project for reference.

Layout (F2)

In Layout you add fixture to the project, position them on layouts, arrange them in groups or customize their behaviour.

See Layout for reference.

Mapping (F3)

Mapping allows you to create virtual video screens and map fixtures to pixels of the screen. Here you also import and manage the media files (either static images or video) which can then be played back on these screens.

See Mapping for reference.

Patch (F4)

In Patch the fixtures are assigned to the connected Controllers (see Network) and assigned to control protocols, universes and addresses. This step can be skipped during design and only completed during installation.

See Patch for reference.

DALI (F5)

In DALI you patch and define DALI groups & scenes for any DALI ballasts in the project. Unlike DMX fixtures, these definitions are stored in the DALI ballasts themselves and so the configuration must be uploaded separately from here.

Only available if Enabled in project properties, a DALI ballast is added to a layout, or a RIO D or RIO D4 is added to the project.

See DALI for reference.

Scene (F6)

In Scene, you can create single effects on any fixture within the project. These Scenes can be used later within Timelines or played back individually using triggering.

See Scene for reference.

Timeline (F7)

Timeline is where you create and edit the timelines that make up your presentation. Each fixture or group of fixtures is a row of the timeline and you can drag-and-drop from an extensive range of Built-in intensity and colour effects, as well as placing Scenes, Media and DALI presets.

See Timeline for reference.

Interface (F8)

Interface allows you to create the Interfaces that are displayed on a TPC.

Only available if Enabled in project properties, or a TPC is added to the project.

See Interface for reference.

Trigger (F9)

In Trigger you connect your programming with the real world. At its most basic you can define which timeline to begin on startup but for more complex environments with external triggers you can define a detailed script, even incorporating conditions if necessary.

See Trigger for reference.

Simulate (F10)

Simulate allows you to view a representation of your project in Layout format. You can play individual timelines to check your programming then run the whole project including triggers. A set of buttons allow you to simulate external triggers in order to test your programming properly.

See Simulate for reference.

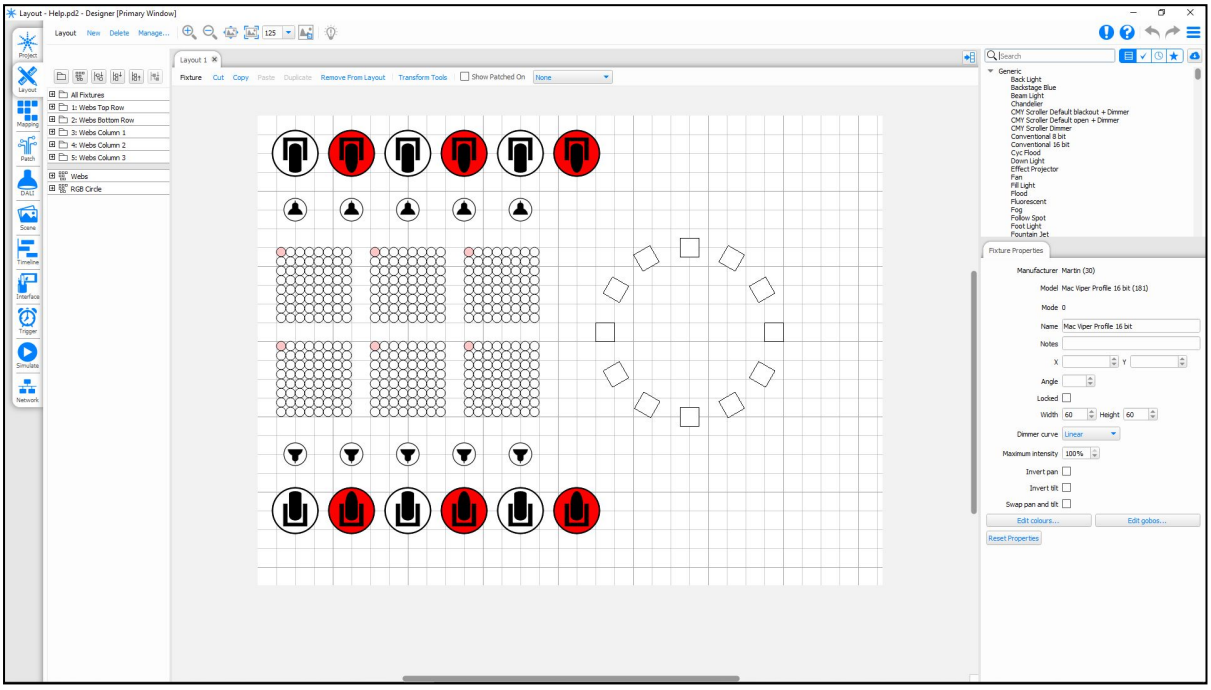
Network (F11)

This is where you manage your Pharos hardware, assigning connected Real Controllers to the Controllers in your project, configuring their input/output interfaces and any connected Remote Devices.

See Network for reference.

User Interface

The software has been designed to present a consistent graphical user interface and so it is worth familiarising yourself with the layout of a typical window before proceeding further:



Main Toolbar

The main toolbar persists across all software modes and allows you to see any Issues in your project, access the Undo and Redo buttons and provides access to the Main Menu.

Issues

The Issues browser will display any problems with the project file and take you to the location where the issue can be fixed. The location of an issue is indicated within the software with the Issues icon .

Help

The Software Help can be accessed using the Help icon on the main toolbar.

Undo/ Redo

The Undo and Redo buttons can be used to step through the last 20 actions completed within Pharos Designer.

Main Menu

The Main Menu allows access to the following options:

- Save Project - Save the show file with the current file name
- Save Project As - Save a copy of the project file with a new name
- Archive Project - Save a copy of the project file as a .archive.pd2 file
- Upload
- Audio Viewer
- Timecode Viewer
- Output Viewer
- Controller Log Viewer
- Import Object - Import fixture layouts, Pixel matrices etc.
- Import Interface - Import a .ptc file created in Interface Editor
- Export Object - Export an object as a .csv file
- Preferences - Access the preferences dialog
- Register - Register your copy of PharosDesigner
- Help - Access this help file
- Send Feedback - Contact Pharos Support with feedback on Designer 2
- About
- Exit - Close PharosDesigner

Mode Tabs

The application is divided into eleven Modes which can be selected by clicking on the appropriate tab.

See the Quick Start overview for a brief description of each Mode and the relevant Reference section for more details.

Tear Off Views 

All tabs can be torn off so that multiple Modes can be seen at once. This is particularly useful for the Timeline and Simulate Modes.

To tear of a tab either:

- Hover over the view tab until the tear off icon appears, move your cursor over this icon and left-click. The selected Mode will appear in a separate window.
- Click on a Mode tab and drag off the right side of the Mode tab bar. The selected Mode will appear in a second window.

NOTE: The currently open Mode cannot be torn off

Mode Toolbar

The view toolbar is populated with tools and options relevant to the Mode which is being worked in. See the relevant Reference section for more details.

Browser

The browser is common to many views and provides the primary interface for selecting, viewing and grouping fixtures in the project. The rows of the browser are then used for Designer's timeline programming interface. Some Modes (Project, Interface, Trigger, Simulate and Network) have no browser since fixture selection is not relevant. Scroll bars will appear as required and the browser can be made wider by dragging the right hand border.

Browser toolbar

The Browser toolbar provides controls for expanding and collapsing groups and compound fixtures as well as for creating Groups and Pixel Matrices.

Object Tabs

Most of the Modes within Designer have multiple objects that can be opened at once, such as Layouts, Timelines and Interfaces.

The exceptions are Network and Trigger, which affect the whole project and don't have separate objects within them.

The tabs can be navigated by selecting them in the Tab bar. They can be closed using the close button within the tab. This doesn't delete the object, it can be reopened from the Manage button on the Mode Toolbar.

Main Workspace

The Main workspace is the central portion of the Designer window and is where most project work is carried out. Each view uses the main workspace in a different way, so see the relevant Reference section for more detail.

Configuration Area

Depending on the Mode and items selected, context-sensitive configuration or control pane(s) will appear here for fast and convenient editing.

Multiple Instances

It is possible to open multiple instances of Designer so that two projects can be worked on at the same time.

This can be done by opening a Designer 2 project from the operating system when Designer 2 is already open, or by selecting to open Designer 2 again.

If necessary one of these instances can be Designer v1.x.x.

NOTE: You cannot open the same project in multiple instances of Designer 2.

Copying between Instances

When you have multiple instances of Designer 2 open, you can copy elements from one project to the other:

- Fixtures
- Timeline Presets

Keyboard Shortcuts

For ease and speed of use various keyboard keys map to application commands, particularly with regards window navigation:

General

Ctrl+Z	Undo the last action (up to 20 actions)
Ctrl+Y	Redo the last undone action
Ctrl+Tab (⌥ + Tab)	Switch to the next tab
Ctrl+Shift+Tab (⌥ + Shift + Tab)	Switch to the previous tab
Ctrl+S	Save the project
Ctrl+Shift+S	Save the Project as New
Ctrl+U	Upload the project to controllers
Alt+F4	Quit the application
Escape	Close an open popover
Function Keys - F1 through F11	Change view, F1 goes to Project, F11 goes to Network etc.
Alt + F4 (⌘ +Q)	Close Designer
(⌘ +,)	Access Preferences
Ctr+T	Open (or Close) the Output Viewer

Project

Ctrl+N	Create a new project
Alt+left-click on New Project	Show New Project wizard
Ctrl+O	Open a project
Ctrl+F4 (⌘ +W)	Close the project

Layout

Ctrl+N	Create a New Layout
Ctrl+D	Create a duplicate of the current layout
Ctrl+I	Show layout properties
Ctrl+A	Select all fixtures
Double-left-click *on a fixture*	Select all instances of the fixture
Ctrl+left-click *on a fixture*	Toggles its selection
Alt+left-click *on a composite fixture*	Select an element of a fixture
Alt+left-click *on background* + drag	Select elements or fixtures using a lasso
Alt+left-click *on fixture* + drag	Select elements or fixtures using a lasso
Left-click *on background* + Alt + drag	Select whole fixtures using a lasso
Left-click *on fixture* + Alt + drag	Constrain movement to one axis (horizontal or vertical directions)
Shift while selecting fixtures with a box	Selection order based on position, otherwise based on fixture number
Tab	Select the next fixture by number
Shift+Tab	Select the previous fixture by number
Ctrl+left-click *while in add fixture mode (blue border)*	Toggle the behaviour of Auto-finish
Alt+left-click *while in add fixture*	Add an instance of the last added fixture (or a new fixture if no fixture is

mode (blue border)	added yet)
Escape *while in add fixture mode (blue border)*	Finish adding fixtures
Escape *otherwise*	Toggle last fixture selection
Ctrl+drag	Create duplicates of the selected fixtures
Ctrl+Alt+drag	Create instances of the selected fixtures
Shift *while dragging fixture/s*	Disable fixture snapping
Delete/Backspace	Delete selected fixtures
Shift+Delete/Backspace	Delete selected fixtures from the Layout but keep the fixture in the project, even if they no longer exist on a layout
Ctrl+Delete/Backspace	Delete selected fixtures from the project
Shift+ 'Remove From Layout'	Delete selected fixtures from the Layout but keep the fixture in the project, even if they no longer exist on a layout
Ctrl+X	Cut the selected fixtures
Ctrl+C	Copy the selected fixtures
Ctrl+V	Paste fixtures from the clipboard
Ctrl+Shift+V	Paste instances of fixtures from the clipboard
Up/Down/Left/Right	Nudge the selected fixtures by the grid spacing, if show grid is turned off, nudge will default to a 1-pixel nudge
Shift+Up/Down/Left/Right	Nudge the selected fixtures by 1 pixel
Space+drag	Pan the view
Ctrl+0	Reset the zoom
Ctrl+F	Zoom to fit the window
Ctrl++	Zoom in
Ctrl+-	Zoom out
Ctrl+ mouse wheel	Zoom in and out
Middle-click + drag	Zoom into the drawn rectangle
Left-Click + drag + Shift	Pressing Shift after starting a lasso selection will sort by the aspect ratio (see here)
Alt+ mouse wheel (Shift+ mouse wheel)	Scroll Horizontally
Ctrl+drag *on Transform tool drag handle*	Maintain aspect ratio of selection
Alt *while dragging fixture/s*	Lock movement to a single axis

Browser

Delete/Backspace	Delete selected fixtures, groups or pixel matrices
Ctrl+left-click	Select multiple fixtures, groups or pixel matrices
Shift+left-click	Select all fixtures, groups or pixel matrices between two selections.
Alt+left-click	Deselects the contents of the group/pixel matrix
Up/Down	Move current row indicator up and down, and select the row
Shift + Up/Down	Move current row indicator up and down, and add the row to the selection
Ctrl + Up/Down	Move current row indicator up and down, but don't change the selection
Left/Right	Collapse/Expand current group
Space	Select current row
Ctrl + Space	Add current row to the selection

Mapping

Ctrl+N	Create new pixel matrix
Ctrl+D	Duplicate the current pixel matrix
Ctrl+I	Show pixel matrix properties
Ctrl+C	Copy the selected media
Ctrl+V	Paste media from the clipboard into the current folder
Delete/Backspace	Delete selected media
Up/Down/Left/Right	Nudge the selected items by 1 pixel
Space+drag	Pan the view
Ctrl+0	Reset the zoom
Ctrl+F	Zoom to fit the window
Ctrl++	Zoom in
Ctrl+-	Zoom out
Ctrl+ mouse wheel	Zoom in and out
Space *with media preview open*	Start/Stop media preview
Shift click *on overlapping elements*	Open selector to chose which element to select
Alt+ mouse wheel (Shift+ mouse wheel)	Scroll Horizontally

Patch

Ctrl+N	Show Add Universe popover
Ctrl+A	Select all patch records
Delete/Backspace	Delete selected patch
0-9	Type a universe number and the view will scroll to it after a short delay
Page Up/Down	Scroll to previous/next universe
Ctrl+Tab	Switch to the next protocol
Ctrl+Shift+Tab	Switch to the previous protocol

DALI

Ctrl+N	Create a new DALI Interface
Ctrl+I	Show DALI interface properties
Escape *in Scene Mode*	Toggle last fixture selection
Ctrl+0 *in Scene Mode*	Reset the zoom
Ctrl+F	Zoom to fit the window
Ctrl++ *in Scene Mode*	Zoom in
Ctrl+- *in Scene Mode*	Zoom out
Ctrl+ mouse wheel *in Scene Mode*	Zoom in and out
Middle-click + drag *in Scene Mode*	Zoom into the drawn rectangle
Alt+ mouse wheel (Shift+ mouse wheel) *in Scene Mode*	Scroll Horizontally

Scene

Ctrl+N	Create a new Scene in the current folder
Escape	Toggle last fixture selection
Ctrl+0	Reset the zoom
Ctrl+F	Zoom to fit the window

Ctrl++	Zoom in
Ctrl+-	Zoom out
Ctrl+ mouse wheel	Zoom in and out
Middle-click + drag	Zoom into the drawn rectangle
Alt+ mouse wheel (Shift+ mouse wheel)	Scroll Horizontally

Timeline

Ctrl+N	Create a New Timeline
Ctrl+D	Duplicate the current timeline
Ctrl+G	Go to timeline (enter name or number to filter the list); when one choice remains, press Enter to show the timeline
Ctrl+I	Show timeline properties
Ctrl+A	Select all timeline programming
Delete/Backspace	Delete selected timeline programming
Ctrl+left-click *while adding presets*	Toggle the behaviour of Auto-finish
Ctrl+drag *start/end of preset*	Snap to nearest preset, flag or waypoint
Shift+drag *preset*	Display the preset's end time and position
Ctrl+left-click *while adding flags*	Add flag and don't leave Add Flag mode
Esc	Finish adding presets or flags
Up/Down/Left/Right	Scroll the view
Space	Start/pause Simulation
Esc while simulating timeline	Stop Simulation
F *while simulating*	If in Add Flag mode, drop a flag at the simulation time
Ctrl+F	Zoom to fit the window
Ctrl++	Zoom in
Ctrl+-	Zoom out
Alt+ mouse wheel (Shift+ mouse wheel)	Scroll Horizontally
Ctrl+ mouse wheel (Cmd+ mouse wheel) *while over browser*	Increase / decrease timeline row height
Esc *while moving Gradient stop*	Cancel changing gradient

Interface

Ctrl+N	Create a new Interface
Ctrl+I	Show interface properties
Alt+ *select Colour Picker*	Sets the startup colour of the colour picker to the selected colour
Ctrl+drag *on one or more control*	Creates a duplicate of the selected control/s

Trigger

Ctrl+N	Create a new trigger of the last created type
Ctrl+left-click *on a trigger, condition or action*	Toggles its selection
Shift+left-click	Select a range of triggers, conditions or actions
Ctrl+A	When nothing is selected, select all triggers; when a condition or action is selected, selects all conditions/actions of the parent trigger

Hold Ctrl *while dropping a dragged trigger*	Create a copy of the trigger at the drop location
Hold Shift *while dropping a dragged condition or action*	Move the condition or action to the trigger it is dropped on
Delete/Backspace	Delete selected triggers, conditions or actions
Up/Down	Move current row indicator up and down, and select the row
Shift + Up/Down	Move current row indicator up and down, and add the row to the selection
Ctrl + Up/Down	Move current row indicator up and down, but don't change the selection
Left/Right	Collapse/Expand current trigger
Space	Select current row
Ctrl + Space	Add current row to the selection
Ctrl+B *in Script Editor*	Compile script

Simulate

Space	Start/Pause Simulation
Esc	Stop Simulation
Ctrl+0	Reset the zoom
Ctrl+F	Zoom to fit the window
Ctrl++	Zoom in
Ctrl+-	Zoom out
Ctrl+ mouse wheel	Zoom in and out
Middle-click + drag	Zoom into the drawn rectangle
Alt+ mouse wheel (Shift+ mouse wheel)	Scroll Horizontally

Notes for Mac OS X users

Unless otherwise noted, keyboard shortcuts on Mac OSX are the same as Windows, except Ctrl is replaced with ⌘ . Shift and Alt work as described for Windows.

Within Layout, Scene and Simulate, you can use Scroll gestures to move around the Plan.

Pharos Designer makes a good deal of use of the two button mouse with right-click being used to invoke context-sensitive dialogs. As the majority of Mac users have only a single button mouse they must hold Ctrl while clicking to get this functionality.

System Requirements

Pharos Hardware Requirements

This version of Pharos Designer can be used with the following controllers:

- Pharos LPC 1/2/4: Serial numbers greater than 006xxx
- Pharos LPC X: All Serial numbers
- Pharos VLC: All Serial numbers
- Pharos VLC+: All Serial numbers
- Pharos TPC (with or without EXT): All Serial numbers

NOTE: LPCs with a Serial number lower than 006000 and AVCs are only supported in Designer 1.x.x.

Computer System Requirements

Supported Operating Systems

- Microsoft Windows 7/8/10/11 (64bit)
- macOS 10.13.x (High Sierra) – 13.1.x (Ventura)

Minimum Requirements

- Intel Core i3 processor or above
- 2GB RAM
- 1GB free hard disk space
- 1024×768 screen resolution
- OpenCL 1.2 graphics support (for VLC/VLC+ simulation)
- Network connection (for connecting to Pharos hardware)

Recommended

- Intel Core i5 processor or above
- 8GB RAM
- 1920×1080 screen resolution

Web Interface Support

The Default Web Interface on a controller is supported by all modern web browsers, e.g. Edge, Firefox, Safari, Chrome and Chromium based browsers.

Custom Web Interface browser support will vary depending upon the Interface.

Hardware Overview

This version of Pharos Designer can be used with the following controllers:

- Pharos LPC 1/2/4: Serial numbers greater than 006xxx
- Pharos LPC X: All Serial numbers
- Pharos VLC: All Serial numbers
- Pharos VLC+: All Serial numbers
- Pharos TPC (with or without EXT): All Serial numbers

NOTE: LPCs with a Serial number lower than 006000 and AVCs are only supported in Designer 1.x.x.

Controllers

LPC 1/2/4

The Pharos Lighting Playback Controllers (LPCs) are solid state lighting controllers capable of being programmed with a series of light shows which can be controlled through either internal of external triggering.

This programming is all preprogrammed through the Pharos Designer software before being uploaded to the controller for stand alone operation.

Each type of LPC is capable of controlling a number of universes of DMX or eDMX compatible lighting fixtures. The number of universes that can be controlled is indicated by the number in the name of the controller (e.g. LPC 1).

More information is available here.

LPC X

The LPC X is designed to meet the unique needs of large landmark projects. It is available in capacities ranging from 10 DMX universes up to 100 DMX universes from a single unit with further scaling over Ethernet.

The LPC X provides the same programming options as the LPC, but the outputs are limited to eDMX protocols.

More information is available here.

TPC

The TPC is a Lighting Playback Controller with a touchscreen, allowing for simple user interaction with the programmed show file, and output of 1 universe of eDMX lighting control data.

As with all controllers, the programming for the TPC is done through Pharos Designer and uploaded to the TPC.

More information is available here.

EXT

The EXT is an extension to the TPC to provide additional connectivity to external systems and devices. It is a mains powered device, which provides PoE to the TPC, along with Digital/Analog inputs, a DMX connection, an RS232 connection and a DALI connection.

More information is available here.

VLC

The VLC is designed to handle large single canvases of DMX fixtures e.g. building façades, bridges or media screens. It is available in capacities ranging from 50 DMX universes up to 1500 DMX universes from a single

unit with further scaling over Ethernet.

As with the other controller's, the VLC is programmed through Pharos Designer and uploaded to the VLC.

More information is available here.

VLC+

The VLC+ is designed to handle large single canvases of DMX fixtures e.g. building façades, bridges or media screens. It is available in capacities ranging from 50 DMX universes up to 3000 DMX universes from a single unit with further scaling over Ethernet, and allows multiple layers of media output to be displayed on the canvas, along with dynamically moving and rotating the content.

As with the other controllers, the VLC+ is programmed through Pharos Designer and uploaded to the VLC+.

More information is available here.

Remote Devices

Remote Devices can be used alongside controllers to increase the functionality and/or connectivity of the control system. They are connected to the controller using Ethernet network connections and most are powered over PoE using the same connection.

RIO (08/44/80)

The three types of input/output RIOs allow various amounts of inputs and outputs to be added to the system. The inputs can be used for triggering within the project, and the outputs can be used to turn on other systems using volt free relays.

More information is available here.

RIO D

The RIO D allows for connection of the Pharos system to a DALI bus. This can be used either to output DALI commands to DALI ballasts, or to receive DALI commands from a DALI controller (or both).

See DALI for more information about outputting and DALI Triggers for more information about receiving DALI commands.

More information is available here.

RIO D4

The RIO D4 allows for connection of the Pharos system to up to four DALI buses. These can be used either to output DALI commands to DALI ballasts, or to receive DALI commands from a DALI controller (or both).

See DALI for more information about outputting and DALI Triggers for more information about receiving DALI commands.

More information is available here.

RIO A

The RIO A provides additional MIDI connections, and an Audio input into a project. This audio input allows for triggering based on the volume of the incoming audio, or the audio input can be used to receive Linear Time Code into the project.

More information is available here.

BPS

The Pharos BPS is a Button Panel Station which provides 8 programmable buttons, which can be used to control aspects of the project. Each button includes a white LED, which can be freely programmed to output various effects.

More information is available here.

TPS 5

The Pharos TPS 5 is a 5" Touch Interface built to replace the TPS hardware. It can be configured with a customisable interface to control aspects of the project. The Interface is designed within Pharos Designer and uploaded to the TPS 5 as part of the project.

Multi-touch support is available for TPS 5 from PharosDesigner 2.10 and above.

More information is available here.

TPS 8

The Pharos TPS 8 is an 8" Touch Interface built to replace the TPS hardware. It can be configured with a customisable interface to control aspects of the project. The Interface is designed within Pharos Designer and uploaded to the TPS 8 as part of the project.

Multi-touch support is available for TPS 8.

More information is available here.

TPS

The Pharos TPS is a Touch Interface which can be used to control aspects of the project. The Interface is designed within Pharos Designer and uploaded to the TPS as part of the project.

More information is available here.

EDN

The Pharos EDN is an Ethernet Data Node which provides 10 or 20 outputs of eDMX to DMX/RDM, dictated by variant, such as EDN 10. It can easily be configured within Designer and is patched exactly the same way as an LPC or any Pharos Controller.

It can also be converted to SDI mode, where it will work with attached SDIs (Serial Data Interfaces) to output a synchronous or asynchronous serial data protocol, set within the Network view.

More information is available here.

RIO G4

The RIO G4 provides 4 outputs of DMX/RDM over a single PoE connection. It can easily be configured within Designer and is patched exactly the same way as an LPC or any Pharos Controller.

It can also be converted to SDI mode, where it will work with attached SDIs (Serial Data Interface) to output a synchronous or asynchronous serial data protocol, set within the Network view.

More information is available here.

SDI

The SDI (serial data interface) is an accessory to the EDN. It connects directly to an EDN via an RS485 serial connection, allowing the EDN to output synchronous (SPI) or asynchronous serial protocols to fixtures such as

addressable LED tape.

More information is available here.

Hardware Setup

LPC 1/2/4

The basic setup for an LPC 1, 2 or 4 requires a power connection, a data connection and an output.

Power Connection

There are two options for power connection:

- DC Power - The LPC can receive 9-48V DC via the DC input on the bottom of the controller.
- PoE (Power over Ethernet) - This is provided by a PoE enabled network switch or PoE in-line injector.

Data Connection

To communicate between a controller and Designer, a data connection is required.

The physical connection can be achieved using either:

- a USB A-B cable
- an Ethernet cable (straight or crossover)
- an Ethernet network (with switch/hub/router)

The data connection for all of these configurations will be a network connection, through either standard network connections or a USB to Ethernet connection.

Output

To control any fixtures there will need to be a connection between the controller and the fixtures.

If using DMX, you will need a 3 core connector between one of the DMX ports and the DMX input of the first fixture on the DMX chain.

If using eDMX, this communication uses the Ethernet connection of the LPC, so the LPC and the receiving device must be connected to the same Ethernet network.

LPC X, VLC, VLC+

The basic setup for an LPC X, VLC, VLC+ requires a power connection and 2 Ethernet connections.

Power Connection

The LPC X, VLC, VLC+ is a mains powered device which can auto-detect the incoming power, and is compatible with all worldwide mains standards: 100-250V 50/60Hz.

This connection is made using an appropriate mains IEC cable.

Data Connection

The LPC X, VLC, VLC+ has 2 Ethernet ports, one for "Management" data and one for "Protocol" data.

Management data refers to any communication between Designer and the controller or between the controller and other controller, remote device or third party control systems.

Protocol data refers to the eDMX lighting outputs which are transmitted over the Ethernet connection to the receiving device/s.

These two connection should be made to separate ethernet networks containing the relevant equipment.

TPC

The basic setup for an TPC requires a PoE Ethernet connection to provide both power and data communication

This can be provided either from a PoE enabled switch or an in-line injector.

TPC + EXT

Alternatively the TPC can be powered by the EXT. This takes in mains power and outputs PoE to a TPC. In addition, the EXT provides connectivity options to complement the touch screen interface.

TPS, TPS 5 & TPS 8

The basic setup for a TPS, TPS 5 or TPS 8 requires a PoE Ethernet connection to provide both power and data communication.

This can be provided either from a PoE enabled switch or an in-line injector.

Remote Devices

All remote devices require a single connection to communicate with their associated controller.

Power Connection

The power for a remote device is provided over a Power over Ethernet connection.

Data Connection

This is provided by an Ethernet connection to the controller that it has been assigned to.

EDN

The basic setup for the EDN requires the EDN be connected to the same network as the data port of a controller.

NOTE: The controller must also be connected to ANY network on it's management port for the EDN to function. The EDN does not have to be connected to the same network as the management port.

Power Connection

The EDN is a mains powered device which can auto-detect the incoming power, and is compatible with all worldwide mains standards: 100-250V 50/60Hz.

This connection is made using an appropriate mains IEC cable.

Data Connection

The EDN has 2 Ethernet ports - one for data in, the other for daisy-chaining EDNs together. We advise daisy-chaining be limited to 8 EDNs or less without connecting a switch between, as this may adversely affect

playback quality.

EDN/RIO G4 + SDI

The SDI connects directly to any of the DMX ports on EDN or RIO G4. Limits on length apply depending on the protocol used;

- If using a synchronous signal, the distance of the SDI from the EDN/RIO G4 should be no more than 40m .
- If using an asynchronous signal, the distance of the SDI from the EDN/RIO G4 should be no more than 200m.

The SDI should receive power from the same source as the light fixtures.

Port Specifications

DMX

NOTE: Relevant to LPC, EXT, EDN, RIO G4, RDM.

The pins on these connectors are marked:

+ Data + ('Hot' or 'True')

— Data - ('Cold' or 'Complement')

⏚ Chassis ground ('Shield')

NOTE: The DMX ground is internally linked to the ground on the DC Input

To make up a cable to a 5 pin XLR the following connections should be made:

	LPC	3/5 pin XLR
Data +	+	3
Data -	—	2
Shield	⏚	1

MIDI Input and Output

NOTE: Relevant to LPC and RIO A

The MIDI input and output connectors are standard 5 pin DIN connections. They may be connected directly to any standard MIDI device.

Inputs

NOTE: Relevant to LPC, EXT and RIO (44/80)

Can be configured as Digital Input, Analog Input or Contact Closure.

Contact closure

An external volt-free switch may be connected between the input pin and the signal ground pin.

In this mode, the input pin is internally pulled-up to 5V via a 2.2Kohm resistor, so the switch only needs to be rated at 5V, 2.5mA or greater.

NOTE: With Contact Closure, "High" is regarded as "High Resistance", such as an open switch or, effectively, the "Off" state. "Low" is the inverse, and is considered "Low Resistence" or the "On" state. This is inverse to what is expected with the Digital Input where "High" is considered "High Voltage", or the "On" state.

Digital input

An external voltage source (such as a 12V trigger output) may be connected between the input pin and the signal ground pin. In this mode, the input pin is internally pulled down to 0V via a 202.1Kohm resistor and the maximum input voltage supported is 24V.

The LPC may be configured to specify what the 'high' and 'low' threshold voltages are. This facility can be used to provide 'Schmitt trigger' action.

Analog input

An external voltage source (such as a 0-10V analog signal) may be connected between the input pin and the signal ground pin.

In this mode, the input pin is internally pulled down to 0V via a 202.1Kohm resistor and the maximum input voltage supported is 24VDC.

The LPC may be configured to specify what the input voltage range is. Voltages inside this range are reported as 0% to 100%.

Relay Outputs

NOTE: Relevant to RIO (08/44)

The RIO features 8 (RIO 08) or 4 (RIO 44) relay outputs on two (RIO 08) or one (RIO 44) 8 way connectors.

The RIO relays are rated at 48V, 0.25A (AC or DC). This comparatively low rating is due to the use of solid-state relays to ensure silent operation and long-term reliability.

All relay outputs are fully isolated from each other (1kV) and all other ports.

Note: An external power supply is required to power the relay outputs. An external PSU can be used to power one or more relay outputs.

Ethernet

NOTE: Relevant to LPC, TPC, LPC X, VLC, VLC+, TPS, TPS 5, TPS 8, EXT and Remote Devices.

A standard 10/100TX Ethernet connection may be made to the device. If the device supports Power-over-Ethernet (PoE), a PoE switch or in-line injector can be used. The LEDs on the RJ45 jack itself are useful for debugging the Ethernet installation:

The Lnk LED will illuminate when an Ethernet link has been established.

The Dat LED will illuminate to indicate Ethernet traffic (not just Pharos-relevant).

RS232/RS485 Serial Port

NOTE: Relevant to LPC, LPC X, VLC, VLC+ , EXT and RIO (08/44/80)

The serial port's protocol (RS232 or RS485), data rate and format settings (baud, parity, stop bits, etc.) are configured using Designer.

In RS232 mode, the port operates in full duplex with the following pinout:

R/+ Receive
T/- Transmit
⏚ Signal Ground

In RS485 (and DMX In) mode, the port operates in half duplex with the following pinout:

R/+ Data +
T/- Data -
⏚ Signal Ground

NOTE: Not available on LPC X, VLC or VLC+ , these support RS232 only.

The serial port is not isolated from the power supply. If isolation is required, it must either be provided by the connected device or a separate isolator should be used.

Analog Audio Input

NOTE: Relevant to RIO A

Balanced stereo audio input is provided @ 0dBV line level on a 6 way connector:


Balanced audio right channel +


Balanced audio right channel - (tie to ground for unbalanced)


Signal ground


Balanced audio left channel +


Balanced audio left channel - (tie to ground for unbalanced)


Signal ground

The audio input can also accept linear time code (LTC) such as SMPTE/EBU on either channel, but not both, configured using Designer. The Audio / LTC LED will indicate peak for audio and valid for time code.

LTC Audio Input

NOTE: Relevant to LPC X S3 and VLC+ M.


Balanced LTC audio right channel +

Balanced LTC audio right channel - (tie to ground for unbalanced)


Signal ground


Balanced LTC audio left channel +


Balanced LTC audio left channel - (tie to ground for unbalanced)


Signal ground

Balanced LTC audio input is provided @ 0dBV line level on a 6 way connector:

This port accepts linear time code (LTC) such as SMPTE/EBU on either channel.

Audio Input and Output

Analog Audio Output

NOTE: Relevant to LPC X S3 and VLC+ M.

Balanced audio output is provided @ 0dBV line level on a 6 way connector:


Balanced audio right channel +

Balanced audio right channel - (tie to ground for unbalanced)


Signal ground


Balanced audio left channel +

Balanced audio left channel - (tie to ground for unbalanced)

⏚ Signal ground

Audio output is configured on a per-timeline basis with Timeline Audio[help/reference/timeline/timeline audio.htm]

Stereo Audio Output

NOTE: Relevant to LPC X Rev 2, VLC and VLC+.

Stereo analog audio ports with RCA connectors.

SPDIF Audio Output

NOTE: Relevant to LPC X Rev 2, LPC X S3, VLC, VLC+ and VLC+ M.

Digital audio port with RCA connector.

DALI

NOTE: Relevant to RIO D, RIO D4 and TPC with EXT.

A DALI bus interface is provided on a 3 way connector:

DALI bus (polarity insensitive)

DALI bus (polarity insensitive)

⏚ Chassis ground (for optional shield) (RIO D and EXT)

NC No connection (RIO D4 only)

NOTE: RIO D, RIO D4 and TPC with EXT do not provide DALI power, so a separate DALI power supply is required.

Video Input and Output

DV Firewire Input

Standard IEEE 1394 (Firewire) connection which may be used as a live video input (configured within a timeline).

NOTE: Only available on LPC X Rev 1 (Serial Numbers below: 011000).

DVI-D Input

Standard DVI-D connection which may be used as a live video input to a Pixel Matrix, or VLC Content Target (configured within a timeline).

NOTE: Available on VLC, VLC+, or LPC X Rev 2 (Serial Numbers above: 011000) as an optional extra.

HDMI Input

Standard HDMI connection which may be used as a live video input to a Pixel Matrix (configured within a timeline).

NOTE: Relevant to VLC+ M. Available on LPC X S3 as an optional extra.

DVI-I Output

Standard DVI-I Output which can be used to connect a monitor to display the current output of a pixel matrix (LPC X) or VLC+ output.

NOTE: Available on LPC X and VLC+.

DisplayPort Output

Standard DisplayPort Output which can be used to connect a monitor to display the current output of a pixel matrix (LPC X S3 and VLC+ M).

TPC Learning Infrared Receiver

The TPC may be taught to recognise up to 16 different infrared (IR) codes from a standard IR remote control. When a key on the remote control is pressed during normal operation, the TPC will react as though one of its user interface controls has been touched.

The TPC does not have to be part of a networked system to learn IR codes, all that is required is PoE power and the donor remote control:

To enter Learn Mode:

1. Enter by pressing the CFG (config) button. This is located underneath the magnetic overlay at the top left of the display, underneath the Reset button.

- The screen will display the IR configuration interface.

To learn an IR code:

1. Press the Set button alongside the code to be learnt.

- A progress indication will appear on the left of the row.

2. Within ten seconds, point the IR remote at the TPC and press the desired key.

- The progress indication will be replaced with a tick icon when the code has been learnt.

To test an IR slot

1. Point the IR remote at the TPC and press a key.

- If the IR code received is associated with an IR slot, the slot will be highlighted.

2. Release the key on the IR remote.

- The IR slot will no longer be highlighted.

To erase an IR code:

1. Press the Clear button alongside the code to be erased.

- The tick icon next to the code will disappear.

To exit Learn Mode:

1. Press the CFG (config) button.

- The screen will display the user interface for the loaded presentation, or indicate that no user interface is present on the memory card.

BPS Learning Infrared Receiver

The BPS may be taught to recognise up to 8 different infrared (IR) codes from a standard IR remote control. When a key on the remote control is pressed during normal operation, the BPS will react as though one of its 8 buttons has been pressed.

The BPS does not have to be part of a networked system to learn IR codes, all that is required is PoE power and the donor remote control:

To enter Learn Mode:

1. Enter by holding down the bottom two buttons while pressing and releasing reset.
    - The buttons will display a clockwise chase sequence
2. Release the bottom two buttons.
    - Each button will flash quickly (4Hz) if an IR code has been learnt, or slowly (1Hz) if not
    - No network communication will operate while in Learn Mode
    - Learn Mode will automatically exit after 60 seconds of inactivity

To learn an IR code:

1. Briefly press and release a single button which should learn the IR code.
    - The button will start flashing rapidly (8Hz) and the other buttons will extinguish
2. Within ten seconds, point the IR remote at the BPS and press and hold the desired key.
    - The buttons will display a clockwise chase sequence when the IR code has been learnt
3. Release the key on the IR remote.
    - The button now will be flashing quickly (4Hz) to indicate that it has an IR code stored

To erase an IR code:

1. Press and hold for three seconds the button which should erase its IR code.
    - The buttons will display a clockwise chase sequence when the IR code has been erased
2. Release the button.
    - The button will now be flashing slowly (1Hz) to indicate that it has no IR code stored

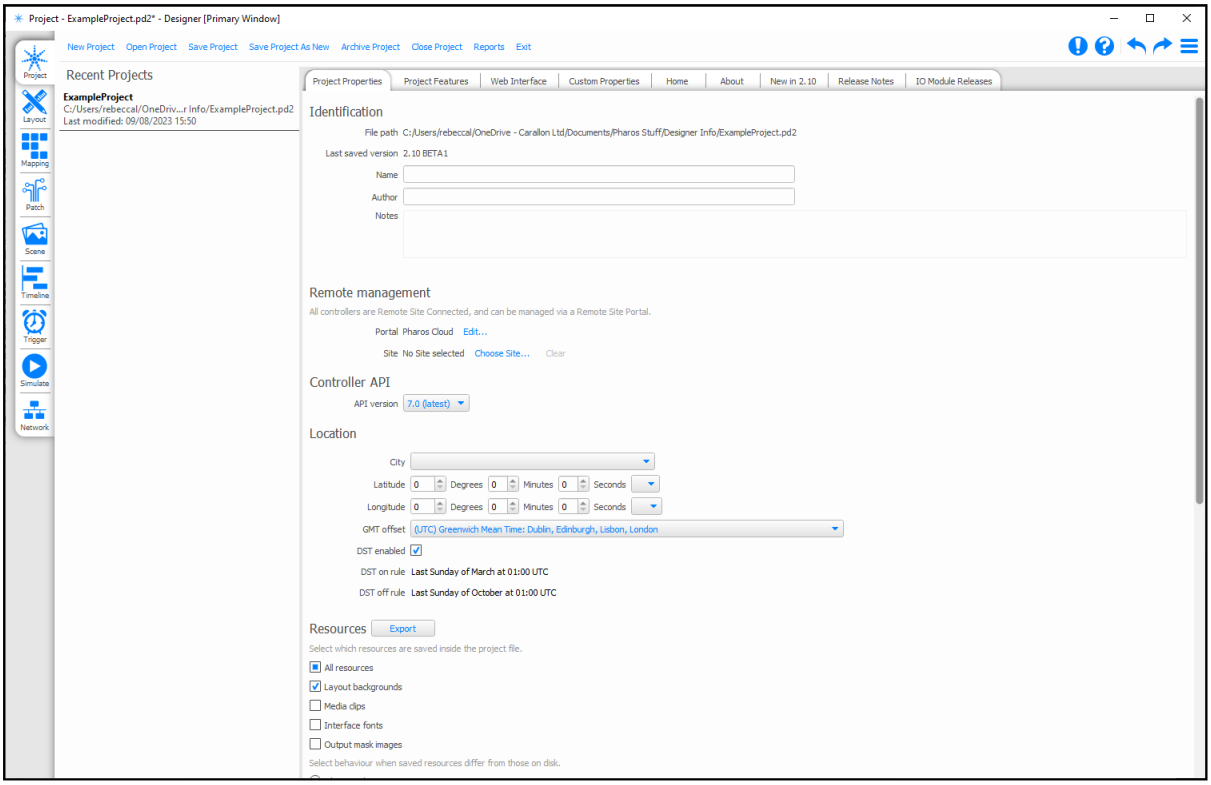
To test an IR code:

1. Point the IR remote at the BPS and press and hold the key to test.
    - The button(s) that has learnt this code will illuminate solidly, all others will extinguish
2. Release the key and test the others.

To exit Learn Mode:

1. Press the reset button or wait for 60 seconds.
    - The buttons will now revert to normal operation
    - Network communication will resume

Project Overview



The Project View is used to manage a project.

Project Toolbar

The project toolbar allows you to start a New project, Open existing projects, Save Project and Save Project As New or Close the current project.

You can also access Reports on the project from here

Recent Projects Browser

The right hand section of the Project tab will display information on the five most recent projects to be worked on.

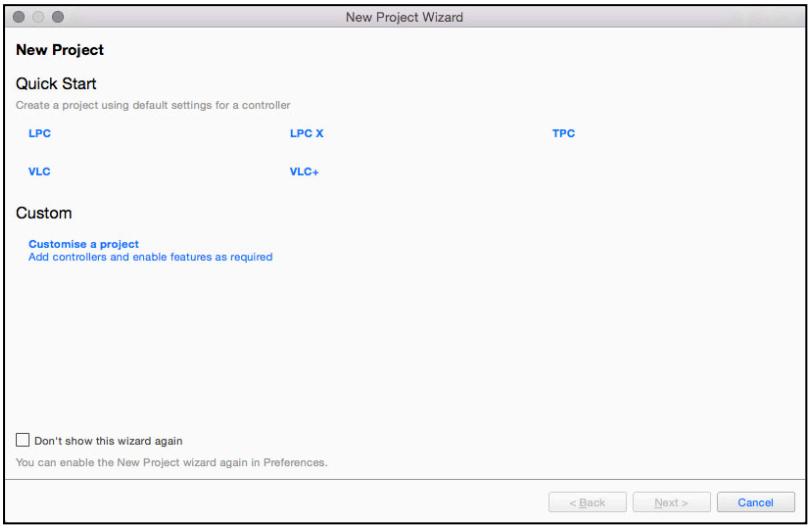
To enable ease of identification, this pane will display the filename, file path, and last modification date and time.

Project Tabs

The right hand section of the Project tab contains the Project Properties, Project Features, Web Interface, Custom Properties and About tabs.

New Project Wizard

The New Project Wizard

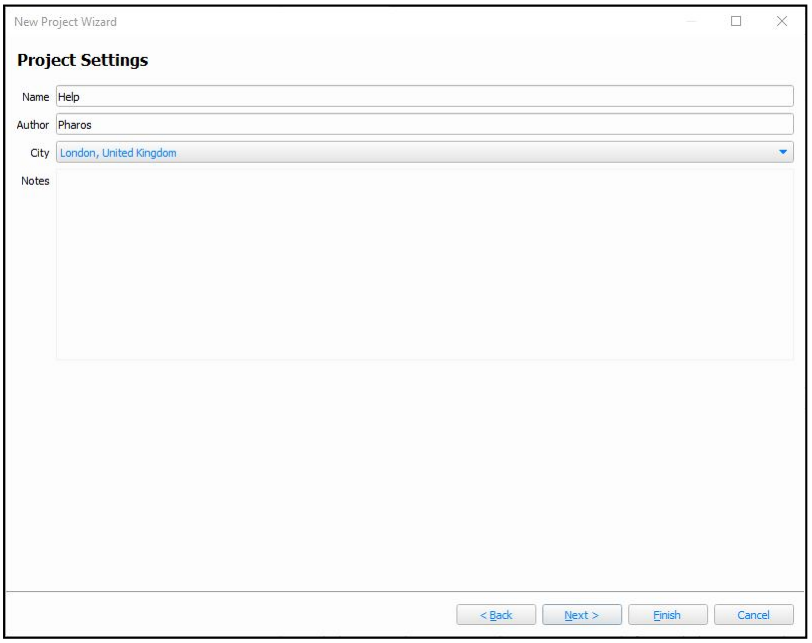


Choosing New Project will open the New Project Wizard, which can be used to either quick start a project for a controller or customise the project.

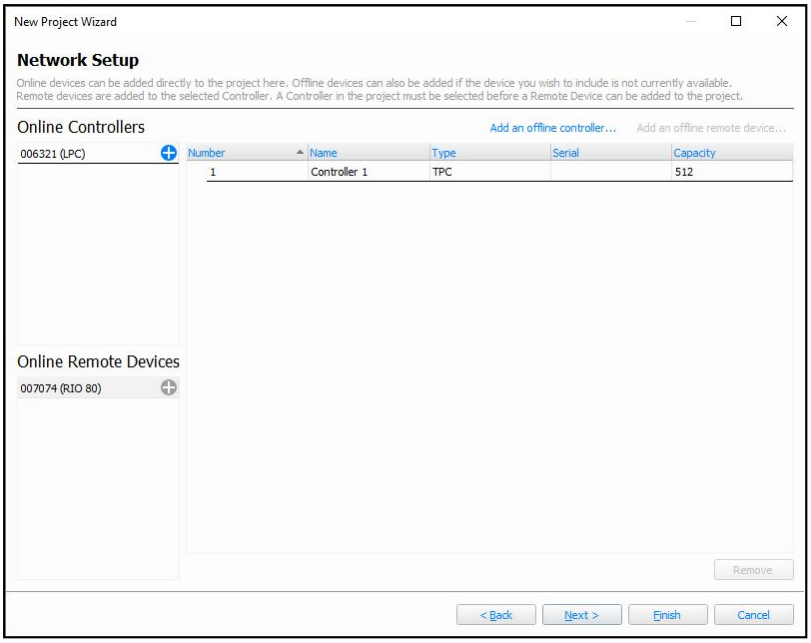
Quick Start

Selecting a controller type will create an empty project with the defaults for the selected controller.

Custom

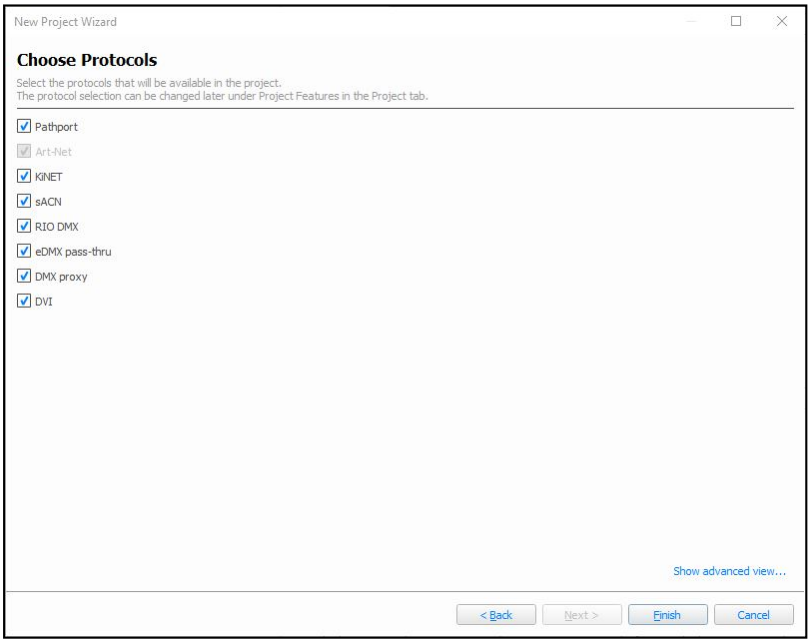


The Project properties page allows you to set the Name, Author, Location and some Notes for the project



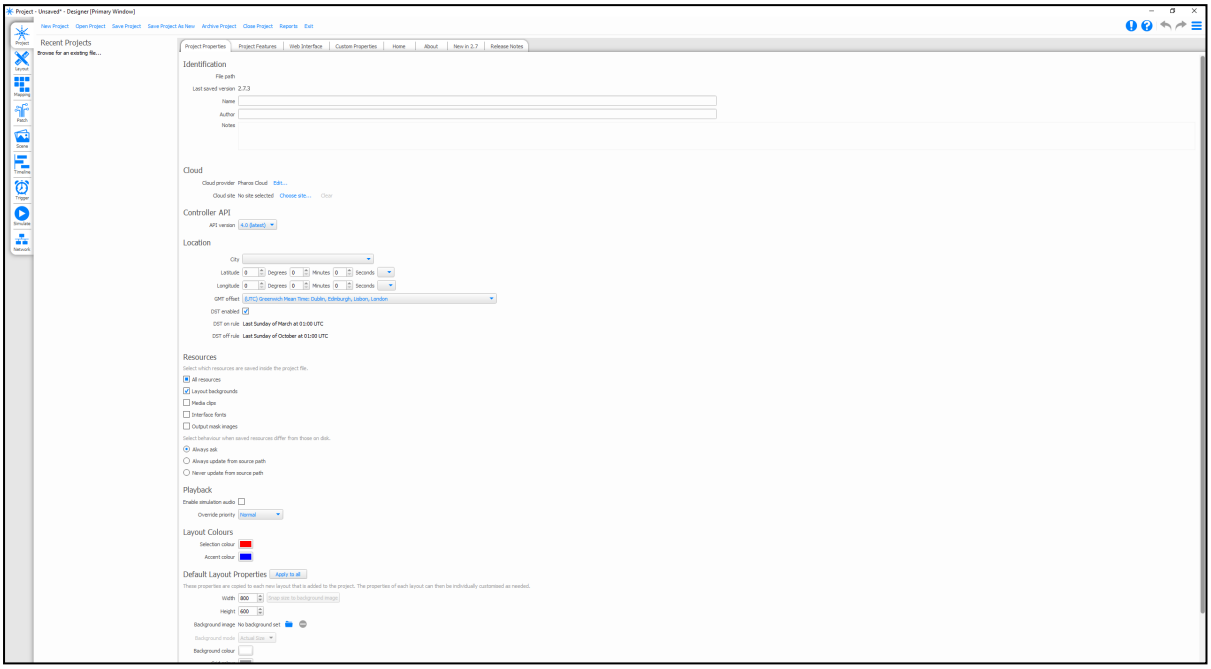
The Network Setup allows you to add any Pharos devices on the network to the project or add an offline device.

To add Remote devices, select a controller in the project, then add an Online or Offline controller to the controller.



The Choose Protocols page allows you to deselect any protocols that will not be used in the project. Show advanced view will allow you to setup any Project Features.

Project Properties



Project Properties

Project Identification

The project filename (*.pd2) and path is displayed for reference.

Underneath are two fields for optionally entering a project title and the project's author, these fields are displayed on the Controller's web interface home page and are useful for reference once the installation is completed.

If the title field is left blank the web interface will instead display the project's filename which may be useful for tracking iterative versions.

The Notes section is a good place to make notes specific to this project.

Logging into Remote Management service

Choosing Portal Provider

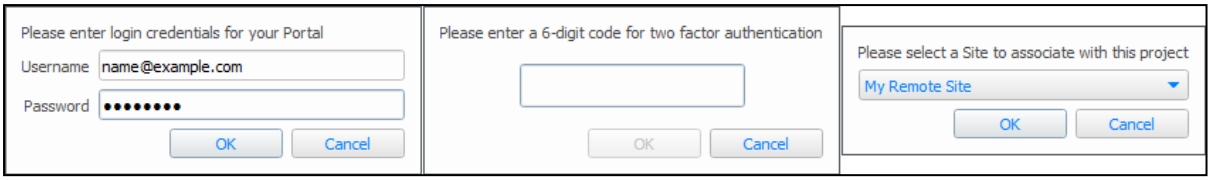
The Remote Management portal can be configured by clicking **Edit**. This will show a text bar in which a URL can be pasted. Many of our clients will have their own portal, in which case they will provide you with a URL, which can be pasted here. If the URL is recognised as a SixEye-powered Portal, the **Save** option will remain greyed-out.

Logging into a Site

Selecting **Choose site** will enable you to log into the Portal that is chosen above.

Once you log in, if you have Two Factor Authentication (2FA) enabled, an authentication window will follow.

You can then choose your Site, with all applicable Sites listed in a dropdown.



Controller API

API Version

The API Version setting will allow you to specify the API version that the project should use. If set to Legacy, this will use the older API (no longer documented in this Help), whereas if an API version is set, that API will be used:

- Latest API version

Location

City or Latitude/Longitude

The location settings are used to set the location of the installation to ensure correct operation of the Controller's internal astronomical clocks. A city picker is provided to facilitate the coordinate entry but values can be entered directly into the Latitude and Longitude boxes, this information can be found online.

Time zone

The local time zone can be entered as an offset to GMT, for example New York would be -05:00 being 5 hours behind GMT. If the city picker is used to select the location then the time zone will automatically be set.

Daylight Saving Time

Check the Daylight Saving Time box to enable automatic DST adjustment. The rules for Daylight Saving differ by region but, if the city picker is used to select the location, the correct settings for that region should appear in the DST on rule and DST off rule fields.

Resources

The resources section allows you to specify which project resources are included when you save the project.

The resources will always be included when the project is archived.

Playback

Simulation Audio

Checking this field will add a new row to your Timelines. In this, audio files can be placed, which will then play when the timeline is simulated.

NOTE: This is not the same as "Timeline Audio" on page 166. This is solely for simulation use, and will not be uploaded and output by the controller.

Override Priority

Chose the priority at which to output Overrides (from the Set RGB action or from scripting).

These priorities match up with the Timeline priorities and work in the same stack.

To ensure that overrides are always on top of other programming, set this to High.

Setting the priority to a lower level allows the override colour to go below other timelines, which can allow the use of a transparency within programming.

NOTE: To match v1.x.x behaviour, set this to High

Layout Colours

Selection Colour

This is the colour used to highlight a fixture icon on the Layout, Scene or DALI Scenes Layout, when it is selected.

Accent Colour

This is the colour used to highlight a fixture icon on the Layout, Scene or DALI Scenes Layout, when it has been programmed or is being Highlighted.

Default Layout Properties

These properties will be used for any new Layout created after changing the properties and will not affect existing layouts (unless the Apply to All button is used). These properties can be adjusted on a per layout basis

Size

The Layout size can be set via the Width and Height fields (in pixels). The maximum Layout size is 8192x8192 pixels.

The "Snap size to background image" button allows you to set the layout size to be the same as an imported background image.

Background image and mode

You can set an image to be the background for a Layout. This may be a fixture plan to allow you to easily locate the fixtures within the project, or a photo of the project space to allow you to visualise the lighting more effectively.

The image can be in Bitmap (*.bmp), Portable Network Graphics (*.png) or JPEG (*.jpg) format.

Once you have added a background image, you can chose how it is applied to the layout using the Background Mode.

- Actual Size - display the image at 1:1 scale
- Fit - scale the image so its largest dimension fits on the layout (maintaining Aspect Ratio)
- Fill - scale the image so its smallest dimension fits on the layout (maintaining Aspect Ratio)
- Stretch - Fit the image to the background (doesn't maintain Aspect Ratio)

Background colour

When a background image hasn't been set, a background colour can be used to help make the lighting more visible against the background.

Grid

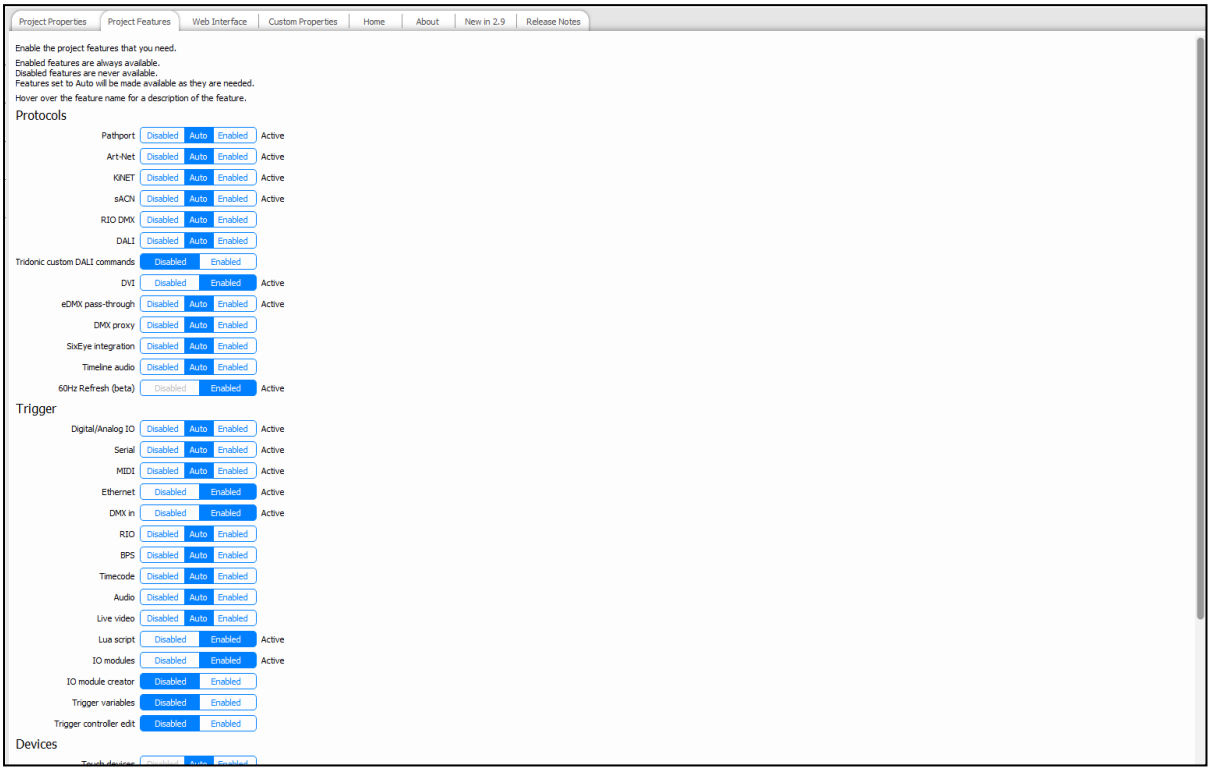
The Grid colour option allows you to set the colour that the grid is displayed in. Generally this should be a colour that is visible on your chosen background.

The Grid spacing can be specified in pixels. This is the grid on the layout which fixtures can be snapped to.

The Grid subdivisions defines the interval of major gridlines. These are the gridlines which are shown in bold on the layout.

NOTE: The Pharos Designer fixture library uses a scale of 1cm:1pixel (0.394":1pixel) for the fixture icons so, for best results, background images should be to this scale. If your installation is too large to be accommodated at this scale (i.e. bigger than 81.92m in either axis) then change the scale and use the Fixture Size settings to adjust the scale of your fixture icons accordingly. Alternatively, you can split your installation across multiple layouts.

Project Features



The Project Features Tab allows you to select the features that will be available to the current project.

By default features such as the DALI, Scene and Interface Editor Tabs are not visible, but when the relevant elements are added e.g. a DALI Ballast, the tab will become visible.

The same is true for various trigger types and protocols.

Options

Disabled	The feature will not be visible in this project
Enabled	The feature will always be visible in this project
Auto	The feature will become visible when an associated element is added to the project

When a feature is marked as Active, it is available within the project

In-Situ Enable

Certain features without an auto option have an in-situ enable option •••. This will enable the feature without having to come back to the Project Features page.

Protocols

Select which protocols are available to create new universes in the Patch View.

- Pathport - Used exclusively with Pathway products (Default: Auto, Active)
- Art-Net - Typically used with Artistic Licence products (Default: Auto, Active)
- KiNET - Used exclusively with Philips Color Kinetics products (Default: Auto, Active)
- sACN - Developed by ESTA as an eDMX standard (Default: Auto, Active)

- RIO DMX - Used to output DMX from a Pharos RIO 08/44/80 (Default: Auto)
- DALI - DALI Mode, triggers and actions (Default: Auto, Enabled by adding a DALI Ballast, RIO D or RIO D4)
- Tridonic custom DALI commands - Specific commands for Tridonic DALI devices (Default: Disabled)
- DVI - An output option on the LPC X (Default: Enabled, Active)
- eDMX pass-through - Used to receive eDMX and send it out of a DMX port (LPC or TPC+EXT) (Default: Auto, Active)
- DMX proxy - Allows an LPC 1 to output a TPC's DMX universe (Default: Auto)
- Timeline audio - Allows audio presets to be placed on timelines for synchronous playback on LPC X, VLC and VLC+(Default: Auto)
- Remote Site Integration - Enables users to integrate into their Remote Management service.
- 60Hz Refresh - Enables the controllers to use 60Hz playback.

Trigger

Select which Trigger types are available:

- Digital/Analog IO - (Default: Auto, Active)
- Serial - (Default: Auto, Active)
- MIDI - (Default: Auto, Active)
- Ethernet - (Default: Enabled, Active)
- DMX In - (Default: Enabled, Active)
- RIO - (Default: Auto)
- BPS - (Default: Auto)
- Timecode - Used in Triggers or Actions (Default: Auto)
- Audio - Used in Triggers or Actions (Default: Auto)
- Live video - Used in Triggers (Default: Auto)
- Lua script - Used in Triggers or Actions (Default: Enabled, Active)
- IO modules - Used in Trigger (Default: Enabled, Active)
- IO module creator - Use in Trigger - Modules to create new IO Modules (Default: Disabled)
- Trigger Variables - Variable options for Conditions and Actions (Default: Disabled)
- Trigger controller edit - Used in triggers to determine which controller/s run the trigger/conditions/actions (Default: Disabled)

Devices

Select the controller specific sections to make available:

- Touch Devices - (TPC, TPS, TPS 5 and TPS 8) Interface Mode, Touch Device Triggers and Actions (Default: Auto)
- VLC - Compositions within Mapping Mode, VLC Actions (Default: Auto)
- VLC+ - Compositions within Mapping Mode, VLC+ specific Content Targets, Masks, Content Target Actions (Default: Auto)
- SixEye Integration - Allows Designer to link to a Remote Site and associate controllers with the Site (Default: Auto)
- EDN - Support for the Pharos Ethernet Data Node and RIO G4 (Default: Auto)
- Multi-controller project - Enables features specific to multi-controller projects (Default: Auto)

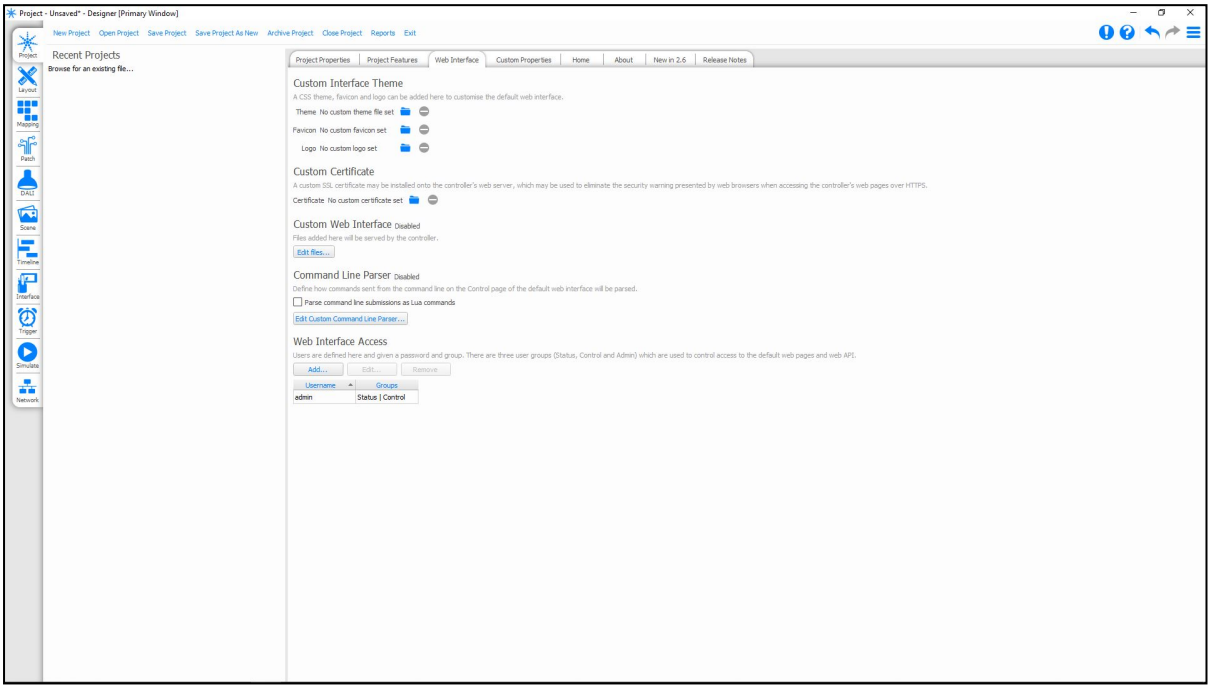
Editors

Select which editors to make available:

- Scene - Scene mode, triggers and actions (Default: Auto)
- Custom Presets - Used within Mapping and Timeline (Default: Disabled)
- Install Replications - Used in Network to replicate the project over multiple sets of controllers (Default: Disabled)
- Additional VLC+ Targets - Used across the project to add Targets 3-8 for the VLC+ (Default: Disabled)

Web Interface

The Web Interface tab within the Project view contains various settings relating to the Web Interface of the project.



Properties for both the Default web interface and Custom Web interfaces can be accessed from here.

Custom Interface Theme

These settings can be used to customise the default web interface where a simple rebranding is required.

Theme

The theme of the Default web interface is defined by a CSS file, based on the Bootstrap environment. This can be overridden by a user created CSS file.

The CSS style could be a Bootstrap theme, such as those available from Bootswatch, or a simple custom written CSS file to style the web interface, utilising web browser inspection tools.

Favicon

The favicon can be specified using the Import button. This will prompt you to search for an image file to add to the project.

A favicon is generally 16 x 16 px, so can't contain much detail.

Once set, you will need to upload the project to see the change.

Logo

Within the web interface a logo is displayed. Typically this will be the Pharos logo, but this can be changed to any image imported into the project.

Once set, you will need to upload the project to see the change.

Custom Certificate

It is possible to access the controller's web interface using a secure connection (HTTPS and WSS).

A custom SSL certificate may be installed. If a DNS record has been setup for the controller's IP address then this allows a certificate for the domain from a trusted Certificate Authority (CA) to be used. Web browsers will automatically verify a certificate from a CA. If no certificate is set, the controller will use a self-signed certificate for HTTPS connections to the web server. Some network setups can have issues with self-signed certificates, so if your installation is affected, adding a trusted certificate can remove these issues.

Custom Web Interface

See Custom Pages for details.

Custom Command Line Parser

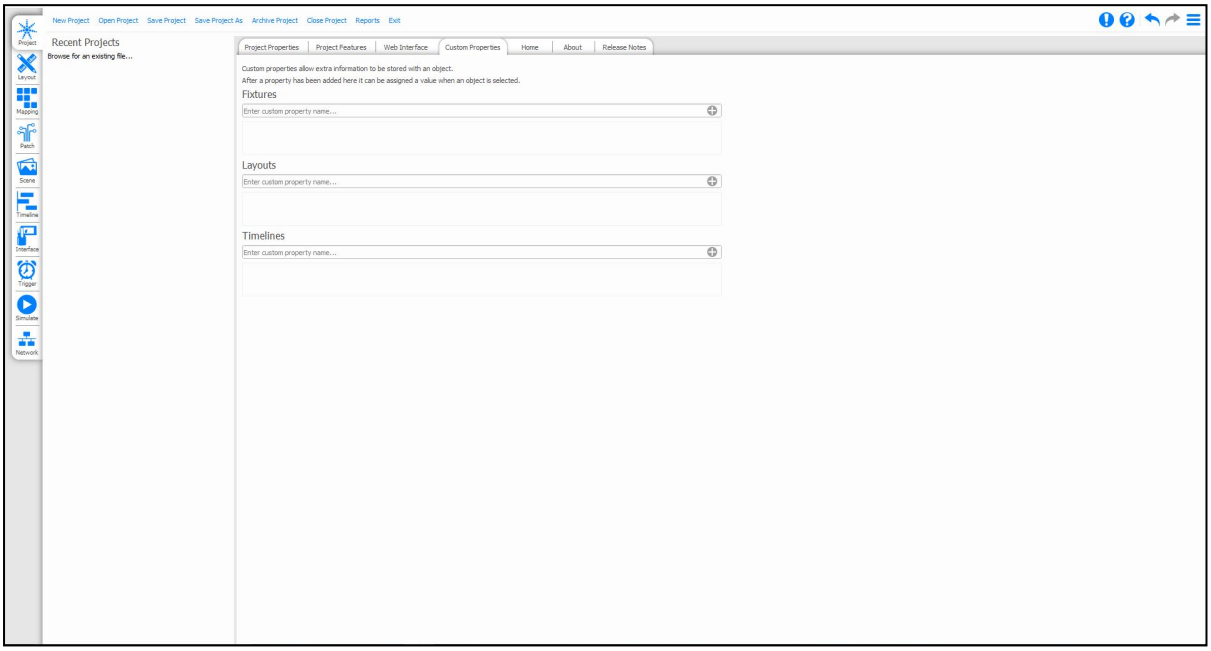
Choosing "Parse command line submissions as Lua Commands" will automatically run commands entered into the command line as Lua trigger scripts, otherwise an additional parser script is required, see Command Line for details.

Web Interface Access

This feature has been deprecated in 2.9. Please see Device Configuration for details on the new system that has replaced it.

Custom Properties

Custom properties can be used to store information about Fixtures, Layouts and Timelines which couldn't otherwise be stored.



To add custom properties

- Open the Custom properties tool from the Project toolbar.
- Select the object type (Fixture, Layout or Timeline) from the tabs across the top.
- Add a property name to the text box and click the add button.
- The property will be added to the selected object.

You can check the property has been added by navigating to the relevant section (Layout for fixtures and layouts and Timeline for timelines) and select a fixture, layout or timeline.

The new property should be available as a text field in the properties display.

Home

The Home tab is the default view when opening Designer.

This will commonly contain any information on software updates, software betas or notices.

New In

This page is where the new features for the software are shown, along with links to more information. On first running a new version of the software, Designer will default to this page, showing the users the new features that they can find in that build.

Release Notes

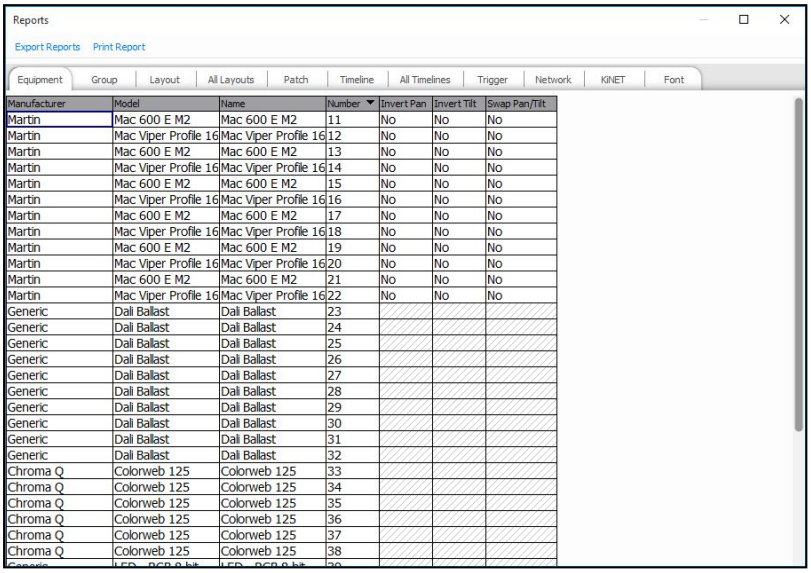
The release notes are a full list of new features, notable changes, bux fixes and known issues. This page is important to show users issues that have been fixed, existing features that have been changed, and issues that we are aware about but have not been able to fix as of yet.

About

The About Tab contains details about the current version of Pharos Designer, the EULA and various other relevant licences.

Reports

Designer can automatically produce reports to aid in producing documentation for the project:



Manufacturer	Model	Name	Number	Invert Pan	Invert Tilt	Swap Pan/Tilt
Martin	Mac 600 E M2	Mac 600 E M2	11	No	No	No
Martin	Mac Viper Profile 16	Mac Viper Profile 16	12	No	No	No
Martin	Mac 600 E M2	Mac 600 E M2	13	No	No	No
Martin	Mac Viper Profile 16	Mac Viper Profile 16	14	No	No	No
Martin	Mac 600 E M2	Mac 600 E M2	15	No	No	No
Martin	Mac Viper Profile 16	Mac Viper Profile 16	16	No	No	No
Martin	Mac 600 E M2	Mac 600 E M2	17	No	No	No
Martin	Mac Viper Profile 16	Mac Viper Profile 16	18	No	No	No
Martin	Mac 600 E M2	Mac 600 E M2	19	No	No	No
Martin	Mac Viper Profile 16	Mac Viper Profile 16	20	No	No	No
Martin	Mac 600 E M2	Mac 600 E M2	21	No	No	No
Martin	Mac Viper Profile 16	Mac Viper Profile 16	22	No	No	No
Generic	Dali Ballast	Dali Ballast	23			
Generic	Dali Ballast	Dali Ballast	24			
Generic	Dali Ballast	Dali Ballast	25			
Generic	Dali Ballast	Dali Ballast	26			
Generic	Dali Ballast	Dali Ballast	27			
Generic	Dali Ballast	Dali Ballast	28			
Generic	Dali Ballast	Dali Ballast	29			
Generic	Dali Ballast	Dali Ballast	30			
Generic	Dali Ballast	Dali Ballast	31			
Generic	Dali Ballast	Dali Ballast	32			
Chroma Q	Colorweb 125	Colorweb 125	33			
Chroma Q	Colorweb 125	Colorweb 125	34			
Chroma Q	Colorweb 125	Colorweb 125	35			
Chroma Q	Colorweb 125	Colorweb 125	36			
Chroma Q	Colorweb 125	Colorweb 125	37			
Chroma Q	Colorweb 125	Colorweb 125	38			

These reports can also be used to adjust some data relating to some elements e.g. fixture name.

Report types

Equipment

Lists all the fixtures used in the project. Including Manufacturer, Model, Name, Number and Pan/Tilt properties.

Group

Lists all the groups in the project and the fixtures which are part of each group.

Layout

Lists all the fixtures on a specified layout. The complete fixture identification is shown complete with name, notes, number, Layout position and rotation.

All Layouts

Lists all the layouts in the project with its number, name and size.

Patch

Lists the complete patch data.

Timeline

Provides a summary of each timeline. Use the pull-down to select the timeline. You can also filter this report so that only presets, flags or both are shown.

All Timelines

Provides a summary of all the timelines in the project.

Trigger

Provides a summary of the trigger programming. Complete with user annotation.

Network

Lists all the Controllers and Remote Devices in the project.

KiNET

Lists all the KiNET power supplies that have been added to Controllers in the project.

Font

Lists the fonts used in Dynamic Text presets on timelines in the project.

All Scenes

Provides a summary of all the scenes in the project.

Report spreadsheets

All the reports are presented in spreadsheet form and present an accurate account of the project programming that updates as changes are made, such that it always shows accurate data.

The reports can be sorted and reorganised. Right-click on the column headings to set/clear primary/secondary sorts. Drag column headers to move them, drag the header divider lines to resize them. These spreadsheet layout settings are stored with the project.

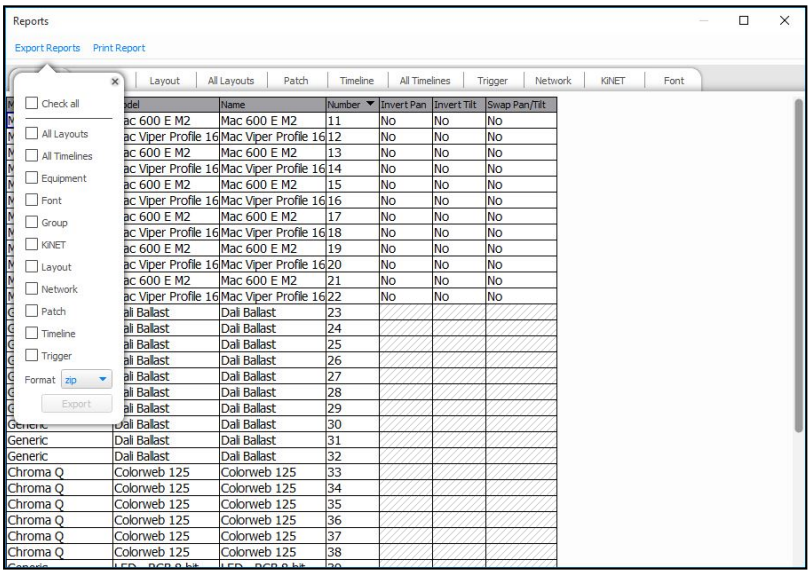
Some columns can be edited within the Report Spreadsheet, and this will have a direct impact on the rest of the project, including:

- Equipment:
  - Name
  - Number
- Group
  - Number
  - Name
- Layout
  - Name
  - Number
  - X Position
  - Y Position
  - Angle
- All Layouts
  - Name
- All Timelines
  - Number
  - Name
  - Group
  - Priority

    - Hold
    - Loop
    - Release at End
- Trigger
    - Number
    - Name
    - Description
- Network
    - Number
    - Name
- All Scenes
    - Name

Exporting Reports

To export one or more of the reports for the show file, the Export tool can be used to select which reports to export.



- Select the reports that you want to include in the export. The first option can be used to Select and Deselect All options.
- The compression option is used to decide which form of file compression you want to use out of: Zip, Tar or Tar.gz
- Select where to save the exported reports to with the Browse button
- Click Export to save the reports to your chosen location, or Cancel to close the dialog box.

Printing Reports

The current report can be printed, by selecting the Print Report option.

This will open a System print dialogue.

Layouts and Instances

There are two types of layouts available within a project, depending upon the controllers in use.

- Default Layouts - available for all controllers except the VLC/VLC+.
- VLC Layouts - Available when using a VLC or VLC+

Default Layouts

Within Layout there is the ability to create multiple views for a project, either different sections of a project, or different views of the same area.

Managing Layouts

New Layouts

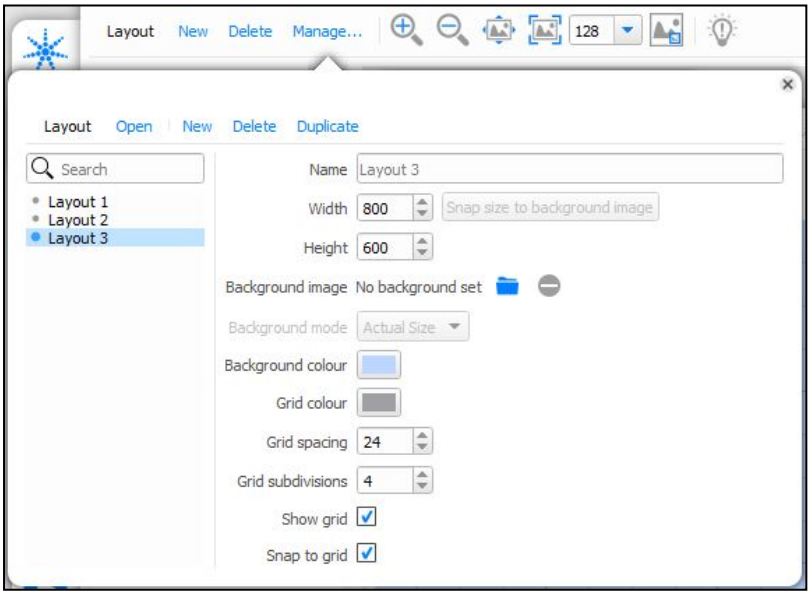
The New button in the Layout menu bar will generate a new blank layout

Deleting Layouts

The Delete button in the Layout menu bar will remove the layout from the project

Managing Layouts

The Manage... dialog can be used to keep track of all the layouts in the project and open layouts that have been closed. This dialog box also allows access to the properties for layouts that aren't currently active.



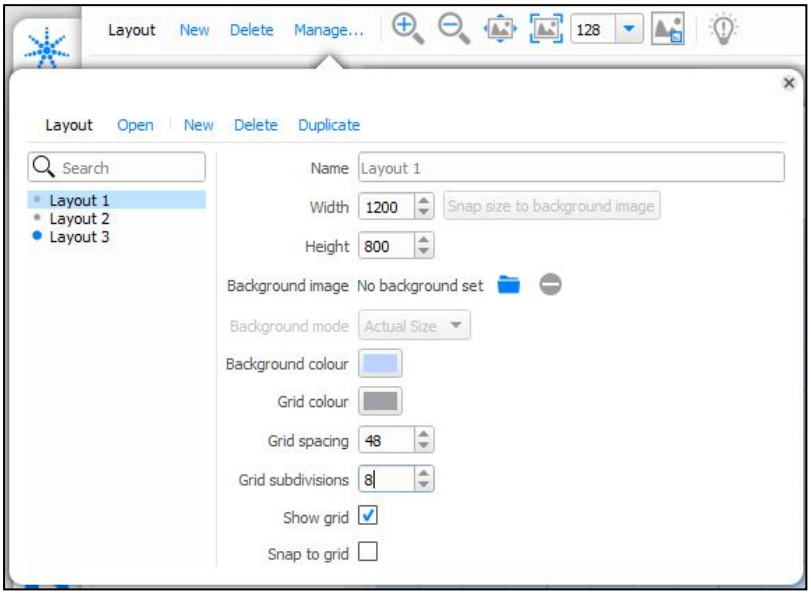
Duplicating Layouts

The duplicate button in the Manage window will create an exact copy of the current layout with new fixtures.

You will be prompted to choose whether to create the new layout with new fixtures or instances of the original fixtures (see below).

Layout Properties

The Properties... button in the Layout menu bar gives access to the properties of the current layout.



Name

Provide a name for the layout to aid identification

Size

Width and height options can be used to change the size of the layout.

The Snap size to Background Image can be used to match the layout size to the background image size.

Background Image and Mode

Select an image to be used in the background, such as a fixture plan.

Background mode can be used to define how the image is displayed.

Background Colour

Select the colour to display in the background of the Layout.

This colour is also used in Scene, DALI Scene and Simulate.

Grid Settings

Grid colour sets the colour of the grid pattern

Grid Spacing defines the number of units to space the gridlines apart.

Grid subdivisions defines the spacing of the minor gridlines

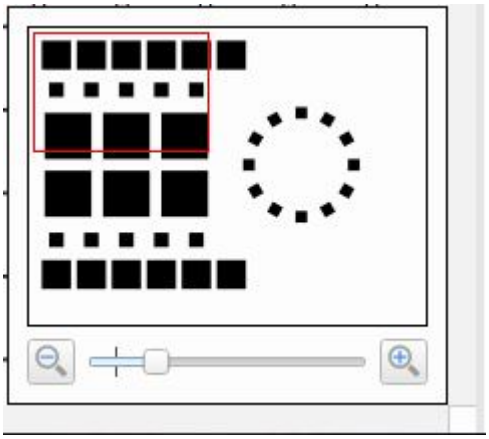
Show grid makes the grid visible or invisible

Snap to grid defines whether fixtures should automatically snap to the grids intersections.

Show Minimap

A minimap can be used to help navigate large complex layouts.

The minimap allows you to zoom in/out and move around the layout while showing a smaller version of the layout:



Instances

Instances can be used with Layouts to add a single fixture to the Layout multiple times on different layouts. This allows the programming to be seen on all layouts where the fixture exists.

Instances can also be used where multiple fixtures are to be controlled by the same control channel. Adding an instance in this case will ensure the fixtures simulate correctly to produce the most accurate visualisation.

Managing Instances

Creating Instances

There are various ways to create fixture instances:

- Right click > New Instance will create a new instance on the same layout.
- Copy and Paste as Instance (available in right click menu or Ctrl+Shift+V keyboard shortcut) can be used across different layouts.
- Using mouse and keyboard any selected fixtures can be pasted within the same layout by doing the following: left click one of the selected fixtures (ensuring to not move the mouse) then hold down Ctrl+Alt and begin dragging, this will create new instances, once you have begun dragging you can release Ctrl+Alt to position them on the layout.
- When a layout is duplicated, as described above; upon clicking Duplicate within the Manage... menu a Duplicate Layout pop up will ask you to choose between creating a layout where the duplicated fixtures are either new fixtures or instances of the ones present in the layout being duplicated. Selecting the Create instances of existing fixtures will create instances of all the fixtures in the duplicated layer.

Removing Instances from the layout

Removing an instance is achieved in the same way as deleting a fixture, but all instances of a fixture can be removed without deleting the fixture from the project as a whole.

When deleting the final instance of a fixture, holding Shift will remove the final instance, but keep the fixture in the project.

Locating Instances

Within the fixture browser, a circle icon indicates the presence of an instance of each fixture on the current layout.

	No Circle	No instances of the fixture within the project
○	Single Circle (Grey)	One instance of the fixture, but not on the current layout
○	Single Circle (Black)	One instance of the fixture, on the current layout
∞	Double Circle (Grey)	Multiple instances of the fixture, but none on the current layout
∞	Double Circle (Black)	Multiple instances of the fixture where at least one is on the current layout.

Hovering over the instance icon in the browser will highlight the instance/s on the Layout and a tooltip will indicate the total number of instances and the number on the current layout.

VLC/VLC+ Layouts

If you are using a VLC/VLC+ in your project, it will be linked to a VLC Layout with the same name as the controller e.g. Controller 1.

This layout can be used in exactly the same way as a Normal layout to bring fixtures into the project and lay them out on the VLC/VLC+ Layout. The difference is that these fixtures elements relate to a single pixel in the real world and therefore are exactly 1 pixel in size.

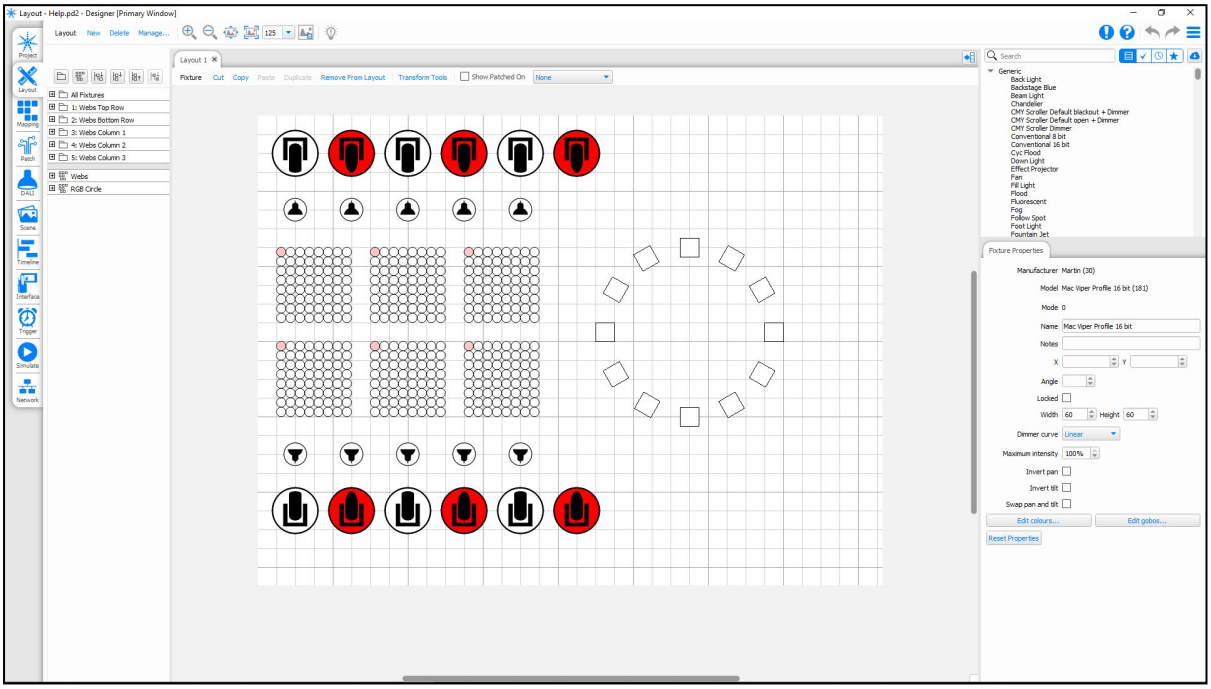
The VLC/VLC+ layout will be used to map any videos or matrix presets to, so the size of the layout should be set to the size of the content output e.g. a 100x100px installation should have a 100x100 VLC Layout.

The fixture pixels should be laid out to match the real world as this layout is used to map the programming onto the fixtures.

NOTE: The fixture library will only show fixtures that the VLC/VLC+ can support and the fixture browser will only show fixtures on the selected controller.

Adding and Organising Fixtures

Once you have the layout set up as desired you can start populating it with the fixtures as required for the installation:



Local Fixture Library

Pharos Designer ships with a limited Fixture Library which should be enough to get started with most simple shows. The Fixture Library is grouped by manufacturer and then sorted in Alphabetical order. A generic manufacturer is provided for standard fixtures such as Dimmers, basic RGB LEDs etc. and a Generic DALI manufacturer contains DALI personalities.

The library is fully searchable, so if you don't know the manufacturer of a fixture you can find it by searching for the fixture name.

Library Groupings

In addition to the main library, which allows you to search for any of the fixtures in your local library.

Used

The Used view shows any of the fixtures that you have used in the current project, to easily reuse existing fixtures.

Recent

The Recent view shows fixtures that you have used in any project on your computer recently.

Favourites

You can Favourite fixtures or manufacturers that you want to always have easy access to from within the Library view. To Favourite a fixture or manufacturer, Click the Star icon in the library.

Deleting Downloaded fixtures

If you have downloaded fixtures and no longer require them in your library, you can delete either individual fixtures or whole manufacturers by right clicking on the fixture or manufacturer and selecting Delete.

This will remove the fixture or manufacturer from the library.

(Legacy) fixtures

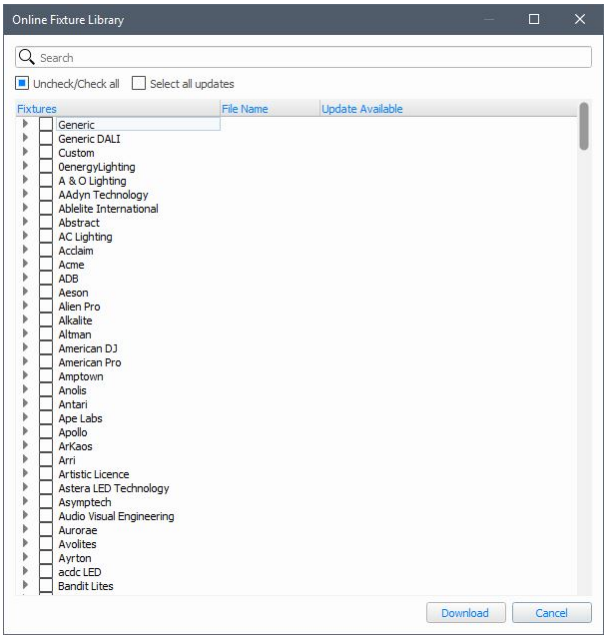
Some fixtures within the library have a (Legacy) suffix. This means that there is a version of this fixture that supports Direct Colour (on the LPC family of controllers).

The legacy fixtures automatically calculate values for White and Amber (where necessary), whereas the Direct Colour equivalents allow these values to be set explicitly.

Online Fixture Library

If you can't find a specific fixture in the Fixture Library, you may find it in the more comprehensive Online Fixture Library (accessed from the button next to the Search box).

Adding fixtures from the Online Fixture Library:



The Online Fixture Library contains all the fixture personalities currently available, to add these personalities to your system:

1. Find the fixture(s) that you want to download, either using the search box or by navigating the folder tree
2. Select the fixture(s)
3. Click Download

This will download the personality to your system and make it available within your local Fixture Library.

When you download a fixture it will be available in all future projects, and can be used offline.

If a fixture personality has been updated since being downloaded, the Update Available column will be checked to indicate that a new version is available. This may be due to a change in the firmware of a fixture or an error in the previous personality.

NOTE: If you need a fixture personality which isn't available in the Online Fixture Library, please contact support.

Fixture icons & scale

The following icons are used to differentiate between fixture classes:


Moving light - wash


DALI ballast (see DALI)


Moving light - spot


Conventional fixture


Moving light - mirror


Non-dim (switched control channel) or controller


Accessory (eg. scroller)


Media server


Discrete LED fixture (to scale)


Compound LED fixture (to scale)


Fountain jet fixture

The LED and compound LED fixture icons are drawn to scale (1cm:1pixel) so that, coupled with a correctly scaled background image, the resulting layout and simulation is as realistic as possible. The other icons are drawn to a standard size that, in most cases, will produce a realistic result. All placed fixture icons can however have their size (scale) and even shape modified using the Fixture Configuration pane.

When using the Simulator these icons instead render the fixture's output, even displaying the selected gobo and iris settings for moving lights. Fountain jet icons do not simulate intensity, but rather fill the icon to mimic the water jet.

Populating the layout

Simply choose a manufacturer, select the required fixture by clicking on it and then placing the fixture on the layout, it will automatically be added to the Browser and grouped with all other fixtures of that type. Once placed, left click to select it, a red highlight will indicate the current selection, see selecting fixtures. Right click to delete, group or duplicate fixtures.

To add a fixture:

There are several ways to add a fixture to the layout:

1. Click on the fixture in the library and then click to place the fixture on the layout (the layout border will turn blue to indicate you are in fixture placement mode)
2. Click and drag the fixture onto the layout and release the mouse button to drop it (it will automatically be added to the Browser)

When using Fixture Placement mode, you can pre-set any parameters for the fixture before placing it on the layout.

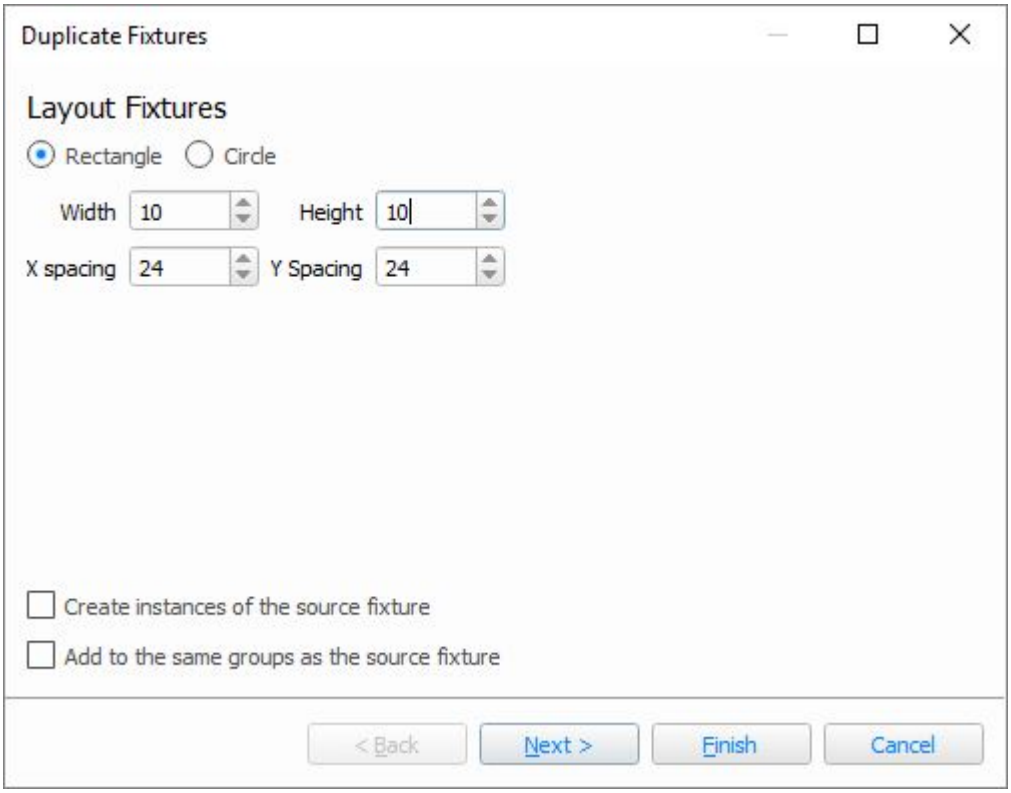
To add multiple fixtures

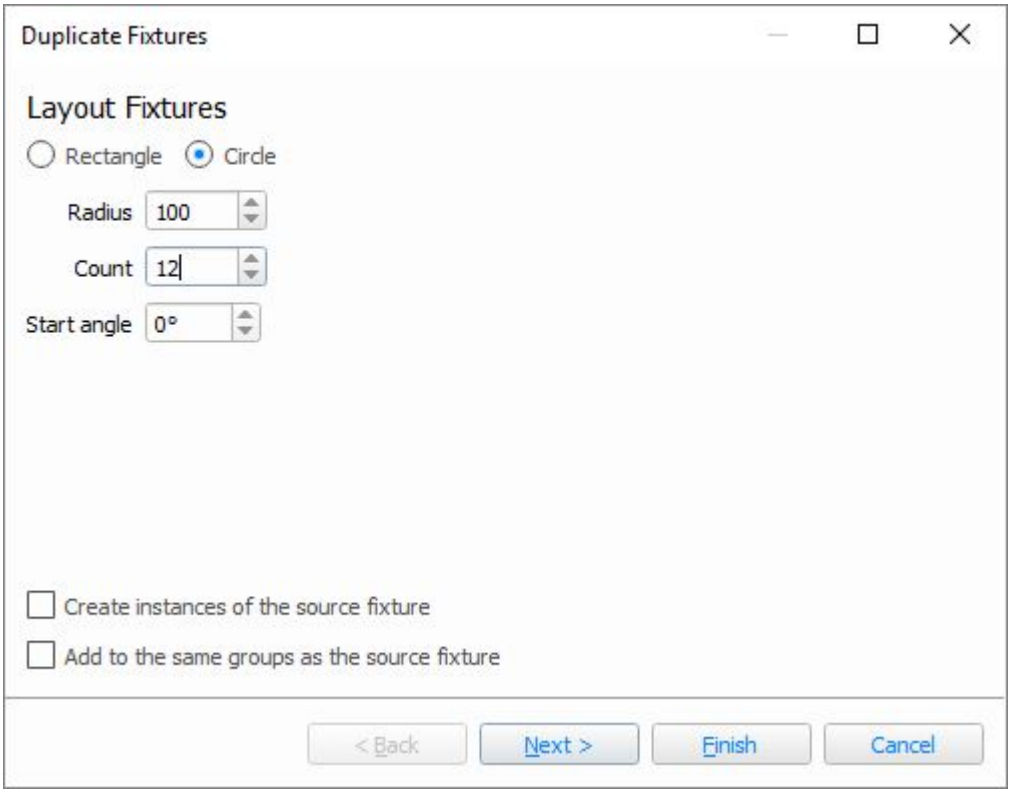
You can continue placing fixtures once they are selected (as above) if the Auto Finish option is not checked, this can also be toggled with Ctrl.

Click Finish to exit fixture placing mode.

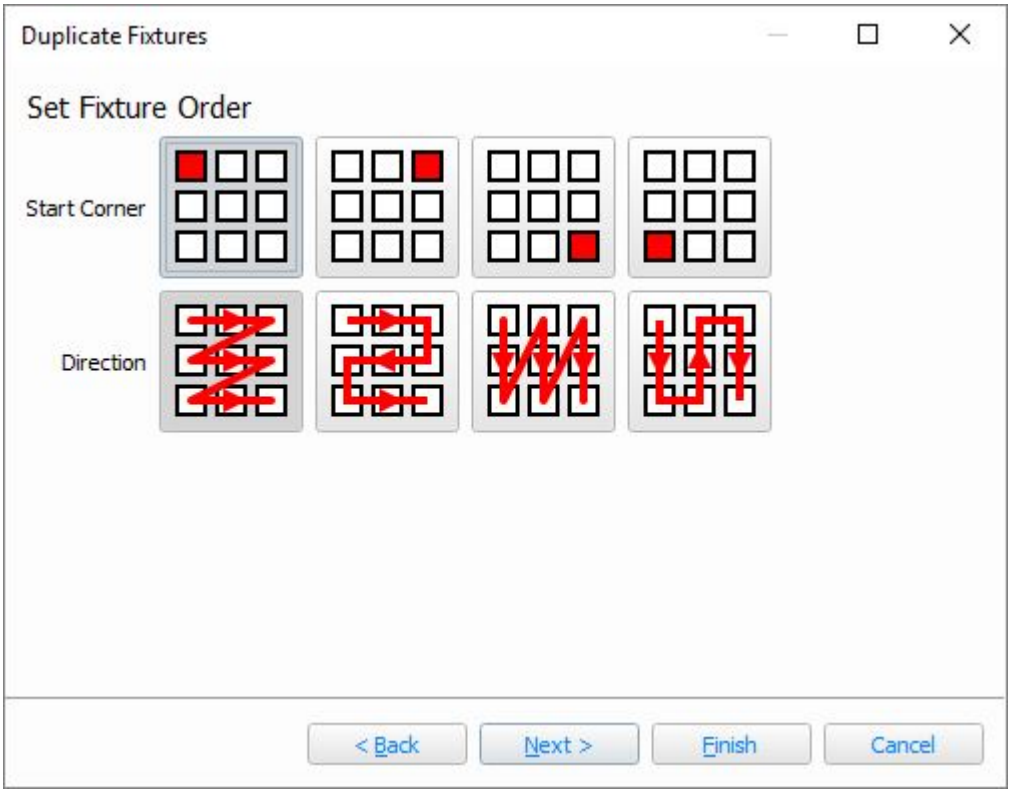
To duplicate a fixture (create an array):

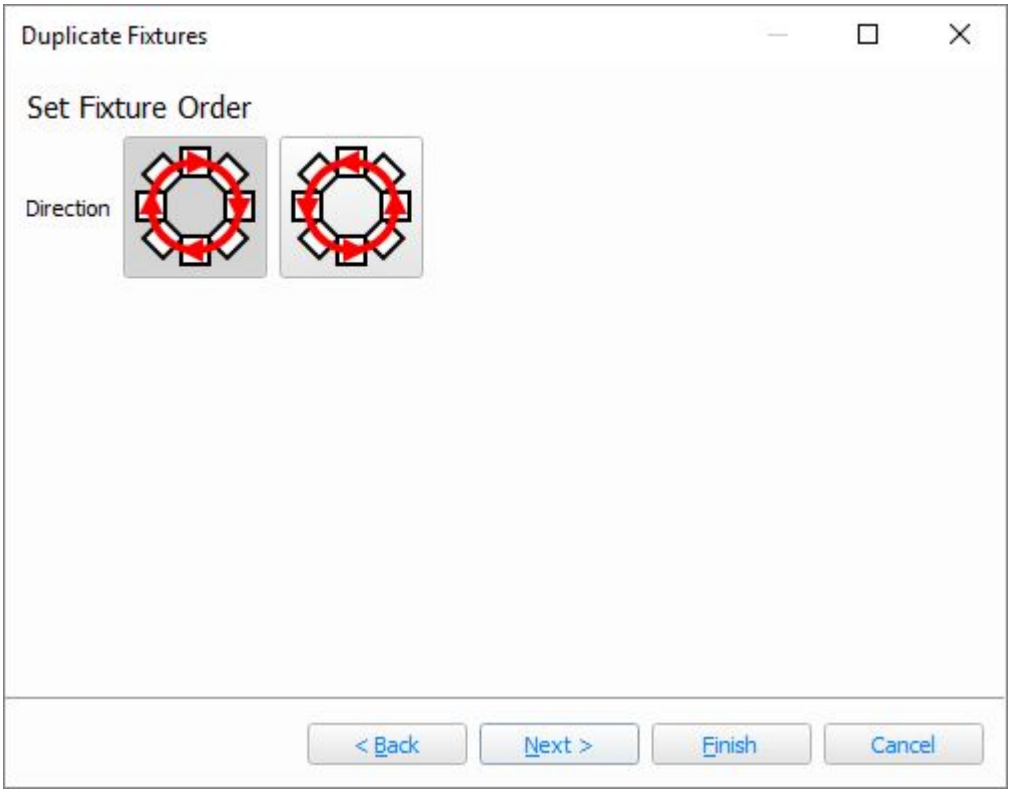
1. Right-click on the fixture (on the layout not the Browser) to be duplicated
2. Select "Duplicate"



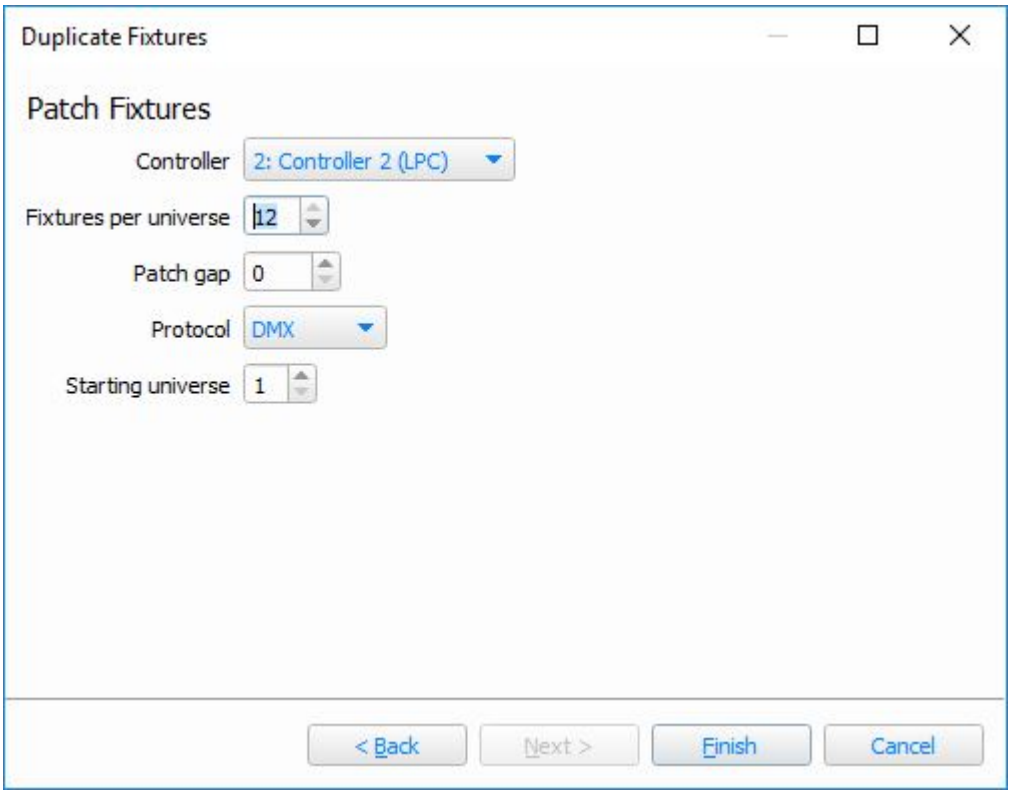


3. Select either "Rectangle" or "Circle"
4. Set the duplication parameters, see below
5. Press Next to set Fixture order or Press Finish to close the Window





6. Select the order of the fixtures in the new array, based on the start position and the direction.
7. Press Next to set Auto Patch or Press Finish to close the Window



8. Auto Patch can be used to patch a specified number of fixtures to a certain specified protocol, starting at a specified universe number with a specified patch gap. By default, Controller is set to none, so select the controller that you want to patch too.
9. Press Finish

For rectangular arrays, positive width and height values will place the copies to the right and below respectively, negative to the left and above.

For circular arrays, select the radius, count (number of fixtures) and start angle- complete circles are created in this way so, if arcs required, just delete those fixtures that are unwanted. The start angle determines where on the fixture the original fixture sits (0^o is at 3 o'clock).

Either array type also allows you to:

Create instances of the source fixture - This will cause all the duplicated fixtures to be instances.

Add to the same groups as the source fixture - This will cause the duplicated fixtures to be automatically added to all the same groups as the original.

To copy a fixture or fixture selection:

1. Select the fixture(s)
2. Press and hold Ctrl (Cmd)
3. Drag the copy to a new location on the layout and release the mouse button to drop (with multiple fixtures, their relative layout and numbering is preserved)

NOTE: pressing Ctrl *after* starting to drag will cause the selection to jump back to its original position and create a copy of the selection under the pointer.

Alternatively, Cut, Copy and Paste functionality is provided on the Fixture Manipulation Toolbar and the right click menu.

To delete a fixture or fixture selection:

1. Select the fixture(s)
2. Press Delete or right-click > Delete

Note that the fixture(s) will be completely removed from the project and all programming discarded.

To see where a fixture is patched:

1. Check 'Show patched on' on the toolbar.
2. Move the cursor over a fixture - the fixture's patch will be shown next to the cursor.
3. Pick a Controller from the drop down list on the toolbar to see all fixtures patched to it - fixtures patched to the Controller will be shown in blue.

To change a fixture's type:

It is possible to swap a fixture for a different type:

1. Select the fixture/s that you want to change
2. Right-click on a selected fixture
3. Choose Change Fixture Type
4. Select the fixture in the library that you want to change to

NOTE: Changing to a fixture that uses more channels will require patched fixtures to be unpatched, changing to a lower channel count will retain the start address, and leave spaces between the fixtures.

DALI fixtures

DALI fixtures/ballasts are added to the layout in the same way as all other fixtures but they do not populate the Browser and no groups are automatically created since DALI fixtures are programmed and controlled via dedicated DALI Interfaces, see DALI.

Import fixtures

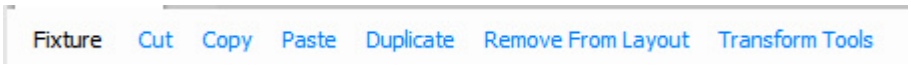
You can use Main Menu > Import Object to import a fixture layout from a CAD application via a delimited text file, or the Philips Color Kinetics Video Management Tool, see Import Objects.

Export Layout

You can use Main Menu > Export Object to export a fixture layout to a CAD application via a CSV file, see Export Objects

Fixture Manipulation

The Fixture Manipulation Toolbar can be used to perform some manipulation functions on fixtures and groups of fixtures.



Cut, Copy and Paste

Make copies of fixtures to paste on the same or a different layout. Cut can be used to move fixtures from one layout to another.

Duplicate

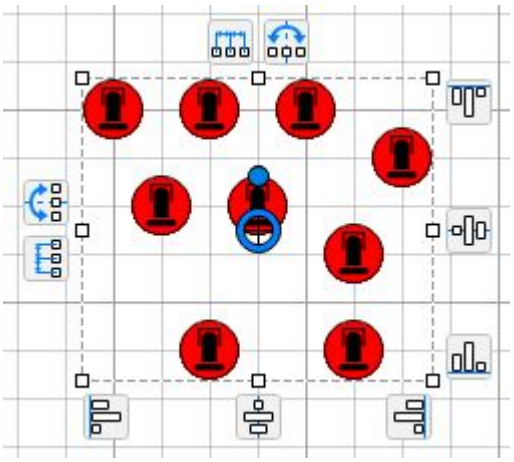
Create arrays of fixtures (see above)

Remove From Layout

Delete the fixture from the current layout, but keep all other instances.

Transform Tools

Selecting Transform Tools with multiple fixtures selected will activate the on-layout transformation options.



Scale selection

The layout of the selected fixtures can be scaled in the X and/or Y direction using the handles round the edge of the selection box.

Reflect 

Mirror the selected group of fixture across the specified axis. This will move the selected fixtures, not create a new set of fixtures.

Rotate 

Rotate the selection, keeping the relative positions the same.

The angle is set by clicking and rotating the lever on the rotation handle.

The centre of rotation can be moved by dragging on the centre of the rotate handle.

Align 

Align the specified part of the selected fixtures. Options:

- Align Left edges
- Align Centres
- Align Right edges
- Align Tops
- Align Middles
- Align Bottoms

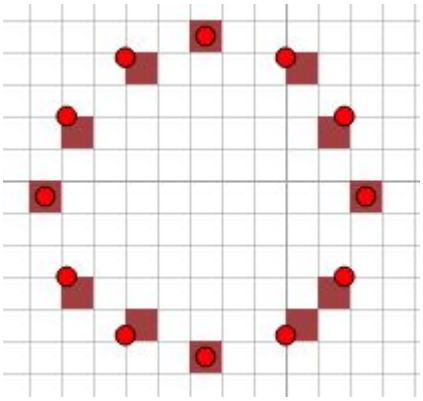
These options are available round the right and bottom edges of the selection box.

Distribute 

Distribute the selected fixtures between the extreme fixtures either vertically or horizontally. Use this to create evenly spaced groups of fixtures where the spacing isn't known.

Show VLC Fixture Centres

On a VLC/VLC+ layout, checking this option will display the centre of the fixture, which will then be mapped to a pixel on the layout. This maintains the relative positions of the fixtures when they don't map directly onto the pixels.



Fixture Properties

When fixtures are selected from the Layout or the Browser, the fixture configuration section will be populated. If you have multiple fixtures selected, any shared properties will be displayed.

Fixture Identification

With a fixture selected the top two fields detail the fixture's manufacturer (manufacturer id) and model (model id), they are for reference only and can not be edited.

Number and Name

Here you can enter a new name for the fixture, useful to help make the browser easier to navigate, and the means to change the fixture's unique user number.

Every fixture added to the project is assigned a user number which is used as a shorthand method of selecting it, using the web interface's command line for example. Use the up and down arrows to change the number but note that only available numbers are shown so you may need to change the number of another fixture first to make that number available. Note that the user number does not affect the order of the fixtures in the Browser and thus the order used for transitions.

Notes

Below this are two fields for entering any comments about the fixture, useful for annotating the project's documentation. These comments will appear in the fixture report and in exported fixture plans.

X and Y

Use these fields to set numerically the fixture's position on the layout

It is desirable to position the fixtures on the layout as accurately as possible to improve both the accuracy of the programming (in particular pixel matrices created automatically from the layout) and the general neatness of the project and simulation.

To nudge a fixture on the layout you can use the cursor keys to nudge the fixture selection up, down, left or right by the amount set as the grid spacing. Alternatively, use the up and down arrows by each of the fixture's position fields. Holding Shift while using the cursor keys performs a fine-nudge of 1px.

Angle

Set the angle of the fixture, setting these accurately will allow Pixel Matrices and Simulate to be as accurate as possible.

Locked

Check this box to prevent the fixture(s) being moved using the mouse. Locked fixtures are still included in drag selections and can be moved using arrow keys.

Width and Height

Use these to set the size of the fixture on the layout (1px : 1cm), this is particularly useful when the background image is not to the same scale.

Shape

LED fixtures can be set to be circular or square to match the physical fixture, and this option can be used to change between the two.

Gel colour

For those working with gelled lights it is possible to simulate the gel's colour so that the fixtures are rendered correctly, press the Gel button and select the required colour via the colour picker.

This can also be used when a fixture is a single colour as these fixture personalities will be simulated as white.

Intensity

The fixture's dimmer curve and maximum intensity can be set, use the Dimmer Curve drop-down to determine the type of cross fade the intensity channel will perform and set a Maximum Intensity level, useful for balancing light output.

DALI

DALI ballasts can be configured as allowed for by the DALI standard; Min Level (0>254), Max Level (0>254), Power On Level (1>254) and Bus Failure level (No change, 0>254). The standard specifies a level range of 0>254 with 255 being used as a special case meaning "no change", a mask if you like. Unlike DMX fixtures, these settings are stored in the ballasts themselves and so must be uploaded separately, see DALI.

Ballasts can also have their default Fade Time and Fade Rate set in the configuration pane.

Emergency DALI ballasts will also have the option to the set the Prolong time in the configuration pane. More information about emergency ballasts can be in the DALI topic.

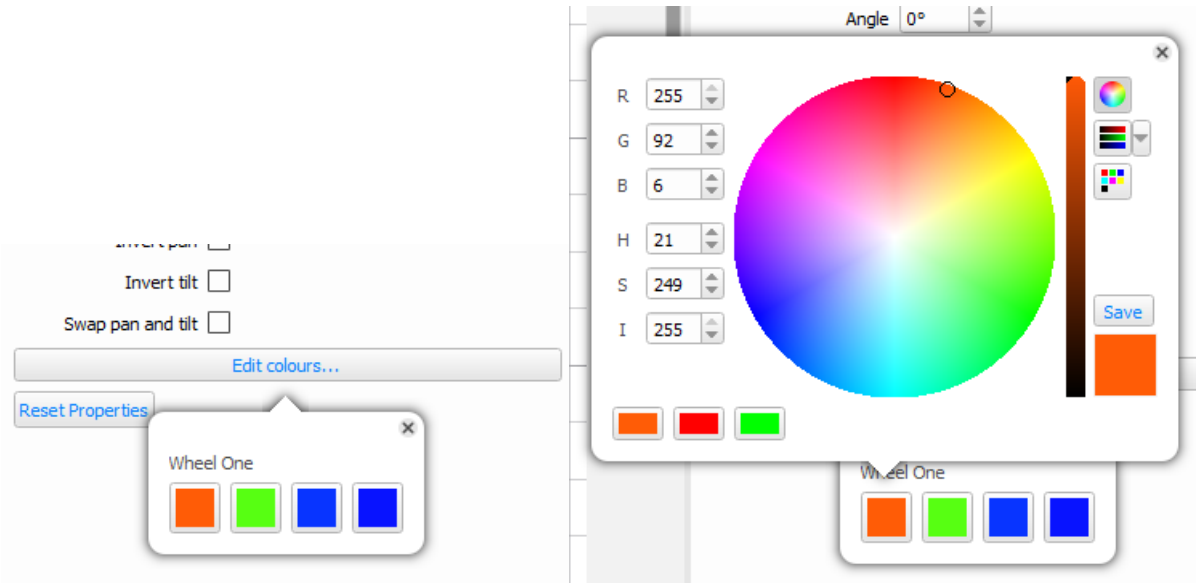
NOTE: The default Fade Time and Fade Rate will be overwritten when new values are sent to ballasts during playback from triggers or programming. This is due to the way DALI ballasts store this information.

Moving lights

Moving lights can be customised for the project as one would on any sophisticated moving light console. Use Invert Pan, Invert Tilt and Swap Pan & Tilt to normalise the way they respond to the position controls.

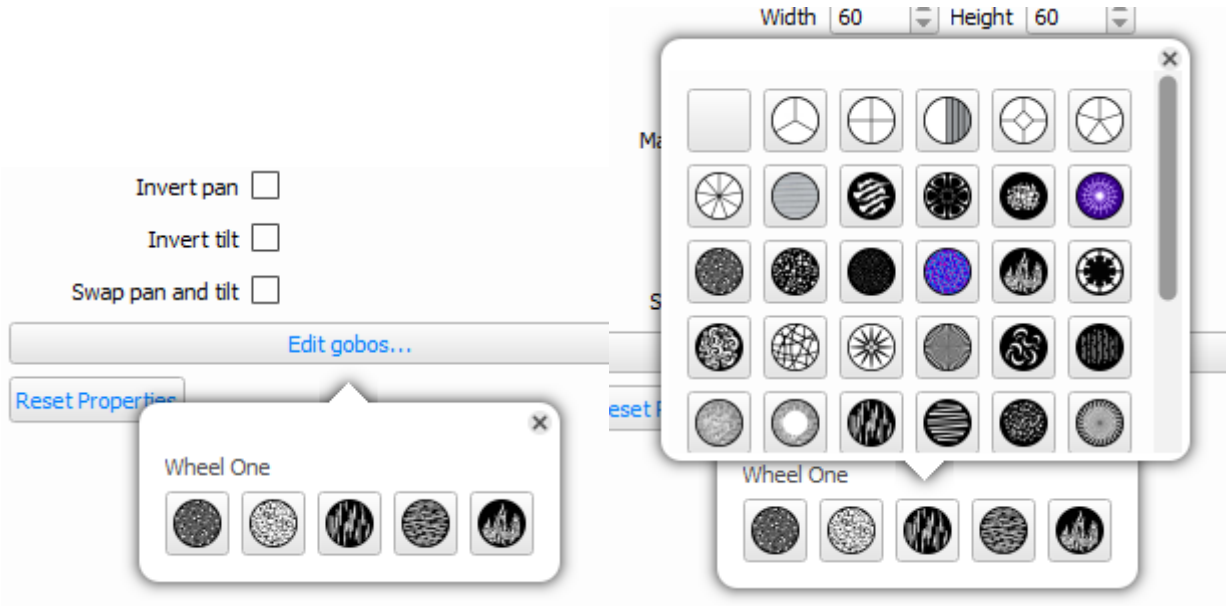
Edit Colours

Fixtures with colour wheels will display an Edit colours... button within the Fixture Properties tab. Clicking it will show the current colours set for the available wheels. Clicking any of the wheels will open the colour picker allowing you to choose the colour matching the colour wheel that is or will be located in that fixture's colour wheel.



Edit Gobos

Fixtures with gobo wheels will display an Edit gobos... button within the Fixture Properties tab. Clicking it will show the current gobos set for the available wheels. Clicking any of the wheels will open the gobo image picker allowing you to choose the gobo image that matches gobo wheel that is or will be located in that fixture's gobo wheel.



Custom gobo images can be added to the following directories:

- Windows: %userprofile%\Documents\Pharos\Designer 2\Gobos
- Mac: ~/Documents/Pharos Controls/Designer 2/Gobos

Requirements for creating custom gobo images:

- Only .png images are accepted.
  - Transparency is preserved.
  - Colour is allowed.
- The default images are 128 by 128 pixels, but larger images are accepted. It is recommended to keep them with a square aspect ratio for clear viewing of the image within the gobo wheel selection window.

NOTE: For new gobo images to be detected the software must be restarted.

Reset Properties

You can return a fixture to its library definition, losing local changes and thus restoring it to its defaults by selecting Reset Properties. This is useful for updating fixtures on the layout with any library definition edits, forcing a redraw. Local changes to a fixture's geometry (shape, size) will be overwritten.

You can also Reset all fixtures of a type from the Used section of the fixture library, by right clicking the required fixture.

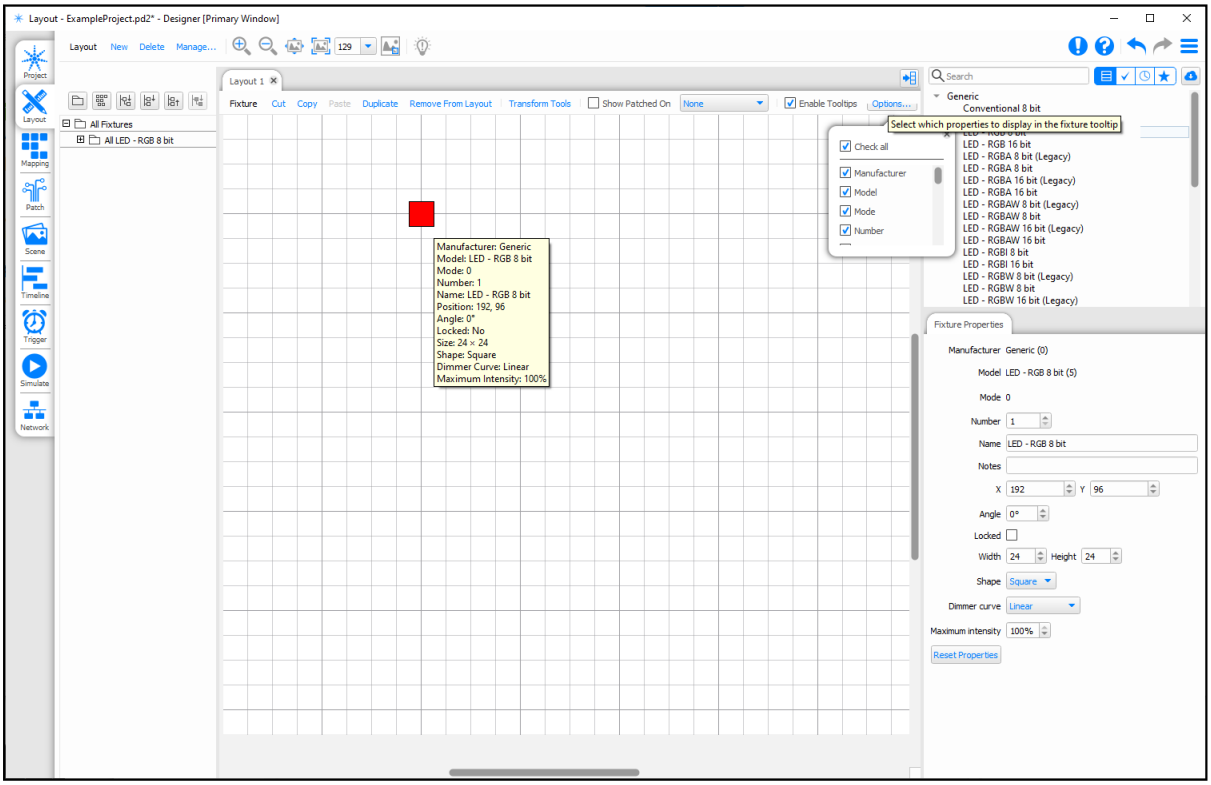
Show Patched Status

To see which fixtures are patched to each controller in a project, you can check the Show Patched On checkbox at the top of the fixture browser. The dropdown can then be used to select the controller to display (None will show fixtures which are unpatched)

This will highlight the fixture on the layout in the "programmed" colour (default: blue). See Project Properties.

Tooltips Status

To enable the fixture tooltips feature select the checkbox at the top of the fixture browser called Enable Tooltips, which provides Options in a popup that contain all the information available regarding a fixture, including “Patch Information".



Selecting Fixtures

Keyboard Shortcuts

Layout

Ctrl+N	Create a New Layout
Ctrl+D	Create a duplicate of the current layout
Ctrl+I	Show layout properties
Ctrl+A	Select all fixtures
Double-left-click *on a fixture*	Select all instances of the fixture
Ctrl+left-click *on a fixture*	Toggles its selection
Alt+left-click *on a composite fixture*	Select an element of a fixture
Alt+left-click *on background* + drag	Select elements or fixtures using a lasso
Alt+left-click *on fixture* + drag	Select elements or fixtures using a lasso
Left-click *on background* + Alt + drag	Select whole fixtures using a lasso
Left-click *on fixture* + Alt + drag	Constrain movement to one axis (horizontal or vertical directions)
Shift while selecting fixtures with a box	Selection order based on position, otherwise based on fixture number
Tab	Select the next fixture by number
Shift+Tab	Select the previous fixture by number
Ctrl+left-click *while in add fixture mode (blue border)*	Toggle the behaviour of Auto-finish
Alt+left-click *while in add fixture mode (blue border)*	Add an instance of the last added fixture (or a new fixture if no fixture is added yet)
Escape *while in add fixture mode (blue border)*	Finish adding fixtures
Escape *otherwise*	Toggle last fixture selection
Ctrl+drag	Create duplicates of the selected fixtures
Ctrl+Alt+drag	Create instances of the selected fixtures
Shift *while dragging fixture/s*	Disable fixture snapping
Delete/Backspace	Delete selected fixtures
Shift+Delete/Backspace	Delete selected fixtures from the Layout but keep the fixture in the project, even if they no longer exist on a layout
Ctrl+Delete/Backspace	Delete selected fixtures from the project
Shift+ 'Remove From Layout'	Delete selected fixtures from the Layout but keep the fixture in the project, even if they no longer exist on a layout
Ctrl+X	Cut the selected fixtures
Ctrl+C	Copy the selected fixtures
Ctrl+V	Paste fixtures from the clipboard
Ctrl+Shift+V	Paste instances of fixtures from the clipboard
Up/Down/Left/Right	Nudge the selected fixtures by the grid spacing, if show grid is turned off, nudge will default to a 1-pixel nudge
Shift+Up/Down/Left/Right	Nudge the selected fixtures by 1 pixel
Space+drag	Pan the view
Ctrl+0	Reset the zoom

Ctrl+F	Zoom to fit the window
Ctrl++	Zoom in
Ctrl+-	Zoom out
Ctrl+ mouse wheel	Zoom in and out
Middle-click + drag	Zoom into the drawn rectangle
Left-Click + drag + Shift	Pressing Shift after starting a lasso selection will sort by the aspect ratio (see here)
Alt+ mouse wheel (Shift+ mouse wheel)	Scroll Horizontally
Ctrl+drag *on Transform tool drag handle*	Maintain aspect ratio of selection
Alt *while dragging fixture/s*	Lock movement to a single axis

Browser

Delete/Backspace	Delete selected fixtures, groups or pixel matrices
Ctrl+left-click	Select multiple fixtures, groups or pixel matrices
Shift+left-click	Select all fixtures, groups or pixel matrices between two selections.
Alt+left-click	Deselects the contents of the group/pixel matrix
Up/Down	Move current row indicator up and down, and select the row
Shift + Up/Down	Move current row indicator up and down, and add the row to the selection
Ctrl + Up/Down	Move current row indicator up and down, but don't change the selection
Left/Right	Collapse/Expand current group
Space	Select current row
Ctrl + Space	Add current row to the selection

Browser

The Browser is the most powerful and flexible method of selecting fixtures. Click on a group heading to select all fixtures within the group, expand a group by clicking on the plus sign and click on fixtures within to select individual fixtures and, with compound fixtures, expand them to select the individual elements within. Fixtures and elements are shown in red (by default) when selected.

Hold down Shift while clicking to select all contiguous groups/fixtures/elements between clicks and hold down Ctrl (Cmd on Mac) while clicking to select multiple non-contiguous individual groups/fixtures/elements. Hold down Ctrl (Cmd) while clicking to deselect a selected group/fixture/element.

Clicking “in space” (anywhere on the Browser that isn’t a fixture) clears the selection.

The Browser also provides the interface to view and change the ordering of fixtures/elements within groups. This order is used by the application to determine cue timing and effects skews, simply drag fixtures about within the Browser to change this order.

Layout

Only fixtures and elements can be selected using the Layout, to select groups you must use the Browser. Fixtures and elements are shown in the colour set within Layout Properties (default is red) when selected.

Ctrl (Cmd) work with clicking as described above to select/deselect and you can also lasso fixtures by clicking and dragging around them, fixtures must be wholly enclosed to be selected.

When lassoing fixtures, by default a group created from the selection will be in numerical order, but holding Shift after you have started dragging will cause the group order to be defined by the position of the fixtures within the lasso. The rule for this being defined by the lasso's aspect ratio at the time the Shift key was pressed. The cursor will change to describe the selection mode:

Example:

1 6 11 16 21
2 7 12 17 22
3 8 13 18 23
4 9 14 19 24
5 10 15 20 25

All the fixtures are selected with a lasso from the top left and the cursor has a right-down arrow  , the fixtures will be selected in the following order:

1,6,11,16,21,2,7,12,17,22,3,8 etc.

Hold down Alt to select individual elements within compound fixtures. Hold down Alt and Ctrl (Cmd) to select/deselect multiple elements.

Clicking “in space” (anywhere on the Layout that isn’t a fixture) clears the selection.

Pressing Esc will toggle the previous fixture selection.

Select next/previous fixture

With a single fixture selected, the Tab key will select the next fixture (next higher fixture number) and Shift + Tab will select the previous fixture (next lower fixture number).

Select all fixtures

Ctrl + A will select all fixtures.

Selection Modes

The Right click menu allows access to various selection modes:

Normal: overrides selection with selected/lassoed fixture/s

Additive: adds the selected/lassoed fixture/s to the selection

Subtractive: removes the selected/lassoed fixture/s from the selection

Invert: deselects selected/lassoed fixture/s that are selected, and selects selected/lassoed fixture/s that are deselected

Context Menu

A context menu can be accessed by right clicking on a fixture, or the layout.

Be aware that when fixtures are selected the behaviour can be different.

- Right-clicking on a selected fixture will never change fixture selection; context menu actions apply to current fixture selection.
- Right-clicking on an unselected fixture will always clear the current selection and select the single fixture that was right-clicked. Context menu actions apply to fixture that was right-clicked (which is now the only selected fixture).
- Right-clicking on the layout never changes fixture selection, but will give access to the layout context menu.

Groups

Non-VLC Groups

Groups are an important concept to grasp as they serve multiple purposes:

Firstly, as you will see later, it is the rows of the Browser that make up the rows of the Timeline interface thus it is convenient to gather fixtures/elements that are to be programmed together into a group to simplify this procedure.

Secondly, as the order of fixtures/elements within a group determines how programming and timing is rendered, it is sometimes useful to make multiple groups of the same fixtures with different ordering.

Thirdly, groups can be used to patch fixtures, as they are a list of the fixtures in order.

Finally, Groups can be used to set up intensity and RGB control zones in the Triggers window.

To create a group:

1. Select the fixtures you want to group using the Browser, the Layout or both - the order you do this in determines the order within the group
2. Right-click a selected fixture and choose New Group or press the New Group button at the top of the Browser
3. Name the group which has been created in the Browser containing these fixtures

NOTE: if you are creating a group of fixture elements, you must continue holding Alt when you right click to create the group.

Alternatively:

1. Press New Group with no fixtures selected
2. Name this empty group
3. Drag fixture/element selections into the group from within the Browser - the order you do this in determines the order within the group

Add fixtures to a group

1. Select the fixtures you want to add to an existing group using the Browser, the Layout or both - the order you do this in determines the order within the group
2. Right-click a selected fixture and choose Add to group
3. Select a group from the list

The selected fixtures are added to the end of the chosen group. If some fixtures in the selection were already in the chosen group, they are ignored and remain at their current positions in the group.

VLC/VLC+ Groups

On the VLC range of controllers, groups are used in a limited way to only be used for patching purposes. These groups are created in the same way, but are only present in Layout and Patch.

To Re-Order Fixtures in a Group

The order of fixtures in a group directly impacts the programming applied to them and the order that the fixtures within them get patched.

When creating these groups, sometimes it can be possible to create them in an order that is not optimal.

If you right-click on the group, you can select Re-order to automatically sort the fixtures in the group numerically, or in order horizontally or vertically.

Group Properties

Right-click on any group to edit its properties. This includes name, number and colour. Selecting a colour will show that colour in various places throughout Designer, including in timeline rows and in triggers which support group selection.

Customising Fixtures

There are multiple ways to customise fixtures:

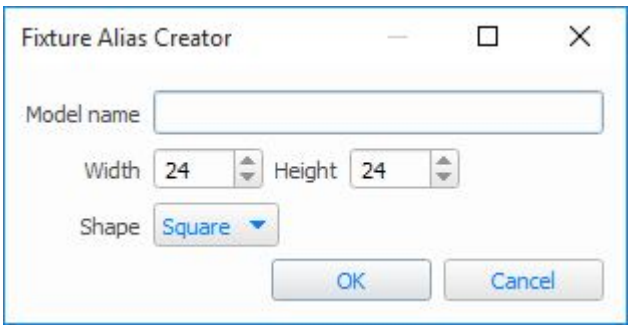
- Fixture alias
- Custom Fixtures
- Fixture Templates

Fixture Alias

If you require a fixture which doesn't exist within the Library or Online Fixture Library, but is similar to an existing fixture (e.g. RGB LED), you can create a fixture with the same DMX footprint but different simulation options. To do this:

- Select the fixture in the Library
- Right click on the fixture
- Select Create Fixture Alias

Here you can customise the Alias fixture with a different Name, Size and Shape.



Custom Fixtures

Should a completely custom fixture be required, this can be created based on an existing fixture by right clicking the fixture in the library and selecting Create Custom fixture.

For more information on the fixture personality syntax, see Custom Fixtures, or contact Pharos Support.

Fixture Templates

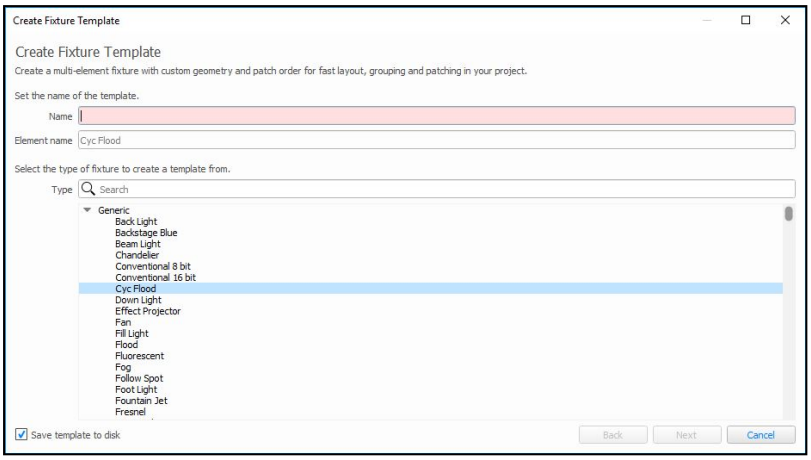
Fixture Templates allow you to create an array which contains a number of fixtures as elements in a predefined layout.

This allows you to create a reusable set of custom composite fixtures.

To Create a Fixture Template

Right click on a fixture in the library to create the template from that fixture, or right click in white space in the library to choose within the wizard.

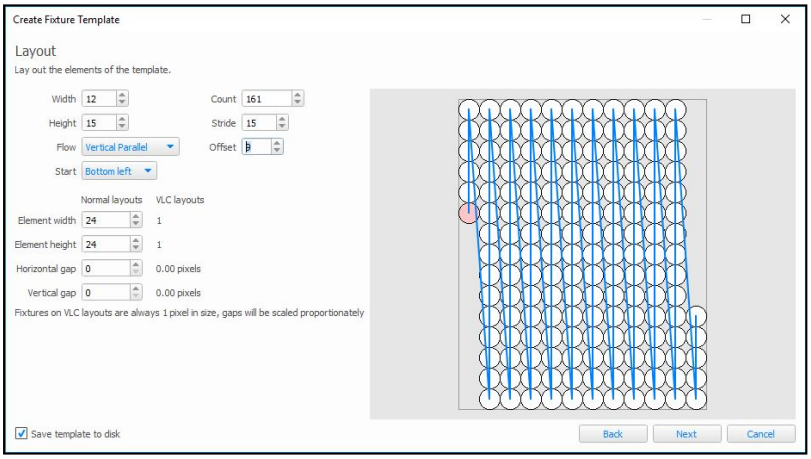
Setup Template Properties



Enter a name for the template, and optionally set an element name.

Select the fixture to use as the elements within the template.

Setup Element Layout



Select the array parameters to create your template array:

Width/height - The size of the template's array

Flow - The order of the elements in the template, linked to the patch order of the elements within the fixture.

Start - The start point of the template order

Element width/height - The size of each element within the template

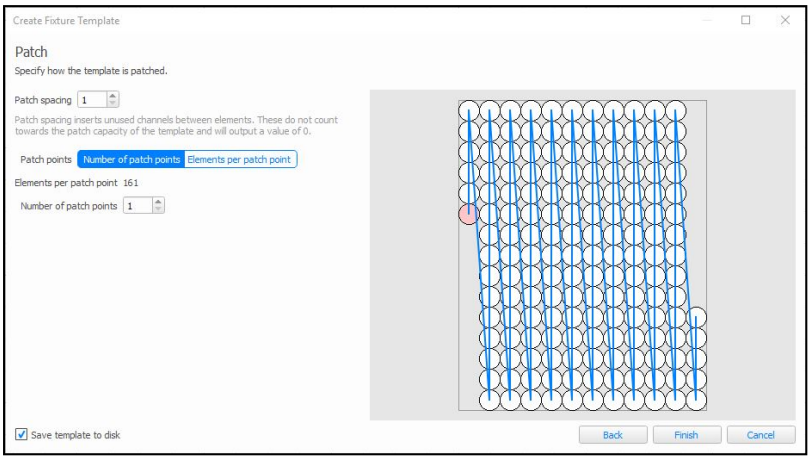
Horizontal/vertical gap - The spacing between each column/row

Count - The number of elements in the template

Stride - The number of elements in the direction of the Flow, adjusts the shape of the array without changing the fixture count

Offset - The offset from the Start that the first element should take in the direction of the Flow.

Setup Patch points



Patch spacing - The number of channels between the first channel of one element and the first channel of the next element.

Elements per patch point - The maximum number of elements in each patch point. This is limited to 1 universe worth of output channels (e.g. 170 x 3 channel fixtures, 128 x 4 channels fixtures etc.)

Number of patch points - The number of patch points that the template should consist of.

A patch point is a section of the fixture that is patched in one go (e.g. multiple strings of nodes). Each patch point can be a maximum of 512 channels.

To Edit a Template

A template can be edited by right-clicking on it and selecting Edit Fixture Template. It can be duplicated before editing when slight changes are required.

When editing a template, the element count and name must remain the same.

If a template has already been used within a project before it is edited, it must be Reloaded from the Library to update to the edited version. This is done by going to the Used Library, right-clicking on the Template and choosing Reload from Library.

To Add a Temple to the Layout

Fixture templates are available within the Templates folder of the Fixture Library and can be added to a layout in the same way as any other fixture.

Custom Fixtures

A Custom Fixture is required if you want to change the properties of a fixture, or create a fixture with additional channels.

Below is a fixture personality for the Generic LED - RGB 8 bit fixture:

```
fixture = LED - RGB 8 bit

manuid = 0

modelid = 5

class = led

shape = square

dimensions = 24x24

patchgroup = fixture

parameter = Cyan

default = 255

crossfade = linear

type = ltp8bit

range = 255, 0, %

parameter = Magenta

default = 255

crossfade = linear

type = ltp8bit

range = 255, 0, %

parameter = Yellow

default = 255

crossfade = linear

type = ltp8bit

range = 255, 0, %
```

There are three main sections in the personality:

- Fixture Header
- Patchgroups
- Channel Definitions

Fixture Header

The fixture header contain all the information that relates to the fixture as a whole.

```
fixture = LED - RGB 8 bit

manuid = 0
```

```
modelid = 5
class = led
shape = square
dimensions = 24x24
```

Each line contains a property name and its value, separated by an equals sign (=).

fixture	Required	The name of the fixture (as displayed in the Designer fixture library)
manuid	Required	The manufacturer ID for the fixture manufacturer. This can be found from an existing fixture by that manufacturer. If the manufacturer doesn't exist, then 100 should be used (Custom)
modelid	Required	A unique modelid (within the manufacturer)
modeid	Optional	The mode identifier where there a multiple modes available for a single fixture
class	Required	The class of the fixture. This will determine how the fixture is displayed on the layout (accessory, controller, conventional, dali, led, media, mirror, spot, wash)
shape	Optional (only used if class = led)	The shape of an LED fixture (circle or square)
dimensions	Required	The size of the fixture on the plan (in px)

Patchgroups

A fixture can have 1 or 2 patchgroups within the personality. A patchgroup is a separate section of the fixture that can be patched independently, e.g. a fixture containing multiple strings of LED nodes, or where the intensity of a fixture is controlled separately to the rest of the parameters.

```
patchgroup = fixture
```

The label of the patchgroup will be displayed within the Patch mode of Designer when you patch the fixture.

Normally this can be left as `patchgroup = fixture`.

Channel Definitions

A channel definition contains all the information required to allow a single channel to work. Although a channel definition doesn't include a channel number, Designer will increment this for each new channel definition, so the channels should be added in the order that they need to be within the fixture.

```
parameter = Intensity
default = 0
crossfade = linear
type = ltp8bit
range = 0, 255, %
```

parameter	The main function of the channel. This should be one of the functions defined later in this document
default	The default level for the channel, this is what the channel will be set to if no Timelines or Scenes are controlling the fixture (0-255 or 0-65535)

crossfade	The crossfade path for the channel (linear or snap)
type	The type of channel (htp8bit or htp16bit). Sets whether the channel is an 8 bit (0-255) or 16 bit (0-65535) channel
range	See below

Range Declarations

A range declaration defines a DMX range or value which can be selected within Designer, and assigns a label to it.

Syntax:

```
range = int, label
```

OR

```
range = int1, int2, label
```

If a single value is set, then selecting the range in Designer will set the DMX output to that specific value. Setting two values will allow the user to select any value in that range.

int, int1, int2 = 0-255 for 8 bit channels, 0-65535 for 16 bit.

int1 may be more or less than int2, but cannot be the same.

The label is displayed within Designer on the button which is used to set the value.

If the label is set to %, then no label will appear.

Special Considerations

RGB Channels

Designer uses a CMY colour mixing engine internally, and as such RGB channels must be defined as their inverse colour with an inverted range:

Red:

```
parameter = Cyan
default = 255
crossfade = linear
type = ltp8bit
range = 255, 0, %
```

Green:

```
parameter = Magenta
default = 255
crossfade = linear
type = ltp8bit
range = 255, 0, %
```

Blue:

```
parameter = Yellow
```

```
default = 255

crossfade = linear

type = ltp8bit

range = 255, 0, %
```

Multi Element Fixtures

If you have a multi-element fixture (e.g. a series of RGB LED Nodes), the fixture personality can be configured to create those nodes for you.

Syntax:

```
element = label

      Channel Definitions

elementcount = value
```

The label will be displayed as the name of the element within Designer (in the fixture browser)

The value of elementcount is the number of repetitions of the element within the fixture.

You will require an additional line in the fixture header:

```
geometry = intX x intY
```

intX and intY are the number of elements the fixture has in the X and Y directions.

Example:

```
geometry = 2x4
```

Comments

Any line within a personality can be commented out using double forward slashes:

```
// This would be a comment
```

Unused Channels

If a channel in a personality is unused, it can be added to the personality to ensure the DMX footprint and channel numbering is correct without actually controlling anything.

```
constant = int
```

Constant defines this as a non-controllable channel, and int is the value to set this constant channel to, generally 0.

Example:

```
constant = 0
```

Saving a Custom Fixture

When you create a Custom Fixture manually, you will need to save it to the following location:

- Windows: %userprofile%\Documents\Pharos\Designer 2\Fixtures
- Mac: ~/Documents/Pharos Controls/Designer 2/Fixtures

The file should be saved as a *.txt file with any name of your choice.

You may need to restart Designer for your new fixture to appear in the fixture library.

Further Assistance

Should you require any further assistance, please send the following to your Dealer:

The name and mode of the fixture that is required

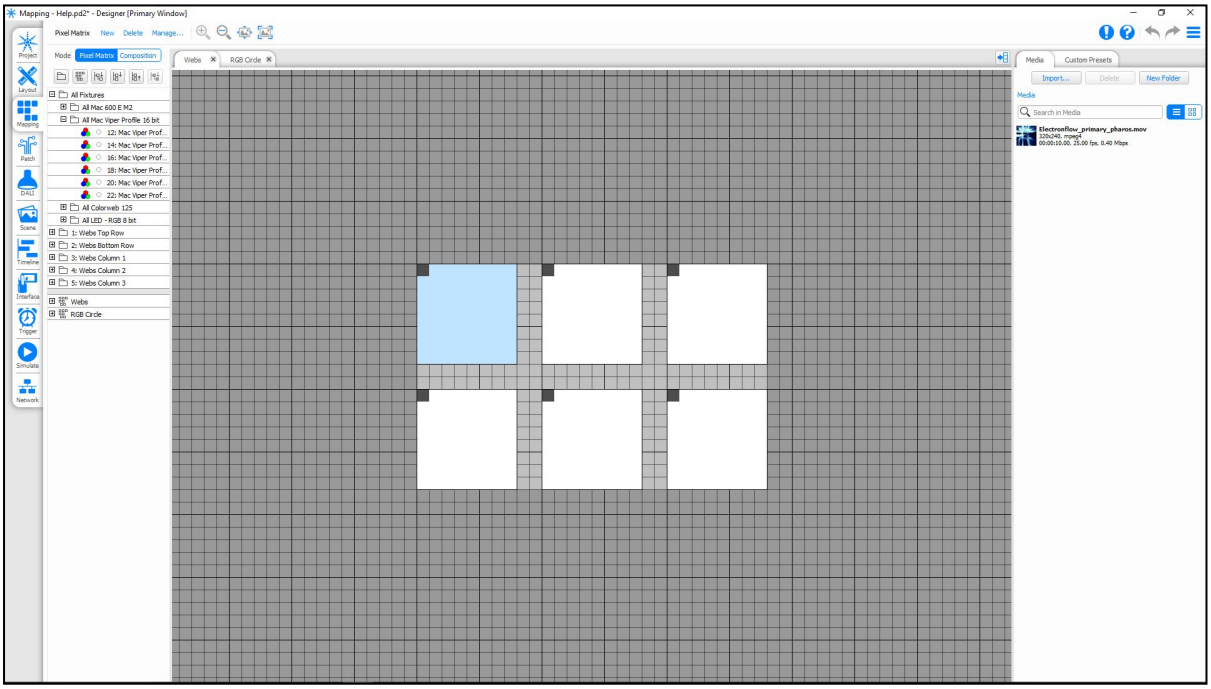
A user manual, including full DMX personality

The date you require the fixture personality

Pixel Matrix Editor

Keyboard Shortcuts

Ctrl+N	Create new pixel matrix
Ctrl+D	Duplicate the current pixel matrix
Ctrl+I	Show pixel matrix properties
Ctrl+C	Copy the selected media
Ctrl+V	Paste media from the clipboard into the current folder
Delete/Backspace	Delete selected media
Up/Down/Left/Right	Nudge the selected items by 1 pixel
Space+drag	Pan the view
Ctrl+0	Reset the zoom
Ctrl+F	Zoom to fit the window
Ctrl++	Zoom in
Ctrl+-	Zoom out
Ctrl+ mouse wheel	Zoom in and out
Space *with media preview open*	Start/Stop media preview
Shift click *on overlapping elements*	Open selector to chose which element to select
Alt+ mouse wheel (Shift+ mouse wheel)	Scroll Horizontally



The window comprises 3 sections: On the left is the Browser, in the middle the Pixel Matrix editing area, and on the right are the Presets panes.

The Presets pane can be hidden with the Tool Toggle button 

Pixel Matrices

You may create as many Matrices as you like and any fixture can be used as a “pixel” in any Matrix but clearly those that can colour mix are the sensible choice, RGB LEDs being by far the best due to their large colour gamut and fast response. That being said, the software will render coloured media as grey scale on intensity only fixtures.

To create Pixel Matrices automatically from the Layout (recommended):

1. Go to Layout
2. Select the fixtures you want to include in the Matrix
3. Press the New Pixel Matrix button on the Browser toolbar, the software will automatically create a Matrix with the fixtures correctly positioned and the Render Window cropped to best fit
4. Name the Pixel Matrix

To create a Pixel Matrix manually:

1. Press New on the toolbar, a default 50x50 Render Window (the pale grey area) will be created
2. Populate the Render Window by dragging on fixtures from the Browser
3. Adjust the size of the Render Window by using the Width & Height fields on the toolbar or the Crop button (see below) to best fit
4. Name the Pixel Matrix using either the Properties window or right-clicking > Rename in the Browser

Typically you will place fixtures to mimic as closely as possible their actual layout, the software will compensate for gaps and irregularities in a matrix so that media will be rendered correctly. Fixtures that have been rotated on the Layout during Setup will be placed on the Matrix using this rotation although this is only relevant to compound fixtures. Such fixtures can be rotated separately for each Matrix by dragging them around when the cursor indicates rotate mode, note that the first element of a compound fixture is displayed a darker grey for easy orientation.

To remove a fixture from a Matrix:

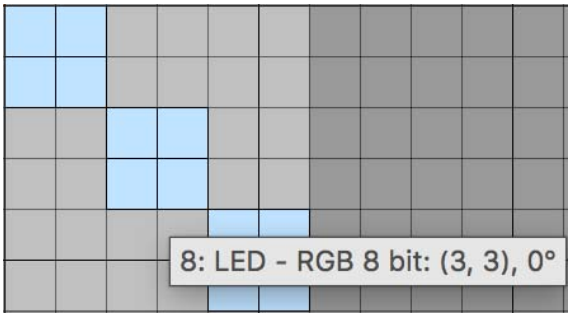
Right-click on the fixture on the Layout and select Remove Fixture.

To break a composite fixture apart into individual pixels

Right-click on the fixture on the Layout and select Break Fixture.

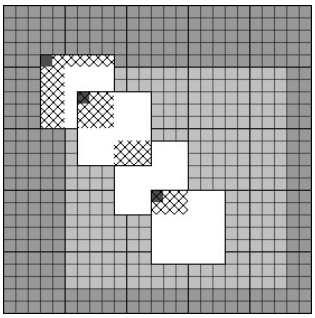
Pixel Information

To find out information about a pixel, hover over the pixel, and a tooltip will appear showing the name, position and angle for all pixels at the current position.



Cross-Hatch pattern

Pixels which are potentially incorrectly positioned to either not receive any colour information (outside Render Window) or the same colour information as another pixel (overlapping fixtures) will be identified by a cross-hatching pattern:



Crop size to contents

Use the Crop size to contents button in the Matrix Properties dialog to trim the Render Window to fit the extents of the fixture array.

Manual sizing of the Render Window is provided by typing in appropriate Width and Height size values and it is perfectly allowable to undersize the Render Window to achieve effects such as picture-in-picture or oversize it to concentrate on a particular area of the imported media.

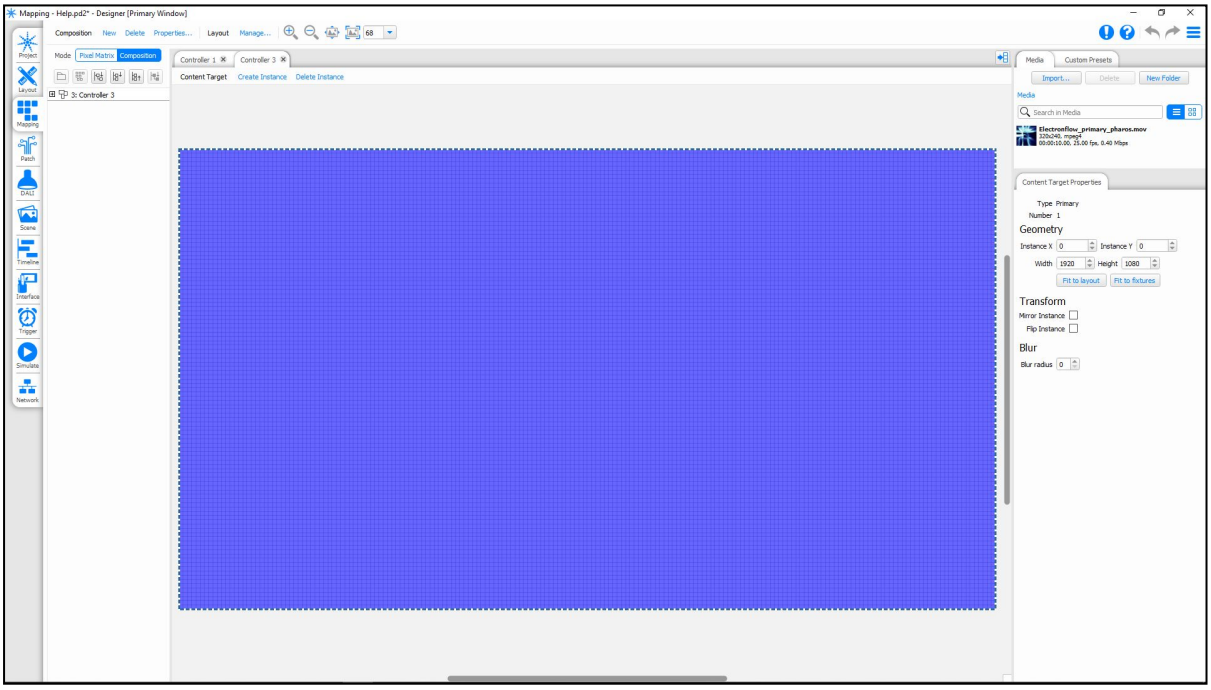
Import pixel matrix

You can use the Import button to import pixel layout from a CAD application via a CSV file, see Import Object.

Composition Editor

Keyboard Shortcuts

Ctrl+N	Create new pixel matrix
Ctrl+D	Duplicate the current pixel matrix
Ctrl+I	Show pixel matrix properties
Ctrl+C	Copy the selected media
Ctrl+V	Paste media from the clipboard into the current folder
Delete/Backspace	Delete selected media
Up/Down/Left/Right	Nudge the selected items by 1 pixel
Space+drag	Pan the view
Ctrl+0	Reset the zoom
Ctrl+F	Zoom to fit the window
Ctrl++	Zoom in
Ctrl+-	Zoom out
Ctrl+ mouse wheel	Zoom in and out
Space *with media preview open*	Start/Stop media preview
Shift click *on overlapping elements*	Open selector to chose which element to select
Alt+ mouse wheel (Shift+ mouse wheel)	Scroll Horizontally



In Composition mode, you can adjust the video composition for a VLC/VLC+. The composition allows you to setup the Content Targets for the VLC/VLC+.

This Content Target is used to specify where video is output on the layout. Only fixtures within the Content Target will get data from media or effects being played back on the VLC/VLC+.

If you are using a VLC+ in your project, you will have access to 8 Content Targets within each composition (Primary, Secondary, Target 3-8).

You will also be able to add Adjustment Targets to the controller which are configured to adjust the levels of Red, Green, Blue and Intensity that are output to the area within the mask.

Compositions

A composition is a set of Content Targets that can be stored together for a controller. When adding content to the VLC/VLC+ you select the composition that the preset should use.

There are two types of object that exist on a Composition:

- Content Targets
- Adjustment Targets

Within the Composition mode, the different types of Target are distinguished by their colour and border type.

Type	Name	Colour	Border
Content Target	Primary	Blue	Solid
	Secondary	Red	
	Target 3	Green	
	Target 4	Yellow	
	Target 5	Cyan	
	Target 6	Orange	
	Target 7	Maroon	
	Target 8	Brown	
Adjustment Target	Adjustment Target	Grey	Dashed

NOTE: The VLC/VLC+ must always have at least 1 composition, you will be unable to delete the final composition.

Content Targets

By Default, your VLC or VLC+ will have a single composition consisting of a Content Target, which is the size of the VLC/VLC+ Layout.

If you are using a VLC+, you can add a Content Target to a composition, choose the Add Content Target option, and draw the required rectangle on the layout.

There are 3 types of Content Target:

- Primary
- Secondary
- Targets 3-8

These Targets can be programmed with most presets from the Built-In preset library.

Targets 3-8 are an advanced Project Feature that must be enabled in Project Features.

The Content Target defaults to the size of the VLC/VLC+ layout, but can be resized using the Content Target properties:

Instance X and Instance Y

The position of the Target on the composition.

This property is applied per instance (where appropriate).

Width and Height

The size of the Content Target.

Fit to layout and Fit to fixtures

Automatically set the size to fit the Layout size or the minimum size to bound the fixtures on the layout.

Angle

The angle of this Content Target on the Layout.

Invert

Check this to invert the rotation.

This property is applied per instance.

Rotation X and Y

The centre of rotation of the Target.

Snap to centre

Sets the Rotation X and Y properties to be the centre of the Target

Multiplier

Sets the Target to increase its Width and Height by either 1x, 2x or 4x. This is for larger canvases with widespread fixtures that do not fit into the maximum resolution of 1920 x 1080.

Mirror Instance

Mirror all content to this target instance horizontally.

This property is applied per instance.

Flip Instance

Flip all content to this target instance vertically.

This property is applied per instance.

Wrapping

The content target can be set to wrap horizontally or vertically. When the X or Y position of the target goes beyond the edge of the layout, it will wrap round to the opposite side of the layout.

Blur Radius

The Blur of the Content Target allows you to soften the content displayed on the target. The larger the Blur radius, the softer the image

Mask

Setting a Mask will allow you to adjust the shape of the Target:

None: The Target will be a rectangle of the specified size

Shape: This allows for roundness of the corners of the Target, with the option to feather the edges.

Image: This allows an image to be used for the mask. A monochrome image works best here. Black portions will be shown by the target, while White portions will not be used.

Creating Content Target Instances

Multiple instances of a Content Target can be created to output the same data to multiple areas in the composition, by selecting a Content Target and choosing Create Instance at the top of the Layout. The new instance can have different positions and mirror/flip settings, but will always be the same size as the original.

Adjustment Targets

Adjustment Targets apply to the whole controller and can be used to reduce (or increase) RGB and intensity of presets that are played back under the mask.

The settings for Adjustment Targets are much the same as for Content Targets, but also include:

Mask Gain

The Red, Green, Blue and Intensity levels for the fixtures under the Adjustment Target can be adjusted using the gain settings. This works as a multiplier on that value. The gain can be anywhere between 0.00 and 2.00. The multiplier acts on the output levels for the fixture, but these will be limited to the 0-255 range.

You can show and hide Adjustment Targets in the Mapping view, using the  icon in the left hand browser.

Media Presets

Media presets are media clips (video or JPG) that can only be played on Pixel Matrices and Content Targets.

Media is imported by clicking Create New in the Media Presets pane. A dialog will open as shown above for you to browse to wherever your media files are located, enable Thumbnails view to preview them. Designer supports most common media files, and it matters not at all the resolution of the media clips as the software will up/down-scale as required when the Media Preset is placed on a Matrix during programming. In short, no media preparation should be required. The first frame is used as their thumbnail in the directory, they can be named using the name textbox at the bottom of the directory pane and they can also be deleted.

Imported media is automatically resized to fit a Pixel Matrices' Render Window, the pale grey area of the Matrix. Only fixtures/pixels within this Render Window will receive the media which is why Render Windows should typically be cropped (see above) to force the media to fit onto just the fixtures.

Media Limits

The maximum media limits are below:

Framerate	33fps - (dependent on Playback Refresh Rate) - the DMX protocol refresh rate is 33Hz by default, so any media with a greater framerate would be capped at 33fps, while lower framerates will display at source framerate. Note; Controllers can also run at 44Hz or 60Hz, so can handle media at higher refresh rates in those modes, see "Playback Refresh Rate" on page 284 for more info.
Bitrate	5Mbps ought to be sufficient for 1080p30, reducing to 1Mbps for 360p30
Resolution	1080p - Best results will be achieved by matching the media resolution to the output resolution (Pixel Matrix or Content Target)

Media File Formats

The following file formats can be imported as media clips:

- .asf
- .avi
- .divx
- .dv
- .f4v
- .flv
- .m2v
- .mov
- .mp2
- .mp4
- .mpe
- .mpeg2
- .mpeg4
- .mpg
- .wmv
- .jpg
- .jpeg

Media Preview

Double clicking on the thumbnail of a media preset will open the Media Preview window. This allows you to watch the media clip that has been imported.

Replacing Media

If you move the media relative to the project file, you will need to replace the content. Right click on the Media clip and select Replace to reset the file path.

This can also be used to change the media file associated with a particular clip within the project.

Designer will automatically search the selected location for any other missing media.

Custom Presets

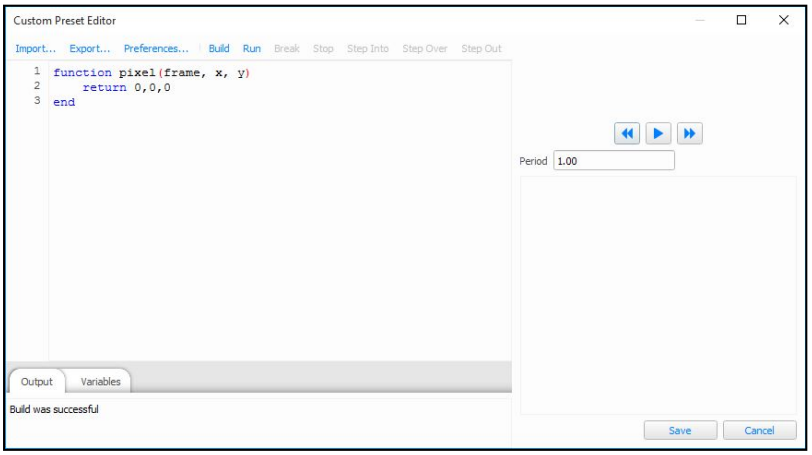
NOTE: Custom presets are an optional feature, and must be turned on from Project Features.

NOTE: The VLC and VLC+ do not support Custom Presets and thus these will not be available for projects using these controllers.

Select the Custom tab. Custom presets can only be played on Pixel Matrices.

Custom Presets use a Lua script to define an effect that can be played back on a Matrix. You can use this to create effects that are not available as standard in Designer.

To create a new Custom Preset, press New in the Custom Presets tab on the right. This will open a script editor dialog:



The script editor initially shows the framework of a custom preset. You can either enter the source yourself, or you can load the source from a file using Import. Designer provides sample scripts which are located in Program Files at \Pharos\Designer 2\resources\scripts\custom_presets.

After editing or importing a script, you compile it by pressing F7. If there are any errors in the script, they will be reported in the Output tab. If there are no errors reported, it is advisable to run the script to preview the preset. To run the script, use Run (F5). If the script takes a long time to run, or appears to have got stuck in a loop, use Stop (Ctrl+F5) to stop execution.

IMPORTANT: When working in the script editor, if your script does have an infinite loop, you are able to stop it executing by pressing Stop (Ctrl+F5). However, outside of the debugger, there is no such way to stop the execution of a badly-behaved script. This will result in Designer locking up and will have to be shut down manually. You must ensure that your scripts do not have such errors in them.

A preview of the preset will be generated and shown on the right of the script editor. You can start and stop the preview and step forwards and backwards with transport controls below the preview. Altering the period below the preview and pressing Run (F5) again will generate the preview again with the specified period.

If the preset defines any properties, after the script is successfully compiled, suitable editors for these properties will be displayed below the preview. You can change the values of these properties and rerun the script using Run (F5) to observe the effect of those properties in the preview.

Once you are happy with the preview, close the script editor. The preset can now be placed on Matrices on a timeline.

You can edit the source of the custom preset script again by selecting it in the Custom tab on the Mapping pane and pressing Edit. If you edit a preset that is already used on a timeline, any changes you make will be applied to everywhere where the preset is used.

Refer to the script editor documentation for more information on editing scripts, the custom presets programmers guide for help on creating a custom preset from scratch and custom preset examples for examples of custom scripts.

Patch

Keyboard Shortcuts

Ctrl+N	Show Add Universe popover
Ctrl+A	Select all patch records
Delete/Backspace	Delete selected patch
0-9	Type a universe number and the view will scroll to it after a short delay
Page Up/Down	Scroll to previous/next universe
Ctrl+Tab	Switch to the next protocol
Ctrl+Shift+Tab	Switch to the previous protocol

Once you have created your Layouts and added your fixtures you need to patch them, that is to say connect them to real fixtures via the appropriate Controller (LPC 1, 2, 4 or X, VLC or TPC), interface (port), protocol and address. Patching is optional for programming and simulation but fixtures must be patched eventually for the LPCs to control them, including using Output Live in the Simulator.

Before we cover patching in detail let's look at some of the terms used:

Patch terminology

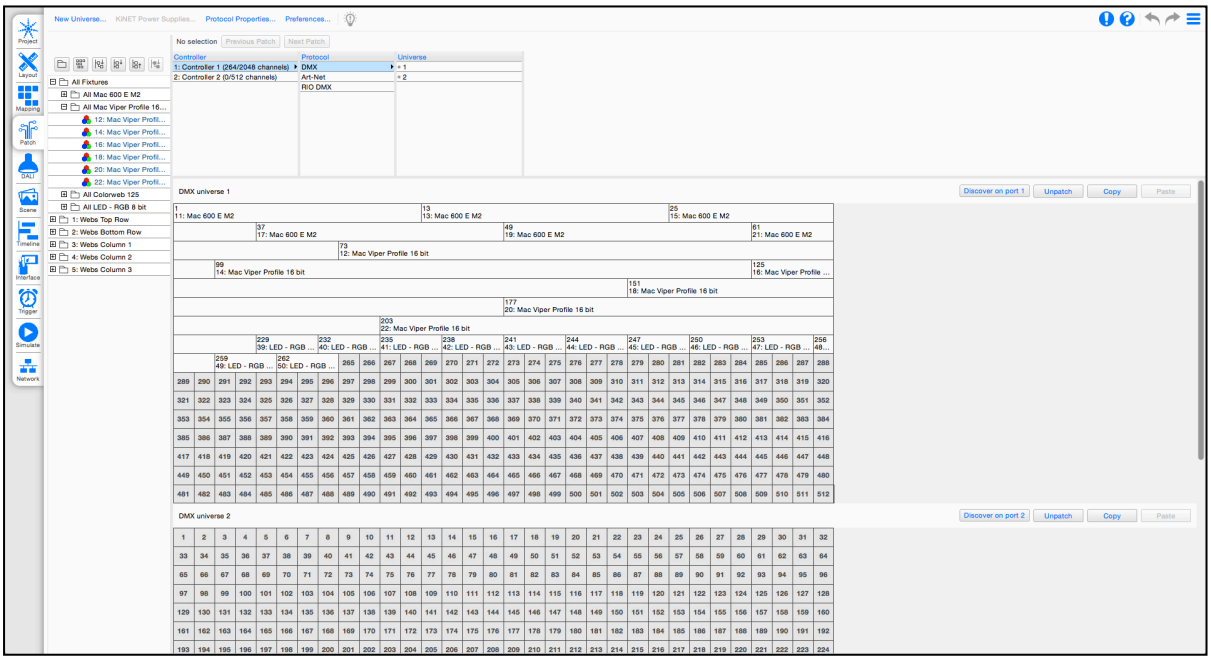
Term:	Description:	For more information:
DMX	A digital serial control protocol for entertainment lighting. Officially called DMX512-A, it was developed by the USITT and has become the standard protocol for entertainment lighting control using the RS485 physical layer.	DMX512
RDM	Remote Device Management, an extension of the USITT DMX512 protocol that supports bi-directional communication with dimmers & fixtures.	RDM
eDMX	A shorthand term for DMX-over-Ethernet protocols, see KiNET, Art-Net, Pathport and sACN below.	
KiNET	A proprietary Ethernet control protocol developed by Color Kinetics (now Philips Color Kinetics) used to control only their Ethernet PSUs. Current supported versions are KiNET v1, KiNET v2 and KiNET v3.	
Art-Net	A DMX-over-Ethernet protocol developed by Artistic Licence and widely used in the entertainment industry to distribute multiple universes of DMX data. Current supported version is v4 (transmitting only. v3 for other functionality).	
Pathport	A DMX-over-Ethernet protocol developed by Pathway Communications and widely used in the entertainment industry to distribute multiple universes of DMX data.	
sACN	Streaming ACN (Advanced Control Network), a DMX-over-Ethernet protocol developed by ESTA to distribute multiple universes of DMX data.	ESTA
RIO	The Pharos Remote Input Output 80/44/08 can output up to 96 channels of DMX per unit. See here for more information on patching to a RIO.	
DVI	Digital Video Interface, a standard for the delivery of digital video data to computer monitors. Used by certain LED manufacturers (for example Barco and Martin Professional) to drive their LED controller products.	

Term:	Description:	For more information:
DALI	Digital Addressable Lighting Interface, a digital serial control protocol for architectural lighting. Developed by Philips Lighting it has become a standard: IEC 60929. DALI fixtures are not patched using this window, see DALI.	DALI
Universe	A common term given to a single DMX data link or port. A DMX universe carries 512 channels of control data each with 8 bit resolution. A single dimmer will use one channel while more complex fixtures will use multiple channels as required.	
Port	The KiNET equivalent of a universe.	
DMX Address	The term used to determine which of the 512 control channels of a DMX universe a fixture should look at to take its own control data. This "start address" must be set on the fixture or dimmer rack itself as well as patching the control system.	

Patch window

This window comprises two main sections, to the left is the Browser, with the rest of the window being a graphical representation of a protocol's ports or universes. The number of address columns displayed per row can be changed using Preferences.

If you are using an LPC X, VLC or choosing to output eDMX from an LPC or TPC then you must use the Protocol Properties dialog to configure these protocols, see Controller Protocols.



Patch Columns

Use the Controller column to select the Controller for patching. Add Universes to the project with the Add Universe dialog. Set properties for eDMX protocols with the Protocol Properties dialog and set your Patching Preferences with the Preferences dialog.

Use the other columns to select the required protocol and universe/s to determine which universe/s is/are displayed in the main working area.

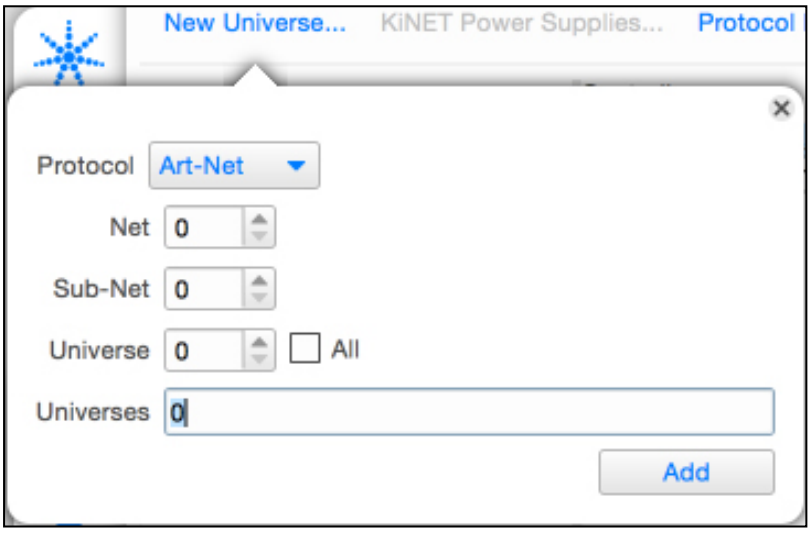
NOTE: The patch columns width can be adjusted to display long value on your screen, by clicking and dragging on the dividers

Universes

By default only the DMX ports of an LPC are configured, TPCs LPC Xs and VLCs have Art-Net Universe 0 configured by default.

To add eDMX universes:

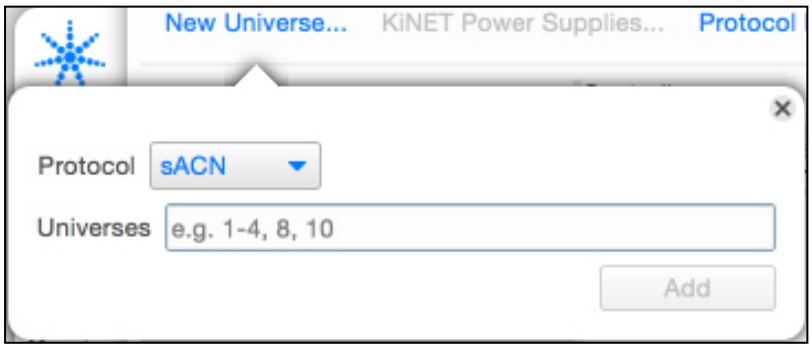
1. Select the Add Universe dialog
2. From the Protocol Dropdown, select the protocol you want to setup.
3. For Art-Net:



You can also enter the Universe using the Net/Sub-Net/Universe notation e.g. 0/0/0 = Art-Net Universe 0, selecting All universes will add all 16 universes in the specified Net/Sub-Net.

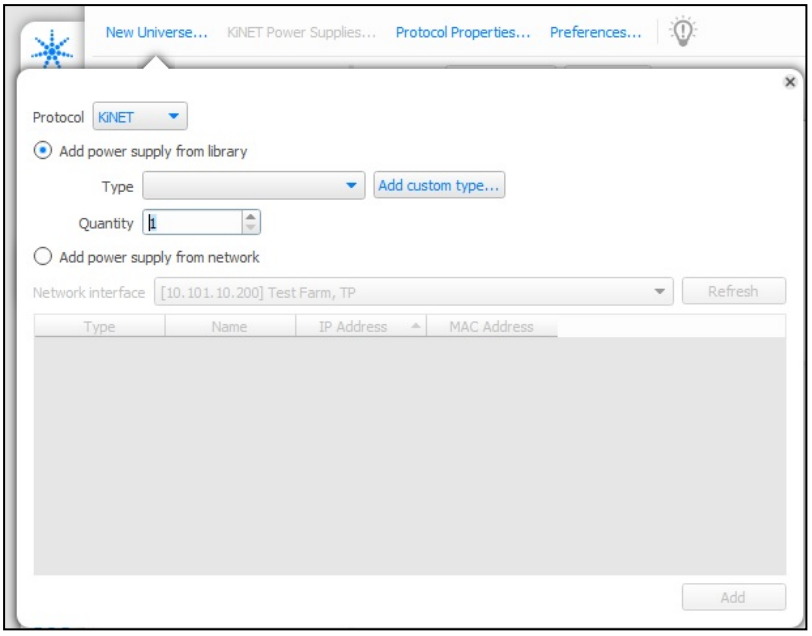
- In the Universes box type the required universes e.g. 1,
- You can separate multiple universes with commas e.g. 1, 5, 7,
- You can select a range of universes with a hyphen e.g. 1-5,
- These can be combined e.g. 1-4, 8,10

4. For sACN and Pathport:



- In the Universes box type the required universes e.g. 1,
- You can separate multiple universes with commas e.g. 1, 5, 7,
- You can select a range of universes with a hyphen e.g. 1-5,
- These can be combined e.g. 1-4, 8,10

5. For KiNET Power Supplies:



- You can add a virtual power supply from the included library of power supplies or add a power supply that exists on the network that you are connected to.
    - To add a virtual power supply (the default option), simply select the type of power supply from the Type dropdown list. Enter the quantity that you intend to use. Click add at the bottom of the dialog.
    - To add a Custom Power supply, you can select "Add Custom type..." and enter the required Name, Port count, KiNET Version and select whether it is Chromasic or not.
    - To add an existing power supply, Select the "Add power supply from the network" checkbox. Click Refresh to display all power supplies visible to Designer on your network. Select the power supply/ies you want to add to the project and click add.
- If you add a virtual power supply, you will need to set its IP Address. This can be done from the table in the KiNET tab. Double clicking on an IP Address will allow you to change it, or use the Assign IP Address option to link the virtual power supply to a power supply discovered on the network.
- When you add a KiNET power supply to a project, it is initially linked to a specific controller, but you can patch different ports to different controllers using the Show All button in the top right corner

Patching DMX & eDMX fixtures

Simply select one or more fixtures in the Browser and drag them onto the required start address of the graphical representation. Right click on a patched fixture to unpatch it or clear the entire universe/port, drag it to move it (change its address). Fixtures may be patched to multiple addresses and universes/ports. Patched fixtures are shown in blue in the Browser, unpatched black.

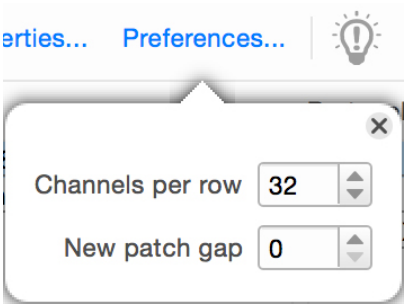
To patch a fixture:

1. Select the relevant Controller, Protocol and Universe/s.
2. Scroll down to the required Universe.
3. Select the fixture in the Browser
4. Drag and drop the fixture onto the desired start address

To patch multiple fixtures:

1. Select the relevant Controller, Protocol and Universe/s.
2. Scroll down to the required Universe.
3. Select a group of fixtures in the Browser
4. Drag and drop the group of fixtures onto the desired start address of the first fixture in the selection
5. If you select more fixture than will fit on the selected universe, or patch them such that the selection runs out of space on the universe, the fixtures will be automatically patched to the subsequent universe.

To Patch Fixtures with a Gap between them



Sometimes it is useful to patch a group of fixtures with a spacing between them, such as when you are controlling an RGBW fixture as separate RGB and White fixtures.

1. In the Preferences option on the Patch Toolbar, change the New Patch Gap setting to be the number of channels gap between the fixtures
2. Patch a group of fixtures as normal.
3. There will be the specified gap between the fixtures.

To change a fixture's address:

1. Select the relevant Controller and Protocol.
2. Scroll down to the required Universe.
3. Select the fixture(s) on the universe layout and simply drag them to a new address - to move them to a different protocol you must unpatch (see below) then repatch.

Fixtures may be patched to as many locations and universes as is required although typically a fixture will only be patched to one unique address. The Designer software will prompt you with a warning dialogue should you attempt to patch a fixture that is already patched, select Continue or Unpatch Existing as required. This prompt can be turned off if it proves aggravating.

However, you can not patch more than one fixture to the same address; a DMX channel can only be controlled by one parameter so fixtures can't overlap at all. If you drag one fixture onto another the incoming fixture will highlight in red to alert you that this address is already occupied. If you go ahead and drop it there anyway the software will prompt you whether to continue and unpatch the existing fixture(s) for you, select Unpatch to proceed or Cancel to abort. Again, this prompt can be turned off.

Some fixtures, for example the Vari*lite VL5, need to be patched twice since they have two distinct patch points, one for the intensity control (patched to the dimmer rack) and another for the fixture's automation controls.

To patch a multiple patch point fixture:

1. Locate the fixture in the Browser and expand it by pressing the plus sign to reveal its patch points
2. Select the relevant Controller and Protocol.

3. Scroll down to the required Universe.
4. Drag and drop the first patch point onto the desired start address
5. Repeat for the other patch point(s)

or

1. Patch the entire fixture as one
2. Drag the patch points apart to their desired addresses

To unpatch a fixture or multiple fixtures:

1. Select one or more fixtures using the Browser or the universe layout
2. Right-click and select Unpatch

To clear patch from a universe:

1. Select the relevant Controller and Protocol.
2. Scroll down to the required Universe.
3. Use the Unpatch button in the Universe header to remove all fixtures from the universe.

To copy a universe's patch:

1. Select the relevant Controller, Protocol and Universe/s.
2. Scroll down to the required Universe.
3. Press the Copy button in the Universe header.
4. Select the target Controller, Protocol and Universe
5. Press the Paste button in the Universe header.
6. Continue pressing paste on other universes if necessary.

To Cut/Copy Multiple Universes' Patch:

- Use the Patch columns to select the required univere/s
- Right click and Cut/Copy
- Go to the new Univere/s
- Right-click and paste the Patch

To highlight a fixture  :

1. Select one or more fixtures using the Browser or the universe layout
2. Press the Highlight button, the fixture(s) will come on to their highlight defaults (typically open white)
3. Press Highlight again to turn off or select other fixtures to highlight

NOTE: The appropriate Controller must be on the network and correctly associated to highlight fixtures. Fixtures can also be highlighted from Layout.

NOTE: If your controller/s is/are password protected, you will need to login to the controller/s from the Network Mode

RDM Discovery

RDM Devices can be discovered on the DMX port of a LPC or TPC + EXT or on an Art-Net universe.

NOTE: If security is enabled on your controller, you will need to login to perform RDM discovery.

Open the RDM Discovery window by clicking the RDM Discovery button at the top of Patch. Select the required controller, protocol and universe and click Discover to find RDM-capable devices attached to the selected output. The RDM discovery table will display all discovered RDM capable devices connected to the selected output.

It's also possible to start RDM discovery on a particular universe by clicking Discover on port… above a compatible universe in the patch.

NOTE: The Discover button is only enabled if the selected controller is associated with a physical controller and that controller has been found on the network.

From the RDM Discovery window, it's possible to carry out the following actions:

- To put a device into its RDM identify mode, toggle the Identify button.
- To readdress a device, enter a new value in the Start Address column. This will send an RDM command to readdress the device.
- To set a different Personality, enter a new value in the Personality column.
- To set the fixture's Type, double-click the Type cell and select a type from Designer's online fixture library.

To add discovered RDM devices to the project

To add all discovered RDM devices to your project, click Add All Discovered Devices from the RDM Discovery window. This will also add the discovered RDM devices to the patch at their respective start addresses. New fixtures will be created in the browser.

To add a custom selection of one or more RDM devices to your project, click and drag the desired fixture(s) from the RDM Discovery window onto a layout, into the patch or into the browser. Devices will be added in the order they were selected.

Dragging a discovered RDM device onto the patch will send an RDM command to readdress the device.

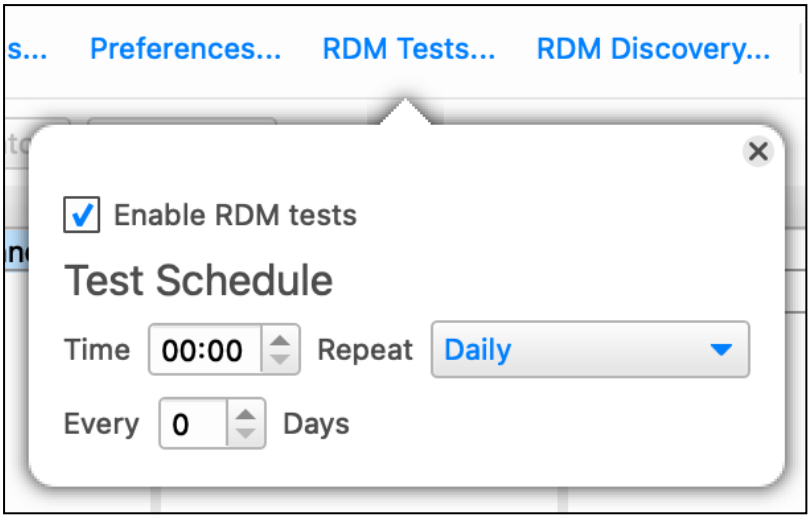
NOTE: Discovered RDM devices can only be patched to the universe they were discovered on.

NOTE: A discovered RDM device cannot be dragged onto the patch if its Type has not been set or if its start address and footprint clashes with another device.

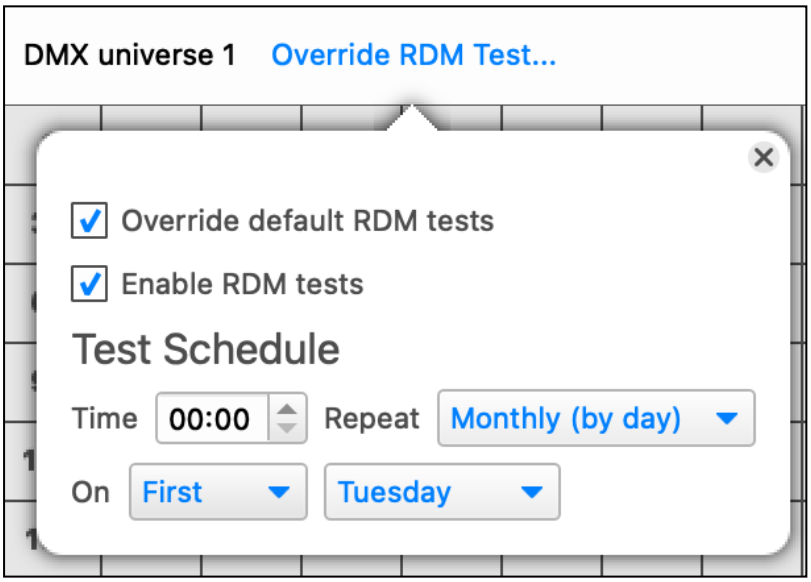
A discovered RDM device can be associated with an existing fixture in the project by dragging the discovered device onto the fixture in the project.

Scheduled RDM Tests

RDM tests may be scheduled for all compatible universes in a project by clicking the 'RDM Tests' button in the toolbar of the Patch window.



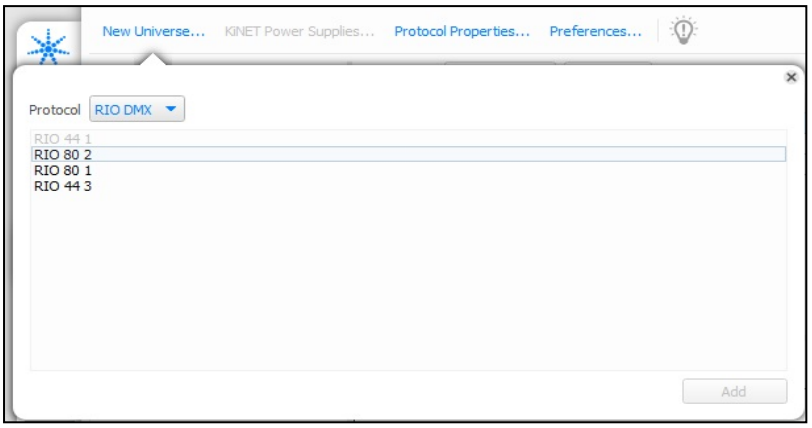
You can override project settings on a per universe basis by clicking 'Override RDM Test...' next to the corresponding universe in the patch view.



Test results can be viewed via the 'Fixtures' tab of the controller's web interface.

NOTE: Due to the nature of the RDM protocol, RDM tests will momentarily stop DMX output on a universe. We recommend scheduling your tests to occur at a time when playback is not critical.

Patching fixtures to a RIO 08/44/80

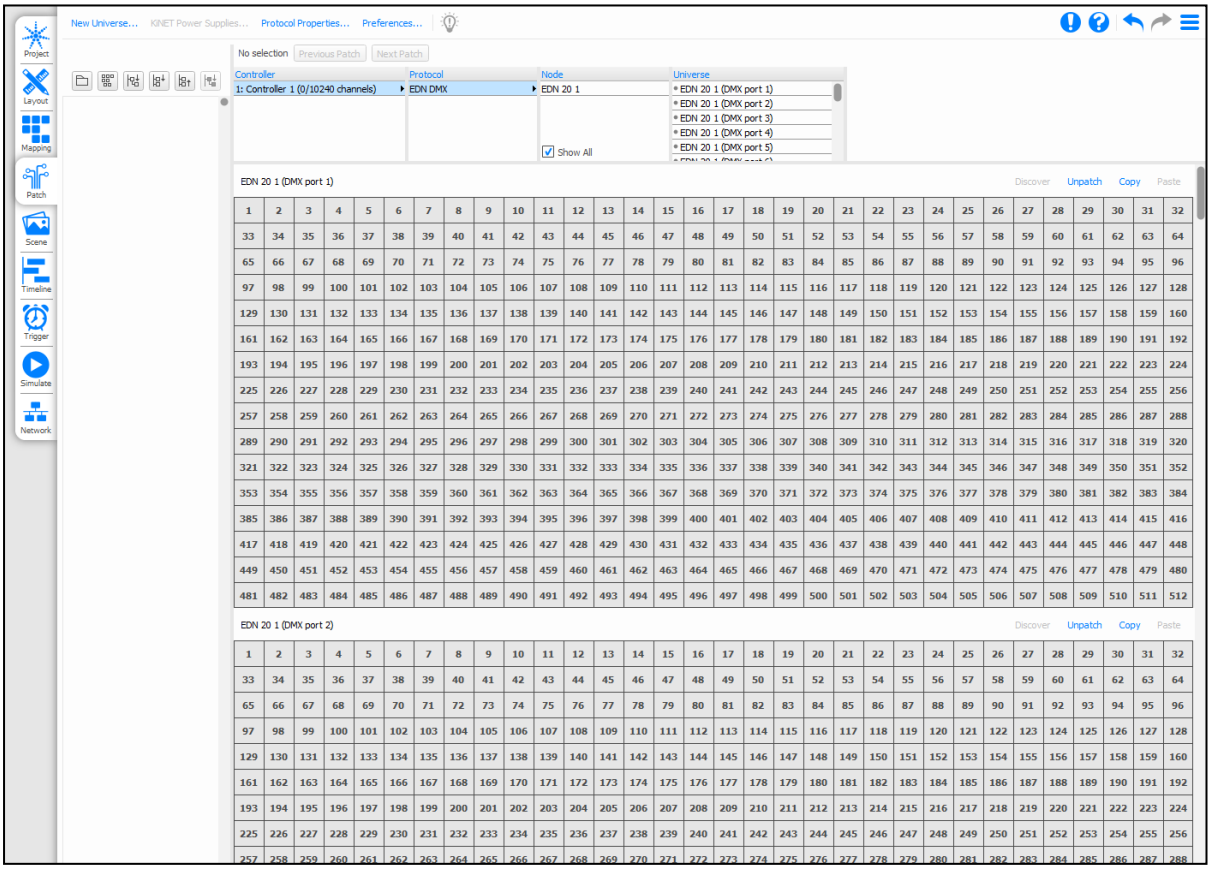


You must first add a RIO 08, RIO 44 or RIO 80 to your project. You will then be able to select RIO DMX as a Protocol in the Add Universe dialog.

Assigning a DMX universe to a RIO 08, RIO 44 or RIO 80 will automatically configure the Serial port to be a DMX out port.

Each RIO 08, RIO 44 and RIO 80 is capable of outputting 96 channels of DMX. Any channels patched to a RIO 08, RIO 44 or RIO 80 will subtract from the maximum channel output capacity of the Controller. The RIO 08, RIO 44 or RIO 80 and Controller must be on the same network.

Patching fixtures to an EDN or RIO G4



You must first add an EDN or RIO G4 to your project. You will then be able to select EDN DMX as a Protocol in the Add Universe dialog.

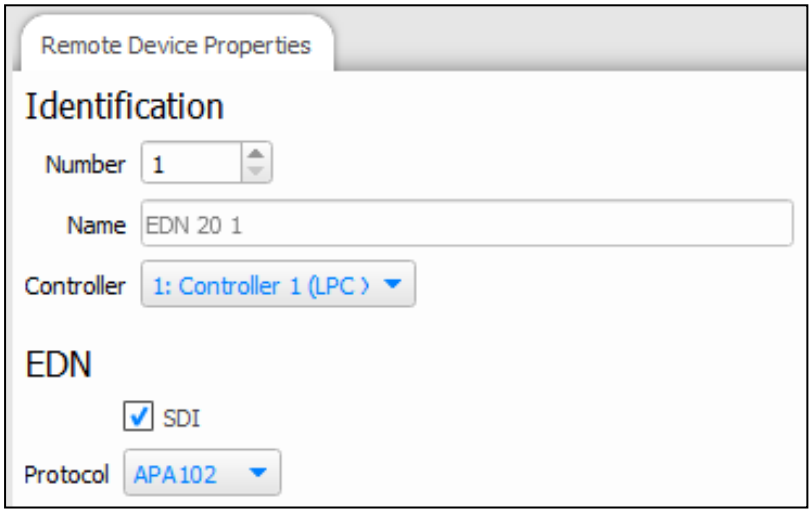
The EDN/RIO G4 will have a number of ports available to patch fixtures to. Any fixtures patched to these ports will used channels from the capacity of the selected controller.

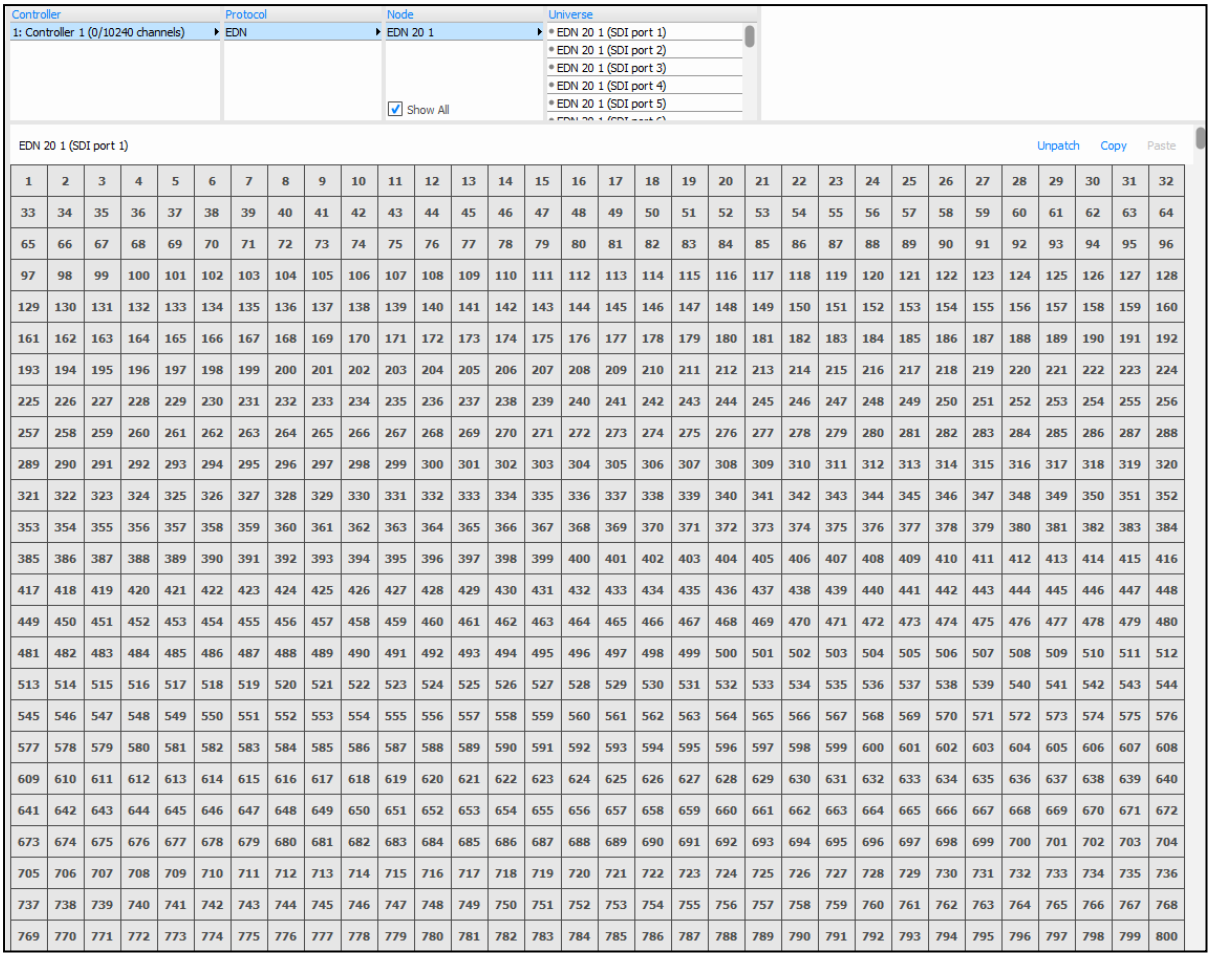
NOTE: Multiple controllers can output to different ports on the EDN or RIO G4, provided they are in the same project.

Patching fixtures to an EDN/RIO G4 + SDI

This functionally is the same as patching to regular DMX, but each port can hold up to 1536 channels.
To enable this, ensure the EDN/RIO G4 has SDI checked in the Network view, as shown below.

NOTE: Fully patching an SDI universe will count as 3 regular universes for the purposes of channel count.





Used channels

In the controller column of the patch universe view is an indicator of the number of used and available channels for each Controller.

When the Controller is an LPC X, and if you have not associated the controller with a physical device, then you are allowed to patch fixtures up to the capacity you have chosen (see Controller Association).

However, when you associate an LPC X with a physical device, you will only be able to associate to devices which have a capacity equal or greater than the patched fixtures. After associating with a device, you will be unable to exceed the capacity of that device.

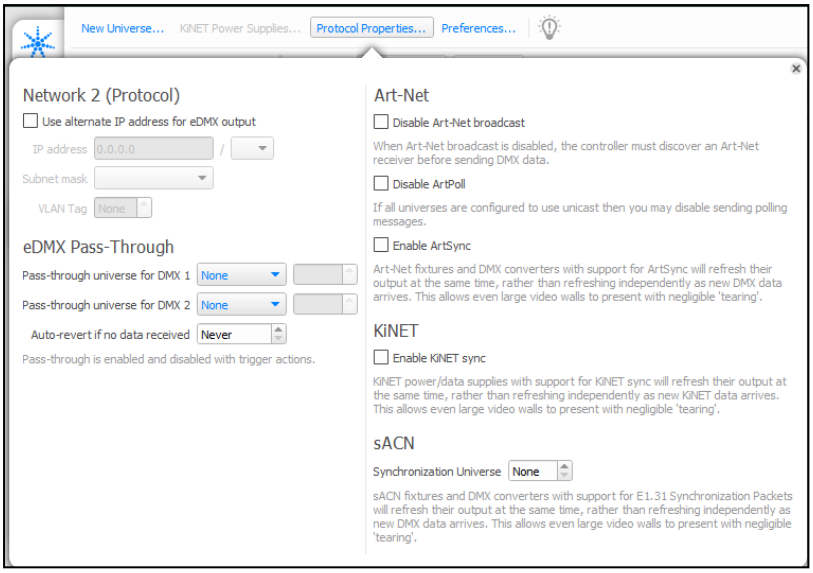
Show patch location

When a fixture is selected, in the fixture browser, or in the patch table, the fixture name will be given at the top of the view, with the option to show each patch location for the fixture.

eDMX Pass-Through

When using an LPC in a project it is possible to allow eDMX from another eDMX source to be passed through to the controller's DMX ports. With an LPC selected, the bottom of the Protocol Properties dialog will show the eDMX Pass-Through settings. Select which universe the DMX port will be transmitting. Note that with an LPC 2 you'll be able to choose a different universe for each DMX port on the controller. There is also the option to auto-revert to the project's output if eDMX isn't received for a specified amount of time.

NOTE: only Art-Net and sACN are currently supported for eDMX Pass-Through.



eDMX Pass Through Merge

If sACN is being used for eDMX Pass Through and multiple sources are received, the controller will use the Source with the Highest priority.

If multiple streams are received with the same priority:

- If there are exactly 2 streams, the controller will do a HTP merge (per channel). Meaning the highest level for each channel from either source will be used.
- If more than 2 sources are received, then all streams are dropped.

Art-Net output customisation

Casting

There are three options for the casting of Art-Net Data, which refer to how the controller outputs the data:

- Automatic
- Broadcast
- Unicast

Automatic will utilise ArtPoll messages to determine which IP Addresses to Unicast the Art-Net data to.

Broadcast will output the Art-Net data so that any device may receive it.

Unicast allows you to specify the destination for the Art-Net universe.

This can be setup more quickly using the Import function. More information can be found here.

In Automatic mode, a controller with less than 30 universes of Art-Net patched will broadcast all data until a device requests unicast for a specific universe. Controllers with more than 30 Art-Net universes patched will only unicast data to devices requesting universe data and will not automatically broadcast.

There is the option to 'Disable Art-Net Broadcast' for a controller. The controller will still unicast data to devices that request it. There is also the option to 'Always Broadcast' on a per universe basis. This will force the controller to always broadcast that universe's data. 'Always Broadcast' will override the 'Disable Broadcast' option.

ArtPoll

It is possible to disable ArtPoll messages from the Protocol Properties. This means the controller will not output these messages for Art-Net discovery and management of casting. This requires data to be broadcast or for unicast addresses to be configured.

Syncing

Within the Protocol Properties, is the option to enable Art-Net Sync, which can be used to ensure multiple Art-Net receivers stay in sync (if they are capable of receiving this).

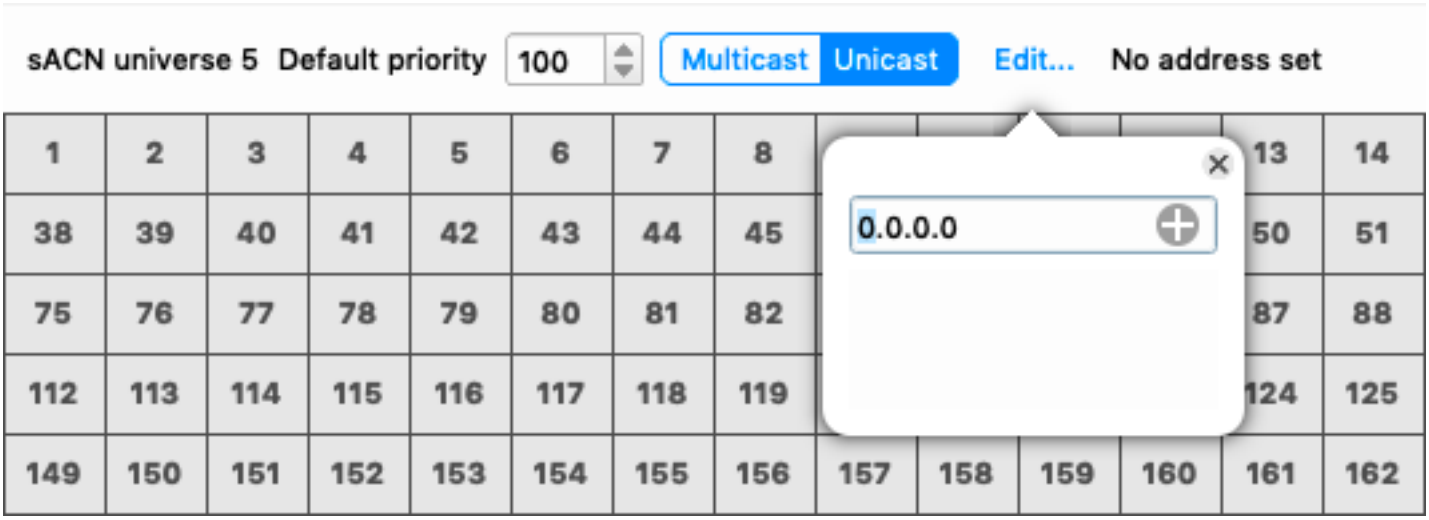
sACN Output Customisation

When configuring sACN Universes, it is possible to set:

Casting

Each universe can be configured to Multicast (the default) or Unicast (send to a specific IP Address).

When setting to Unicast, multiple IP addresses can be set:



This can be setup more quickly using the Import function. More information can be found here.

NOTE: The controller will always output "Discovery" messages to the Multicast Discovery address every 10 seconds, so sACN tools can see the controller.

Priority

Each Universe can have its priority set within the Universe header. This is used by any receivers to determine which data to output if it receives multiple data streams.

Each channel can also have a priority set, allowing further customisation of how receivers will handle data from multiple sources.

Syncing

Within the Protocol Properties, is the option to enable sACN Sync and indicate a Sync Universe, which does not need to be a universe controlled within the project. This can be used to ensure multiple sACN receivers

stay in sync across a whole network (if they are capable of receiving this), using a single source as the synchronisation controller.

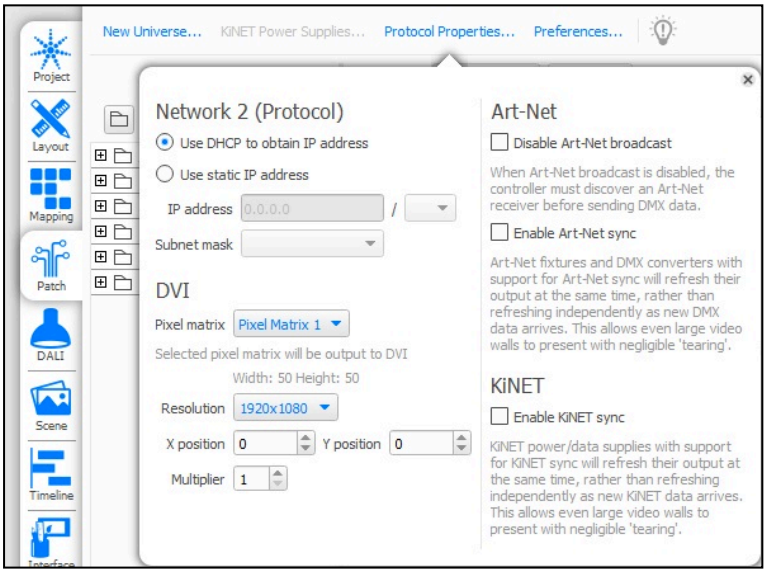
KiNET Output Customisation

Within the Protocol Properties, is the option to enable KiNET Sync, which can be used to ensure multiple KiNET receivers stay in sync (if they are capable of receiving this).

Patching DVI fixtures (LPC X)

To output data using the DVI/DisplayPort port you must first create a pixel matrix that matches the LED controller's pixel map. Once this has been done, use the Pixel Matrix pull-down on the Protocol Properties dialog to select which matrix will be output via the DVI/DisplayPort port and select an X & Y offset as required. Any programming for the fixtures in the pixel matrix will now output on the DVI/DisplayPort port, not just programming applied directly to the pixel matrix.

The LPC X's DVI/DisplayPort port is set to a fixed 1024x768 @ 60Hz resolution which is compatible with most LED controllers. The LED controller (or monitor) MUST be connected when the LPC X boots or resets for the port to become active.



DALI

Keyboard Shortcuts

Ctrl+N	Create a new DALI Interface
Ctrl+I	Show DALI interface properties
Escape *in Scene Mode*	Toggle last fixture selection
Ctrl+0 *in Scene Mode*	Reset the zoom
Ctrl+F	Zoom to fit the window
Ctrl++ *in Scene Mode*	Zoom in
Ctrl+- *in Scene Mode*	Zoom out
Ctrl+ mouse wheel *in Scene Mode*	Zoom in and out
Middle-click + drag *in Scene Mode*	Zoom into the drawn rectangle
Alt+ mouse wheel (Shift+ mouse wheel) *in Scene Mode*	Scroll Horizontally

By default this Mode is not available, but will become active if a DALI ballast or DALI remote device is added to the project.

Overview

The Digital Addressable Lighting Interface (DALI) is a digital serial control protocol for architectural lighting. Developed by Philips Lighting it has become a standard: IEC 60929. DALI differs from DMX in a number of important ways:

- Only 64 ballasts per DALI bus (interface)
- Only 16 DALI groups per interface
- Only 16 DALI scenes
- Ballast configuration (including address), groups and scenes is uploaded to and stored in the ballasts themselves
- Topology-free DALI bus operates at very low data rates
- Command-based protocol, ballasts perform fades and maintain levels
- Only certain discrete fade values are permitted

As a result the DALI protocol is not suitable for rendering effects and media, programming is restricted purely to recalling lighting levels via the Set Level and DALI Scene presets, see DALI presets.

DALI Ballasts

DALI Ballasts are added to the project in the same way as DMX based fixtures, from the fixture library within Layout.

Ballasts are available that support DALI Type 0 (intensity control) and the following colour control modes from DALI Type 8:

- XY Colour Control
- Tc Colour Temperature Control
- RGBWAF Colour Control

These DALI Type 8 ballasts can be controlled within Pharos Scenes (as opposed to DALI Scenes).

The Fixture library also includes composite DALI fixture personalities which use multiple addresses to control different parameters e.g. Intensity and Colour Temperature. Each parameter can be individually patched to a DALI address.

DALI interfaces

Each RIO D or TPC with EXT added to the system supports one DALI bus (up to 64 ballasts) and is assigned to a DALI interface within Designer. Each RIO D4 added to the system supports four DALI buses, with one DALI interface assigned per port. Due to the nature of the DALI protocol, these DALI interfaces are insular affairs with each having its own unique set of ballasts and groups. Each interface must be configured and uploaded to individually.

When you add a RIO D, RIO D4 or TPC with EXT you select which DALI interface to assign. For RIO D4, one interface can be assigned to each of the device's four ports. Use the Add Interface button on the DALI toolbar to add another. Use Remove Interface to remove an unwanted one.

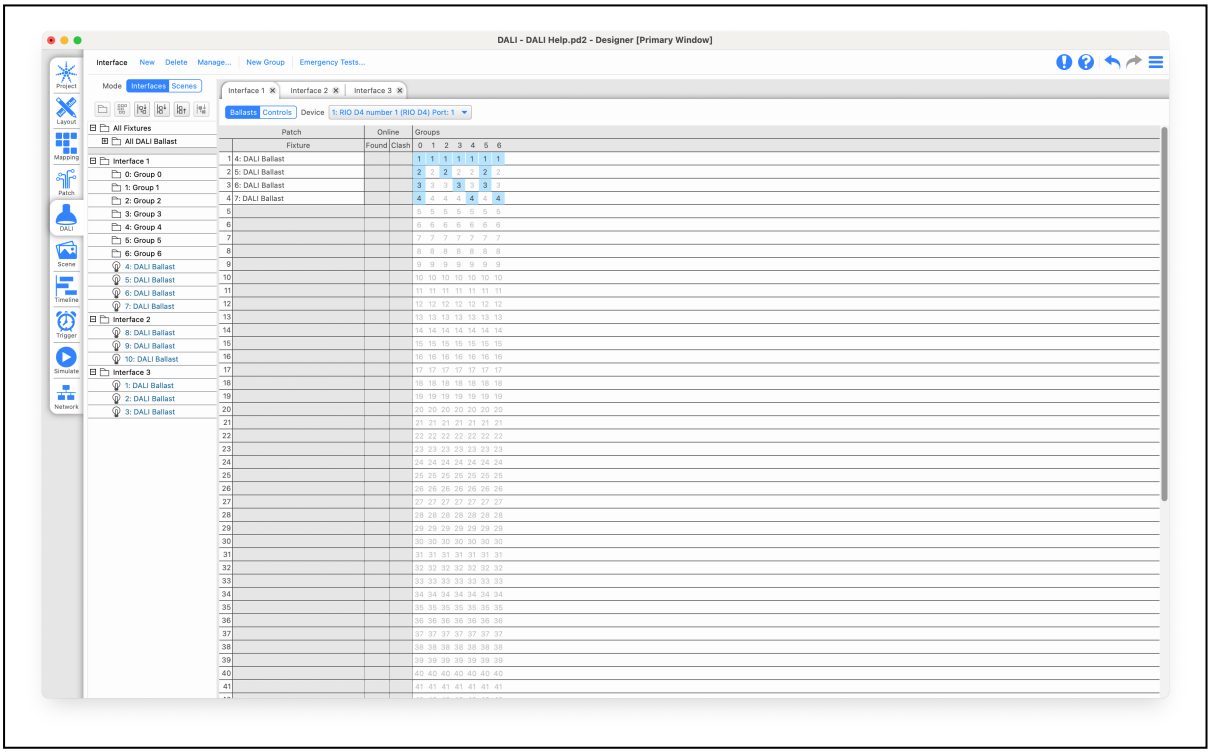
The maximum number of DALI interfaces assigned to DALI remote devices for each controller is as follows:

- LPC 1 / TPC - 16
- LPC 2 - 32
- LPC 4 - 64
- LPC X / VLC / VLC+ - 100

Up to 200 DALI interfaces can be added to a project.

DALI addressing & grouping

Each DALI interface is configured separately, use the pull-down on the DALI toolbar to select an interface for configuration:



The Addressing table allows you to link DALI ballasts within the project with an address on the DALI bus. You are also able to set the address of the physical ballasts and assign them to groups. The device commands are available to manage the ballasts on each DALI interface.

Device commands

IMPORTANT: Designer must be connected to the Controllers with RIO D, RIO D4 or EXT, the project file must have been uploaded at least once and the DALI ballasts must be active to perform these operations.

A DALI ballast internally stores its address, this is a number between 1 and 64. It is also possible that the ballast has never been addressed so it does not have an address. These commands are used to discover and address DALI ballasts:

Find addressed ballasts

This queries the ballasts attached to the device and reports all addressed ballasts found, an icon is added to each address cell when the corresponding ballast has been found.

Address ballasts

This finds all ballasts without an address and randomly assigns them a free address. It will not change the address of any already addressed ballasts.

Readdress all ballasts

This will clear the addresses of all ballasts and then assign every ballast a random address.

Resolve clash

It is possible that two ballasts can have the same short address. If that happens the ballasts clashing are shown with a red icon. The resolve clash button will move the clashing ballasts to a random address that is unused by any other ballast.

Identify emergency ballasts

Send all emergency ballasts a command to indicate their address on the multicoloured LEDs on the fixture. Whilst this is enabled the command will be sent every 10 seconds.

To manually readdress a DALI ballast:

1. Select the ballast icon in the current address cell
2. Drag it to the target (preferably empty) address cell

The ballast is readdressed to the target cell.

To highlight a ballast

Select an address cell (containing a ballast icon) and press Highlight to bring this ballast to full level, select another cell to highlight instead or press Highlight again to turn off.

Patching DALI fixtures

When you add DALI fixtures to your layout these "abstract" ballasts are assigned to the Unpatched DALI group in the Browser and must be mapped to real DALI interfaces and ballasts using this window.

Once you have discovered and addressed all the real DALI ballasts for each interface (if more than one) you can then patch your DALI fixtures to them simply by dragging them from the Unpatched DALI group onto the required interface and address cell. As each DALI interface is assigned DALI fixtures, the Browser refreshes to reflect these changes. The Unpatched DALI group will become empty once all the DALI fixtures in the project have been patched.

It is of course possible to patch your DALI fixtures blind in advance of being connected to the real ballasts. The patch is stored with the project data and not on the DALI ballasts.

DALI groups

Unlike other groups in Designer, DALI groups are a property of the real DALI interface not an abstract collection of fixtures, and there can only be 16 DALI groups per interface. The right-hand side of the addressing table allows you to add and define groups for each interface:

To add a DALI group:

1. Press the New Group button
2. You can give the group a name by typing in the pre-selected name field
3. Select which ballasts are to be a member by clicking the fixture's address in the corresponding group column

The DALI group is added to the Browser and group configuration data ready to be uploaded into the ballasts.

To delete a DALI group:

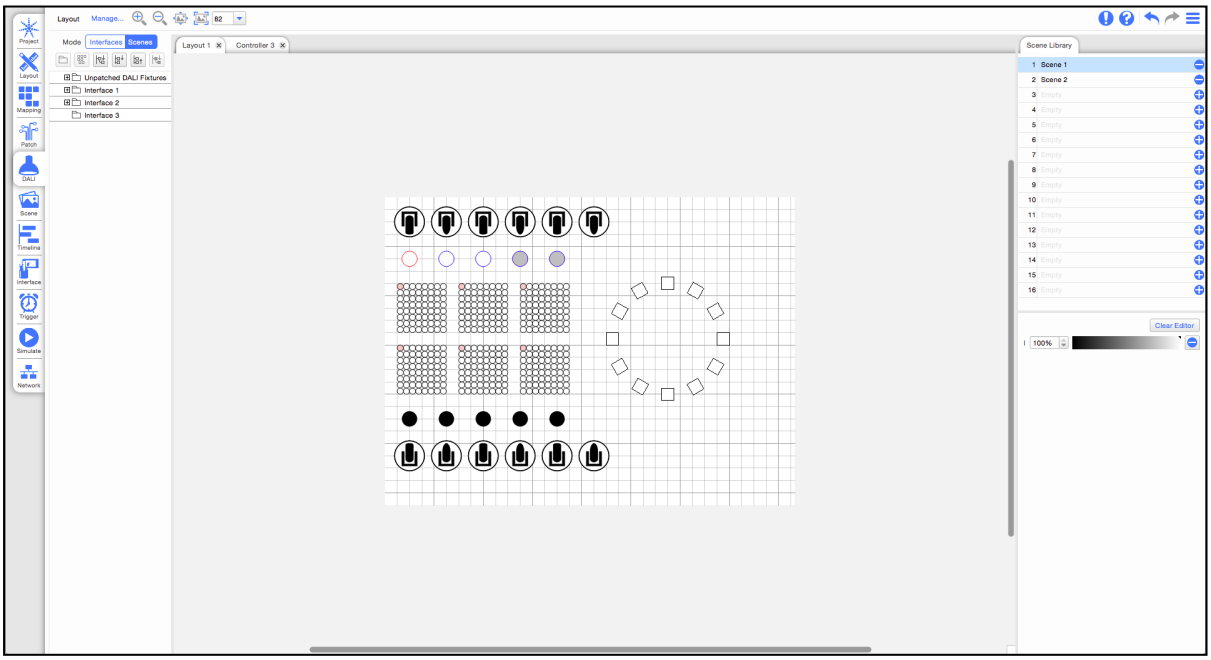
1. Right-click on the group in the Browser
2. Select Delete

The DALI group is removed and the group configuration data updated ready to be uploaded into the ballasts.

If groups have already been programmed onto the DALI fixtures you can press the Discover Groups button to automatically populate group information from the fixtures on the interface.

DALI scenes

In Designer's implementation, DALI scenes are common across DALI interfaces, change the mode from Interfaces to Scenes (on the top left):



The Browser and Layout are displayed so that you can select the DALI fixtures and groups. On the right are the DALI scenes library and the Intensity controls.

There can be up to 16 DALI scenes which can contain programming for some or all of your DALI fixtures, even if spanning different DALI interfaces. In general you should include programming for all your DALI fixtures in each scene since you can determine when creating timelines which fixtures or groups should run the scene by dropping the DALI scene on the appropriate timeline row, the choice is yours though.

NOTE: DALI Type 8 ballasts can only be controlled in Scenes (not DALI Scenes)

To create a DALI scene:

1. Press the  button next to a new scene to create the scene.
2. Name the new scene
3. Select the DALI fixtures or groups
4. Set the required level (0-254), the fixtures on the layout will simulate these levels

If DALI RGB Ballasts are being used, you will also be able to set the colour of the ballast.

The DALI scene is added to the folder and scene configuration data is ready to be uploaded into the ballasts.

To delete a DALI scene:

1. Press the  button next to the scene to be deleted

The DALI scene is removed and scene configuration data updated ready to be uploaded into the ballasts.

To edit a DALI scene:

1. Select the scene in the folder
2. Select the DALI fixtures or groups
3. Adjust the levels

The DALI scene is edited and scene configuration data updated ready to be uploaded into the ballasts.

To remove a DALI fixture from a scene:

1. Select the scene in the folder
2. Select the fixture to remove
3. Press the  Knockout button to the right of the Intensity controls

The DALI scene is edited and scene configuration data updated ready to be uploaded into the ballasts.

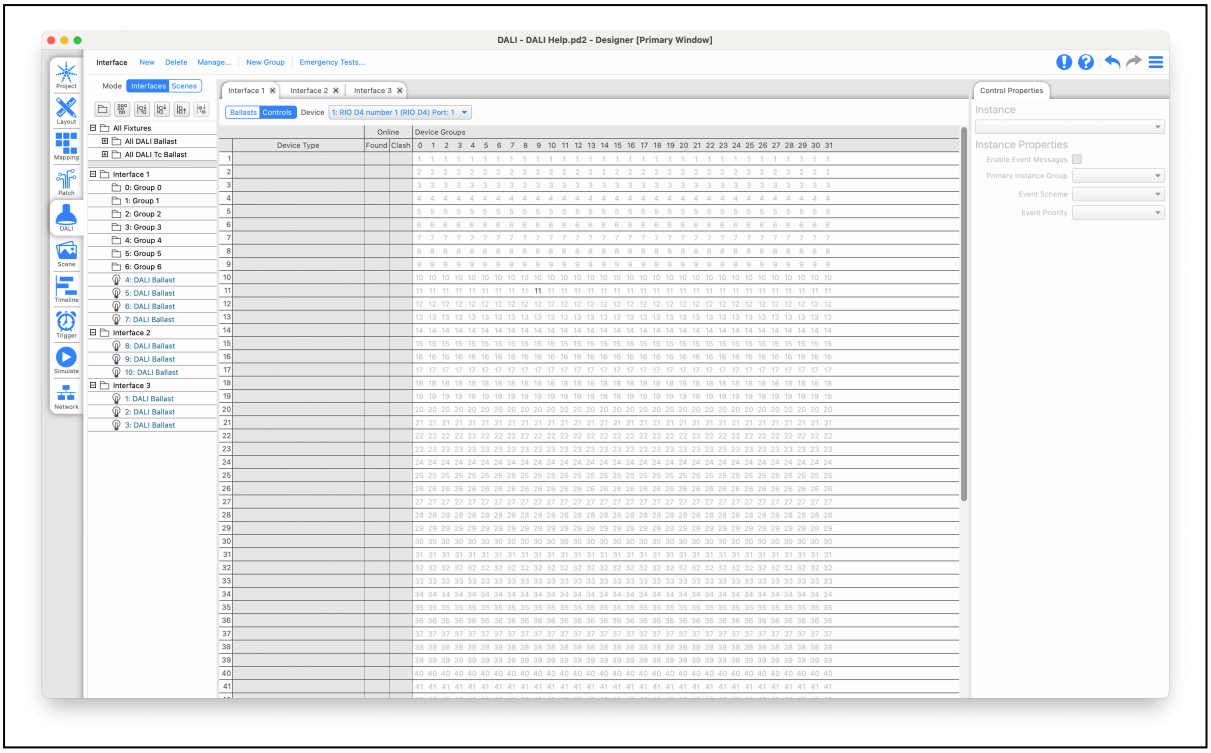
DALI Controls

Select 'Controls' using the switch at the top of the DALI Interface tab to view and configure your DALI controls. From here, you can view a table of control addresses 0-63 showing the device type, whether the device is online (Found indicates the control has been found, Clash indicates clashing devices at this address), and group assignment.

Buttons are provided for the following functions.

- Find Addressed Controls
- Address Controls (assigns addresses to un-addressed devices)
- Re-Address All Controls (randomises all addresses and re-addresses all devices on the interface)

NOTE: Buttons for these functions are only available if there is a RIO D, RIO D4 or TPC+EXT online and associated with the interface.



Instance Properties

When a row is selected, properties for the instance(s) within the selected device can be configured. Use the drop down list to select different discovered instances.

The properties available are:

Push Buttons

- Enable/disable event messages
- Select primary instance group
- Event scheme
- Event priority
- Event filters:
    - Button released
    - Button pressed
    - Short press
    - Double press
    - Long press start
    - Long press repeat
    - Long press stop
    - Button stuck/free
- Timers
    - Configurable delay times for
        - Short
        - Double
        - Repeat
        - Stuck

Occupancy Sensors

- Enable/disable Event messages
- Select primary instance group
- Event priority

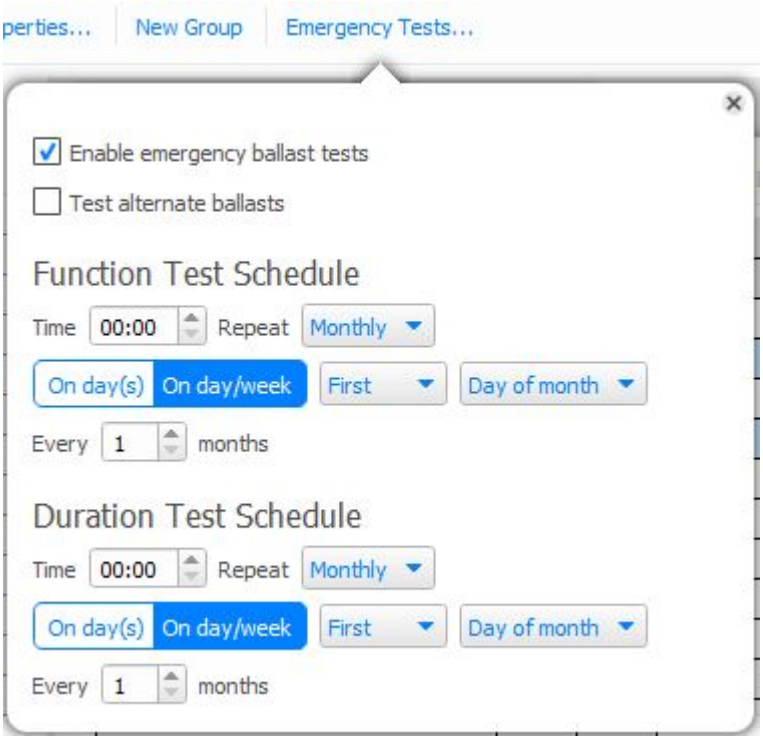
- Event filters:
  - Occupied
  - Vacant
  - Still occupied/vacant
  - Movement
  - No movement
- Timers
  - Configurable delay times for
    - Deadtime
    - Report
    - Hold

Light Sensors

- Enable/Disable Event messages
- Select primary instance group
- Event scheme
- Event priority
- Event filters:
  - Illuminance Level
- Timers
  - Configurable delay times for
    - Deadtime
    - Report Time
- Editors for hysteresis band
  - Hysteresis min (lux)
  - Hysteresis (percent)

Emergency Ballasts

Emergency DALI ballasts are set-up in the same way as standard DALI ballasts and support Highlight and Re-Addressing.



Emergency tests may be scheduled for all emergency ballasts in a project via the Emergency Ballast Tests tab on the right of the DALI window. You can override project settings on a per interface basis by using the emergency settings in the Interface Properties tab. Function and Duration tests can be scheduled independently of each other on a daily, weekly, monthly or yearly basis. The time of day that the tests run can also be specified. Choosing to test alternate ballasts will test every other patched ballast - the remaining ballasts will then be tested once the initial test is complete.

Test information is stored on the memory card of the controller responsible for that interface. This information can also be viewed via the web interface.

Upload configuration

Once you have configured all your DALI interfaces and programmed all your DALI scenes you must upload the configuration to each DALI interface in turn so that this data can be stored on the DALI ballasts themselves. Select an interface and press the Upload Configuration button, a progress bar will track this rather slow procedure.

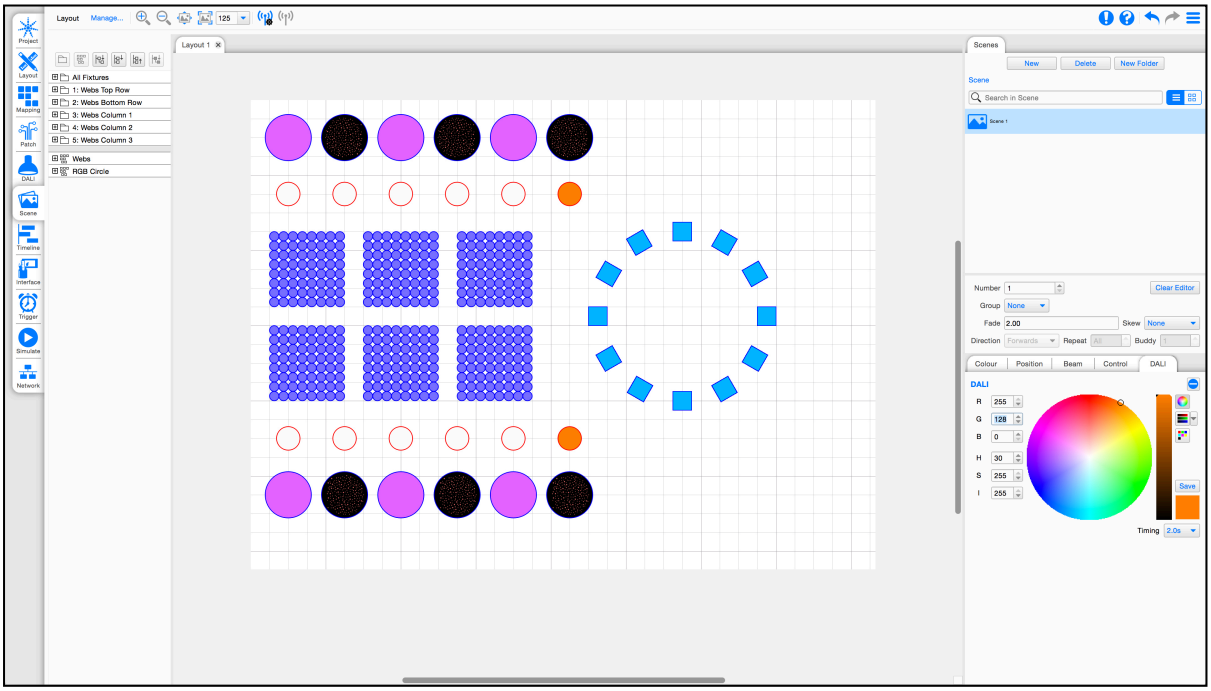
Scene

Keyboard Shortcuts

Ctrl+N	Create a new Scene in the current folder
Escape	Toggle last fixture selection
Ctrl+0	Reset the zoom
Ctrl+F	Zoom to fit the window
Ctrl++	Zoom in
Ctrl+-	Zoom out
Ctrl+ mouse wheel	Zoom in and out
Middle-click + drag	Zoom into the drawn rectangle
Alt+ mouse wheel (Shift+ mouse wheel)	Scroll Horizontally

Scene allows you to create single effects on any fixture within the project. These Scenes can be used later within Timelines or played back individually using triggering. These Scenes can control the following parameters either with static values or wave based effects:

- Intensity
- Colour
- Beam Shaping
- Position
- Beam Quality
- Control Parameters
- Effect Macros
- DALI (including RGB)



NOTE: Scenes are not available for the VLC/VLC+

Scene Management

Scenes are managed in the Scene browser in the right hand section of the Scene View.

From here you can create new scenes, delete scenes and sort scenes into folders.

To create a new scene

- Choose the New button in the Scene manager
- Give the Scene a name and press Enter.

To delete a Scene

- Select a Scene in the Scene Browser and press Delete/Backspace

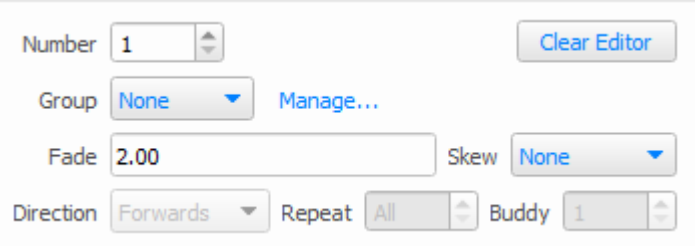
or

- Select a Scene and press the Delete button in the Scene browser

To create a New Scene Folder

- Press the New Folder button and Name the new folder.
- Scenes can then be created to the folder

Scene Properties



- Number - Each Scene has a unique number to refer back to it later on, which can be changed here
- Group - Scenes can be placed in a playback group to allow multiple Scenes and Timelines to be handled together within Actions. Click 'Manage...' to add, rename, renumber or delete playback groups. Numbers and names must be unique. The number of playback groups is limited to 100.
- Clear Editor - This option removes the currently selected Scene from the Scene editor and from Simulate.

The transition options below are a default and can be overridden when placed on a timeline. If the scene is run from an action, these default values will be used.

- Fade - the time taken to crossfade into this Scene
- Skew - Various different skews can be selected to alter how individual fixtures (and elements within fixtures in the case of compound fixtures such as battens and tiles) behave within the fade, default is None which means that all fixtures/elements fade together. Use skews to create “multi-part” fades so that fixtures/elements fly in one by one for example (set to Individually) - you may have to increase the fade time to clearly perceive the skew especially with lots of fixtures/elements.
- Direction - The ordering of a skewed transition depends on the fixture/element ordering within the group. The Skew Direction drop-down provides further ordering options such as Forwards and Backwards for additional flexibility. Additional groups can be created with different fixture/element ordering to achieve other skewed effects.

- Repeat - Specifies the number of adjacent fixtures/elements over which a skewed transition is applied, default is All meaning that the skew will span the entire selection. Typically you set this value to be equal to the number of pixels in a compound fixture or the number of fixtures in a zone or on a truss, experimentation is recommended as interesting effects can be achieved.
- Buddy - Specifies the number of fixtures/elements that will fade together within a skewed transition, default is 1 meaning that each fixture/element will fade independently. Set to 2 to make pairs fade together, 3 for threesomes etc. Again, experimentation recommended.

Scene Contents

To control a fixture in a scene

Select a fixture in the layout in the centre of the view. The fixture will get a red border to indicate selection.

Only fixtures with programming on them in a Scene will be controlled by the scene.

Once you add control of a parameter to a fixture it will get a blue border.

To directly control the colours of a fixture

Some fixtures within a Scene allow control of non-RGB/CMY colours directly within the Colour control. Choosing Direct Control with a fixture that supports additional colours to RGB will provide direct control of these colours

To add control of a parameter group

Locate the parameter group which contains the parameter you want to control.

Use the appropriate controls within the parameter group to set the value/s for the required parameter.

Static values

Within a parameter group, there will be controls for each parameter, either with buttons for specific values or a slider for a range of values.

FX

If you want to run an effect on the parameter, use the FX button to apply a wave effect to the parameter.

Position Effects

Clicking the FX button in the Position tab presents options for position effects when using fixtures with pan and tilt. The following parameters are available.

Shape:

The shape of the path the fixture will follow.

- Circle - Fixture rotates around an axis
- Pan Sine – Back and forth along X axis
- Tilt Sine – Back and forth along Y axis
- Eight – Figure of 8 loop
- Infinity – 90 degree rotated figure of 8 loop

Direction:

The direction in which the fixture moves around the shape.

- Forwards – Normal rotation
- Backwards – Reverse rotation

Period:

How long the fixture takes to complete one circuit of the shape in seconds.

Style:

Whether the fixture completes a circuit of the shape once or loops. Or, if using multiple fixtures, whether they all complete the shape simultaneously, spread, or one after the other.

- None – All fixtures follow the same path in a continuous loop.
- Spread – Fixtures repeat the same path with a time offset between fixtures.
- Chaser – Fixtures follow the same path one by one in a continuous loop.
- Once – All fixtures follow the same path only once.
- Chaser Once – Fixtures follow the same path one by one only once.

To remove control of a parameter group

Locate the parameter group that you want to remove and select the  Knockout button. This will remove the selected parameters from the Scene.

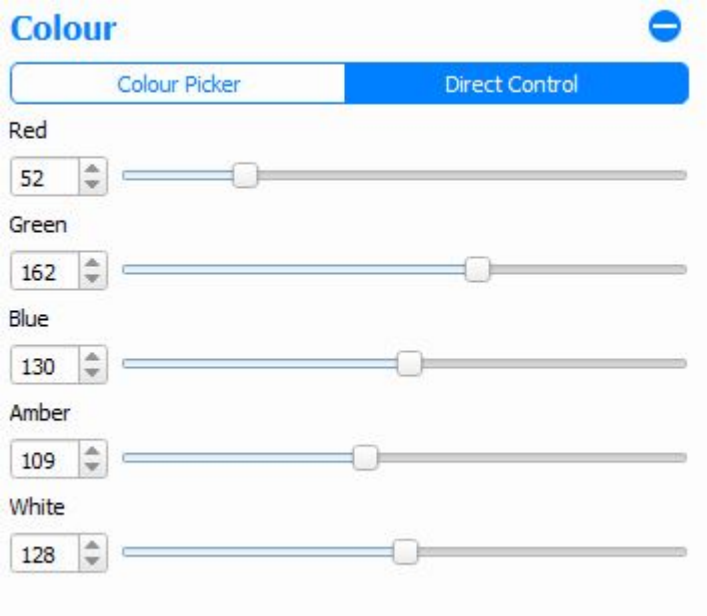
Scene Simulation

When you are creating a Scene, the Scene will be simulated within the Scene view, and the Simulate view.

This will be held until the Clear Editor button is clicked, allowing you to visualise a Scene and a Timeline in Simulate at the same time.

Direct Colour Control

If you have a fixture in your project with "complex" colour combinations e.g. RGBW, RGBA, RGBACL etc. these additional parameters cannot be controlled by the normal RGB colour wheel properly. This control is also available for DALI Type 8 RGBWAF fixtures (but not DALI Type 8 XY fixtures).



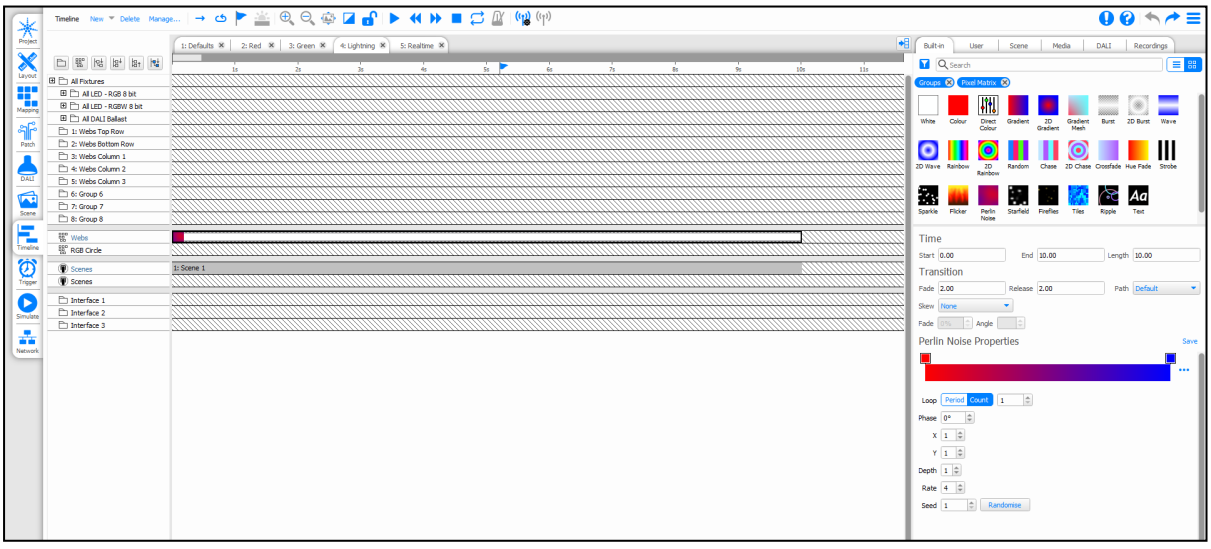
Within Scene, when setting a fixed colour for a fixture, there is an alternate control mechanism called Direct Control. This is only available for fixtures with additional colour channels, and allows direct control of the colour channels within that Scene.

Simulation of Direct Control within Designer is limited to just showing the RGB colour, but the full colour will be output using Output Live to the controller, so accurate colours can be set using this.

Working with Timelines

Keyboard Shortcuts

Ctrl+N	Create a New Timeline
Ctrl+D	Duplicate the current timeline
Ctrl+G	Go to timeline (enter name or number to filter the list); when one choice remains, press Enter to show the timeline
Ctrl+I	Show timeline properties
Ctrl+A	Select all timeline programming
Delete/Backspace	Delete selected timeline programming
Ctrl+left-click *while adding presets*	Toggle the behaviour of Auto-finish
Ctrl+drag *start/end of preset*	Snap to nearest preset, flag or waypoint
Shift+drag *preset*	Display the preset's end time and position
Ctrl+left-click *while adding flags*	Add flag and don't leave Add Flag mode
Esc	Finish adding presets or flags
Up/Down/Left/Right	Scroll the view
Space	Start/pause Simulation
Esc while simulating timeline	Stop Simulation
F *while simulating*	If in Add Flag mode, drop a flag at the simulation time
Ctrl+F	Zoom to fit the window
Ctrl++	Zoom in
Ctrl+-	Zoom out
Alt+ mouse wheel (Shift+ mouse wheel)	Scroll Horizontally
Ctrl+ mouse wheel (Cmd+ mouse wheel) *while over browser*	Increase / decrease timeline row height
Esc *while moving Gradient stop*	Cancel changing gradient



Timeline Tabs

Across the top of the Timeline Window, you will see a tab for each timeline that you currently have open. This allows you to quickly switch between timelines that are currently being worked on. You can close a timeline using the button on its tab. This only closes the view of the timeline, the timeline isn't deleted.

To see all timelines in the project, go to the Manage option in the View toolbar.

Creating a timeline

To create a timeline, click New and a fresh set of blank rows will appear to the right of each Browser entry. Use the Timeline Properties pane to name the timeline and adjust the timeline length, if necessary.

The default timeline length is 5 minutes (timeline length is really just a user interface setting and can generally be left at the default value, or increased to provide more programming space).

The New option also allows you to easily create a timeline with the settings for:

- Default timeline (internal time source)
- Real Time Timeline (24hr timeline with real time time source)
- Astronomical Timeline (24hr timeline including Sunrise and Sunset Waypoints)
- Timecode Timeline (time source set to timecode)
- Audio Timeline (time source set to audio)

Timeline row categories

There are five categories of timeline row which determine the preset type that can be deployed on them.

When you attempt to add a preset to the timeline, rows that the preset cannot be applied to will be dimmed.

Groups and fixtures

The majority of the timeline rows will be your groups of fixtures, the All... groups created by the system as you added fixtures and the groups you made to organise your programming. Only certain presets can be applied to this category. Click the plus sign to expand a group and expose its members:

A fixture, or element within a compound fixture, capable of colour mixing, for example an RGB LED or automated light with CMY colour mixing. Use the Group colour presets to set static or dynamic intensity and colour.

A fixture incapable of colour mixing, for example a conventional light with/without a scroller or automated light with only a colour wheel. Use the Group intensity preset to set static or dynamic intensity. Create Mover presets to control the scroller or colour wheel as required.

Matrices

These are the Pixel Matrices created in Setup and Mapping, only certain presets can be applied to this category. In many ways, creating matrices and working with these powerful presets is the preferred way to go with Designer.

Scene

Unlike the categories above these rows do not specify a fixture selection but are instead simply a placeholder for any Scenes created in the project. The fixture selection is a property of the Scenes depending as it does on the fixtures selected when the preset was created. More rows can be added as required by right-clicking in the Browser and selecting New Scenes Row, similarly to remove them.

DALI

These rows are for the DALI ballasts and groups and will only appear if some DALI fixtures have been added to the project, see DALI. Note that each DALI interface has its own set of rows due to the restrictions of the DALI

standard. Only DALI presets can be applied to this category.

VLC/VLC+ Rows

These rows can be used to output content to the VLC content target and can accept Matrix or Media presets. The Content Target on a VLC is directly linked to the VLC layout, so any presets placed on this row will be output to the whole layout.

If using a VLC+, the row will be able to expand to show Secondary and Target 3-8 rows (where enabled). The controller row corresponds to the Primary Content Target. These refer to the content targets which can be created in the Composition Editor.

Audio Rows

There are two types of audio row possible within a timeline:

- Simulation Audio
- Controller Audio

Simulation Audio

Simulation audio rows are available if Simulation Audio is enabled in the Simulate Mode. Audio clips can be dropped onto this row to playback on your computer when simulating the timeline. This is not uploaded to the controller for use during playback.

Controller Audio

Controller Audio is available if an LPC X, VLC or VLC+ are in use in the project and the Timeline Audio feature has been enabled.

This audio will be played back by the controller, and output from the audio connectors on the rear of the controller.

Timeline row priorities

While the Latest Takes Precedent system determines what should be rendered and output as presets come and go over time, it is the order of the rows that determines what should be rendered and output should two or more presets *with fixtures in common* start simultaneously, with rows higher up the list taking precedent.

For example a fixture may be a member of two groups with a preset applied to both starting at the same time. In this case the fixture will render the preset for the group higher up the list. Groups can be reordered in the browser simply by dragging them to new positions, although this will affect all timelines.

Accordingly, simultaneous presets placed on groups and fixtures have a higher priority than presets placed on matrices.

Browser controls and feedback

The Browser toolbar provides controls for expanding and collapsing groups and compound fixtures as well as highlighting rows with programming. The Browser provides useful feedback as to which rows contain programming; elements, fixtures and groups will be displayed in blue, compound fixture or group headings will indicate the presence of any programming on members even when collapsed.

Expand all

Expands all compound fixtures and groups so that all element rows are displayed. Items with programming will be shown in blue.

Expand all groups

Expands only groups so that all fixture rows are displayed. Items with any programming, even on a concealed member, will be shown in blue.

Collapse all

Collapses all so that only group rows are displayed. Groups with any programming, even on a concealed member, will be shown in blue.

Hide unused

Use this filter to hide all the unused rows, press again to turn off. Only items with any programming, even on a concealed member, will be shown in blue.

Selecting timelines

To choose which timelines are open for editing, go to the Manage option in the View toolbar. This will then display a dialog of all the timelines in the project. You can open a timeline by double clicking on it.

From this dialog you can also search your timeline list to narrow down the options within the timeline list.

Copying timelines

Timelines can be copied using the Copy button, the copy is a brand new instance that operates independently, useful for creating similar timelines.

Deleting timelines

Timelines can be deleted using the Delete button, a warning dialog will you prompt you to confirm.

Maintaining indefinite output

There are two ways of maintaining a timeline's output beyond the end of the last preset. This is a particularly important feature for architectural use where a simple wall panel could be used to recall "scenes" at random which would remain active indefinitely until another is recalled:

Hold

Press the Hold button to prevent the timeline from releasing at the end (the default). Presets will remain active until overridden, effects and media will continue to play. Press the button again to reinstate the release.

Loop

In Default and Audio Timelines, Press the Loop Timeline button to make a timeline loop indefinitely. If using a time source other than internal, setting to loop will allow the timeline to run again next time the timecode occurs. This is useful if you want to loop a sequence of presets immediately, or every time the timecode is used, or every day. Press the button again to remove the loop.

Release at End

In Real Time, Astronomical and Timecode Timelines, Release at End can be used to prevent a timeline from replaying when the time source loops. By default, when the time source loops, the timeline while go back to the start to stay in sync with the time source (e.g. real time across midnight). This can be disabled using the Release at End option

It's also worth noting that a Timeline Running condition won't detect timelines that are holding at end. A Timeline Onstage condition will detect a looping or held at end timeline as long as the timeline is affecting the output of at least one fixture. Timeline Started and Timeline Ended triggers will match whenever a looping timeline loops. A Timeline Ended trigger will never match a timeline that is holding at end.

NOTE: Projects with lots of timelines set to Hold or Loop can eventually overwhelm the Controller(s) if these timelines are not explicitly released when no longer required.

Auto-Release

If a Timeline has the Time Source set to Timecode or Real Time, the Loop option changes to Auto-Release. When set, the timeline will be released if the playhead reaches the end, otherwise it will continue running linked to the time source. If using Timecode, this means that it can restart if the timecode loops. If using Real Time, this means that the timeline will play again the next day.

Flags

Flags can be dropped onto timelines for use with triggers to create more complex presentations; perhaps incorporating remote sensors and conditional logic or triggering show control or AV equipment.

To set a flag, press the Add Flag button and drop it onto the timeline ruler at the required position. Hold down Ctrl (Cmd) while pressing Add Trigger Flag to drop multiple flags in a single session, press the button again to finish.

To Edit or Delete a flag, click on the flag that you want to edit and a properties dialog will appear:

From here you can adjust the flag time, give the flag a name or Delete the flag.

Name

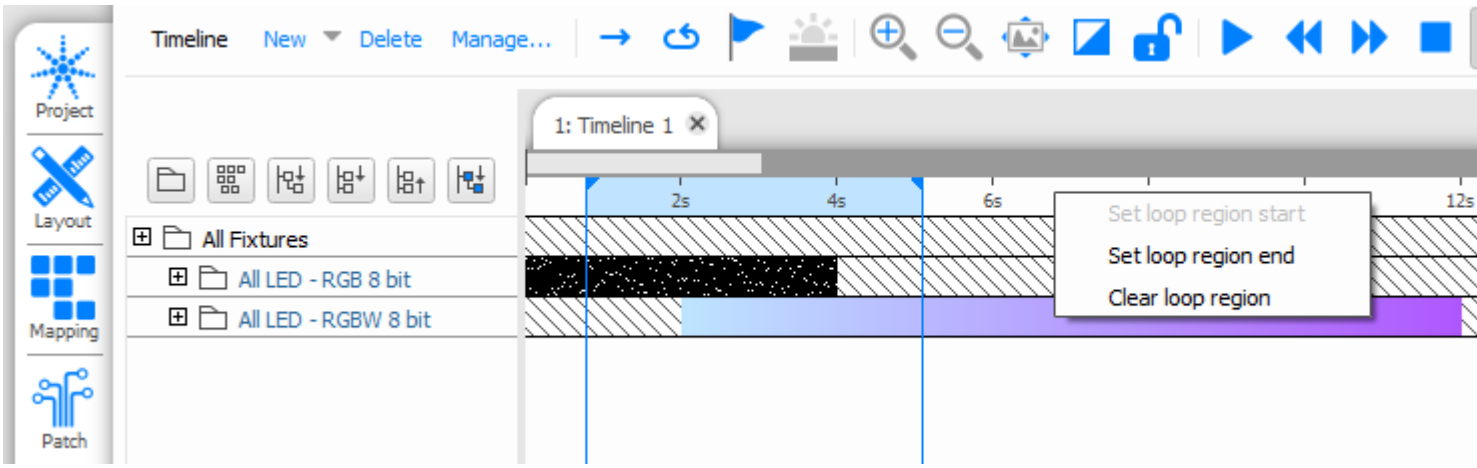
The name property of a flag is used within Flag triggers to easily identify the flag rather than using the time that the flag is set at.

Use the Triggers window to determine what these flags will do.

Learn timing

When simulating a single timeline, flags can be dropped interactively after pressing the Add Flag button to enter learn timing mode. Press F to drop a flag at each appropriate playback time then depress the Add Flag button to exit learn timing or click anywhere on the timeline (in which case a final flag will be dropped).

Loop regions



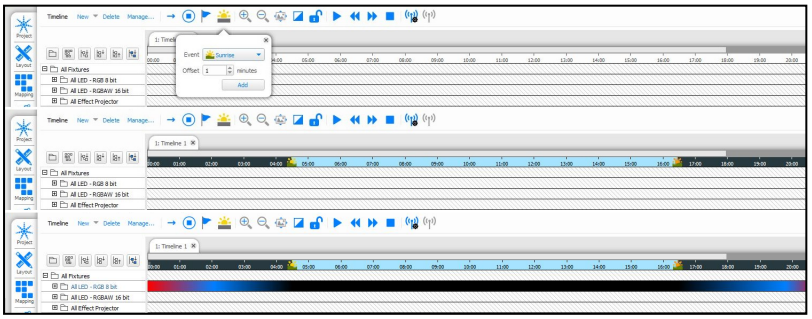
Loop regions are a programming tool which allow a section of the timeline to repeat during simulation.

To set a loop region, right-click anywhere in the timeline header to set start and end points. Vertical lines, which can be dragged to adjust their position, indicate the loop region. A loop region can also be set to selected timeline programming by right-clicking and selecting "Set loop region to selection".

Click the Enable Looping button in the toolbar to toggle the loop region on and off.

Loop regions are not sent to the controller as part of upload and do not affect playback.

Waypoints



To use waypoints, you must set the Time Source of the Timeline to Real Time

A waypoint is an astronomical time which should always display the programming at that point on the timeline. The timeline around the waypoint/s (as shown by the coloured section/s) will be rate shifted to ensure that the correct output is displayed at the specified astronomical time.

There are various Waypoints available, the same astronomical options available in the Astronomical Trigger:

- Nautical Dawn
- Civil Dawn
- Sunrise
- Sunset
- Civil Dusk
- Nautical Dusk

It is possible to place multiple waypoints in the same range, e.g. the range spans the 24 hour timeline with waypoints for Sunrise and Sunset. This would create three sections within the timeline which would be played back a different rates to ensure that the waypoints are hit at the required astronomical time.

The example above uses a Sunrise Waypoint and a Sunset Waypoint. The timeline will output Red at midnight, crossfading to blue at 3am, then holding blue then crossfading to black at Sunrise. The black output will be held until Sunset, when it will crossfade back to blue, then back to red. The rates of the sections between Midnight, Sunrise, Sunset and Midnight will all be adjusted so that the colours specified at each event are reached at the correct time for the astronomical event. As Sunrise gets earlier, the Midnight to Sunrise section will play back at a greater rate so it takes a shorter time to reach the Sunrise colour (black).

NOTE: To use Waypoints, you must have a Location set for the project.

/Locking timelines

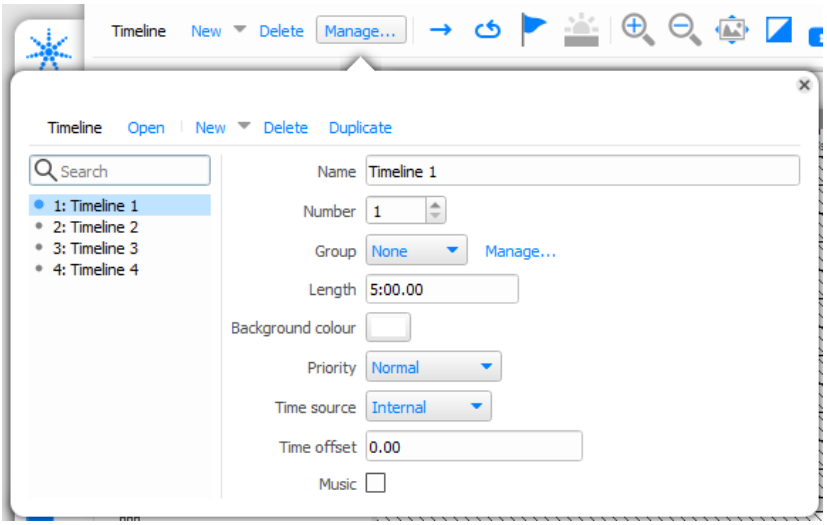
When clicking on presets to edit their properties it is sometimes all too easy to move or stretch them by accident so it is possible to lock a timeline using the Lock button on the timeline toolbar. When a timeline is locked it is only possible to edit the preset properties, moving or stretching them is prohibited. Press the Lock button again to unlock.

Timeline Properties

Keyboard Shortcuts

Shift *while selecting timelines*	Select a range of timelines (by choosing the first and last in the range)
Ctrl *while selecting timelines*	Select discrete timelines (each timeline selected while holding Ctrl will be added to the selection)
Ctrl + Mouse Wheel *while over browser*	Increase / decrease timeline row height

To change to properties of a timeline, select Manage... from the Timeline Toolbar. The dialog shown below will appear and you will be able to change the properties for the selected timeline/s.



Name

Give your timeline a name here, a descriptive name will help you identify the correct timeline when creating triggers and viewing the web interface's status and control pages.

Number

Every timeline has a unique number which is primarily for reference but can be changed if necessary. The timeline number is used to identify a timeline for creating triggers, for example when using LUA scripts, and when using the web interface's command line.

Group

The Group dropdown can be used to assign a timeline to a playback group. Clicking 'Manage...' opens a new popover where playback groups can be added, renamed, renumbered or deleted. Numbers and names must be unique. The number of playback groups is limited to 100.

See Actions for more information.

Length

The default timeline length is 05:00.00 (5 minutes) and you will need to increase this before placing or extending presets beyond this time. The maximum timeline length is 24 hours to prevent them becoming unmanageable - use triggers to stitch together multiple timelines to create longer time frames.

Background colour

Select the background colour of the timeline. Setting this to a colour that is not used within the timeline will make it easier to see some presets, e.g. a 255 intensity preset won't show up very well if the background colour is white.

Priority

Use the pull-down to select a priority level for the timeline. There are 5 priority levels which each feature an LTP+ timeline stack:

- High - the High Priority stack always plays above other timeline stacks
- Above normal
- Normal - the default priority
- Below normal
- Low - the Low Priority stack always plays below other timeline stacks

Time source

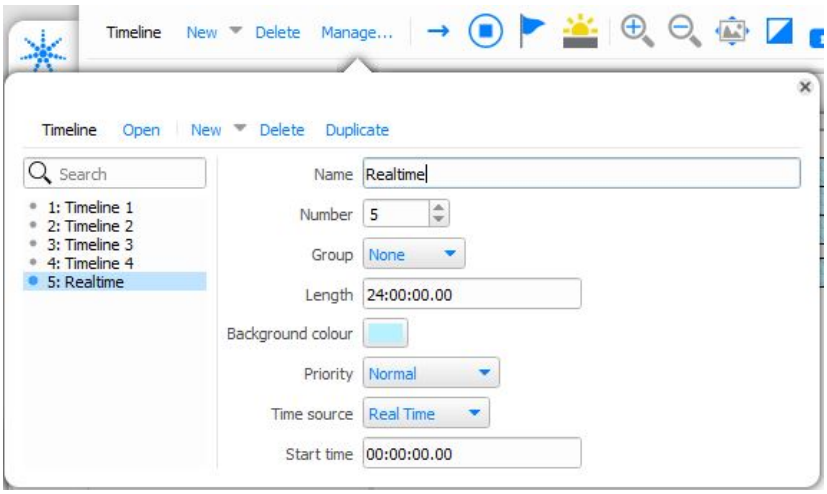
Use the pull-down to select a time source for the timeline to follow:

- Internal - the timeline will run autonomously although playback speed and position can be overridden using triggers.
- Real Time - the timeline will follow real time (see below)
- Timecode Bus - the timeline will follow one of the Timecode Buses (see below).
- Audio Bus - the timeline will follow one of four Audio Buses (see below).

Music

Enable tools for programming to music on the timeline. See Working with Music for more information.

Working with Real Time



When a timeline's time source is set to Real Time, it will always line up its position with the time according to the controller.

This means that timelines can be created that will always play the same effect at the same time every day.

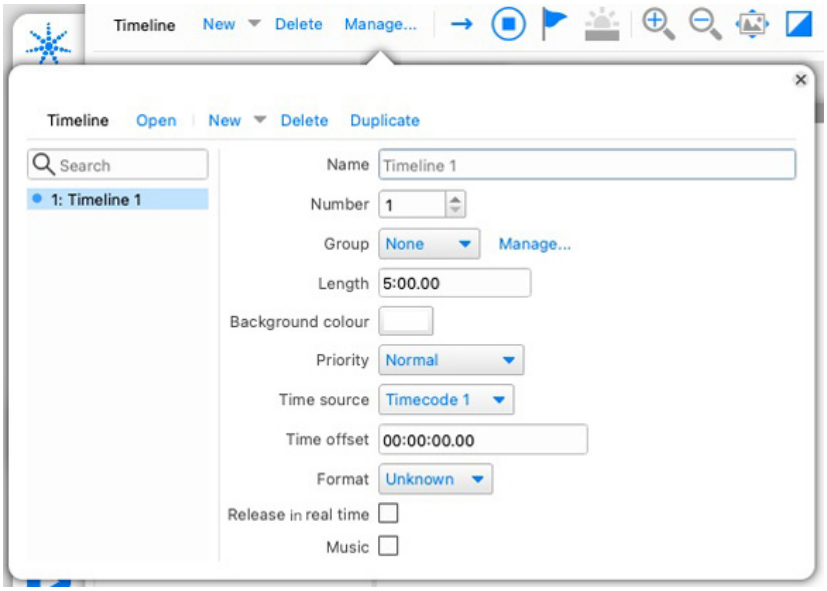
Timelines can be up to 24 hours long, but they can be any length less than that. If the timeline isn't 24 hours long, then a Start Time can be set to define when in the day these effects should be run.

Setting a timeline's time source to Real Time allows you to use waypoints on that timeline.

NOTE: When Simulating timelines, the playback rate can be increased to 60x normal speed to accommodate long Real Time timelines

Working with Timecode

By selecting one of the Timecode Buses, the timeline's time bar will display timecode values and the properties pane will give further options:



Time Offset

Timelines by default start at 00:00:00.00 (hours:minutes:seconds.frames) but the timecode source may not do so, the tape may have been "striped" with an offset of an hour (01:00:00.00) for example. Enter the source's starting value in this box to synchronise.

Format

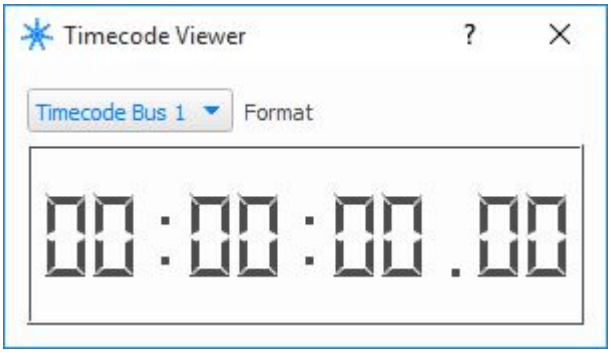
Timecode comes in four formats that depend on the source media used, select the appropriate format here (Film24, EBU25, SMPTE30 & NTSC30) to prevent missed frames and stuttering playback.

Release in real time

When selected, the timeline will follow the real time clock when releasing rather than following the timecode source.

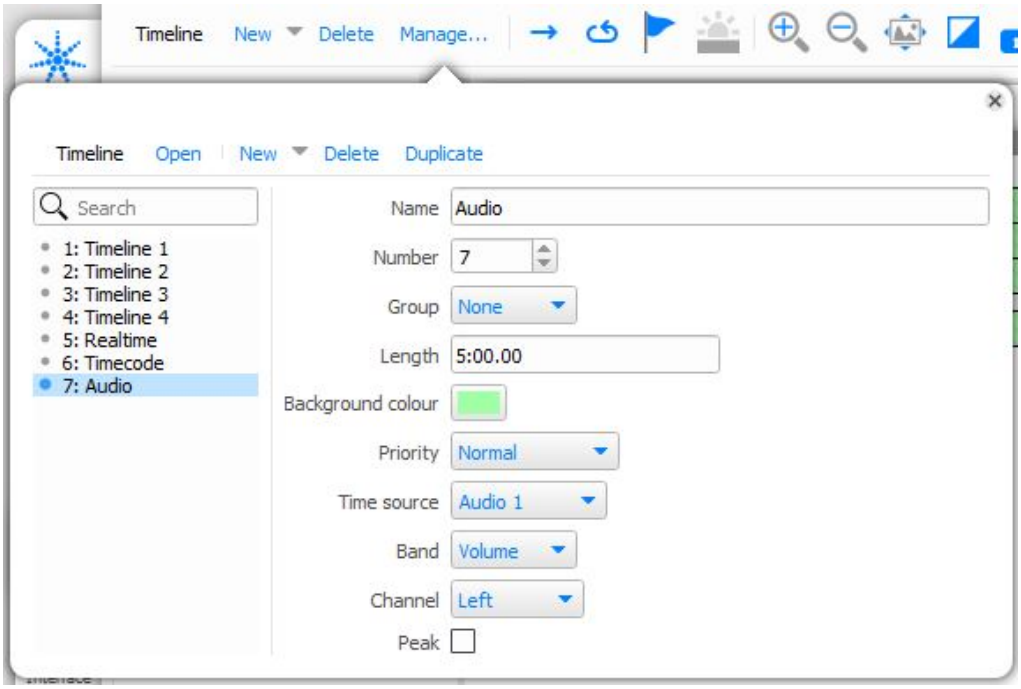
Timecode Buses

Timecode Buses are internal buses to which one patches the external timecode sources available to the system. These may be MIDI timecode (MTC) sources input via one or more LPCs' or RIO As' MIDI Inputs or linear timecode (LTC) sources input via one or more RIO As. You can use the Timecode Viewer available from the main menu to monitor each Timecode Bus:



Working with Audio

By selecting one of the four Audio Buses, the properties pane will give further options:

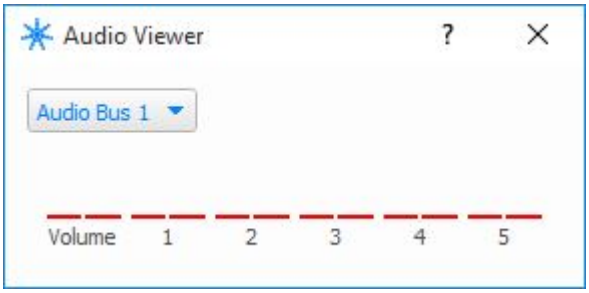


Band

The RIO A can generate up to 30 frequency bands (configured in Remote Device properties). Use this pull-down to select which band will drive the timeline.

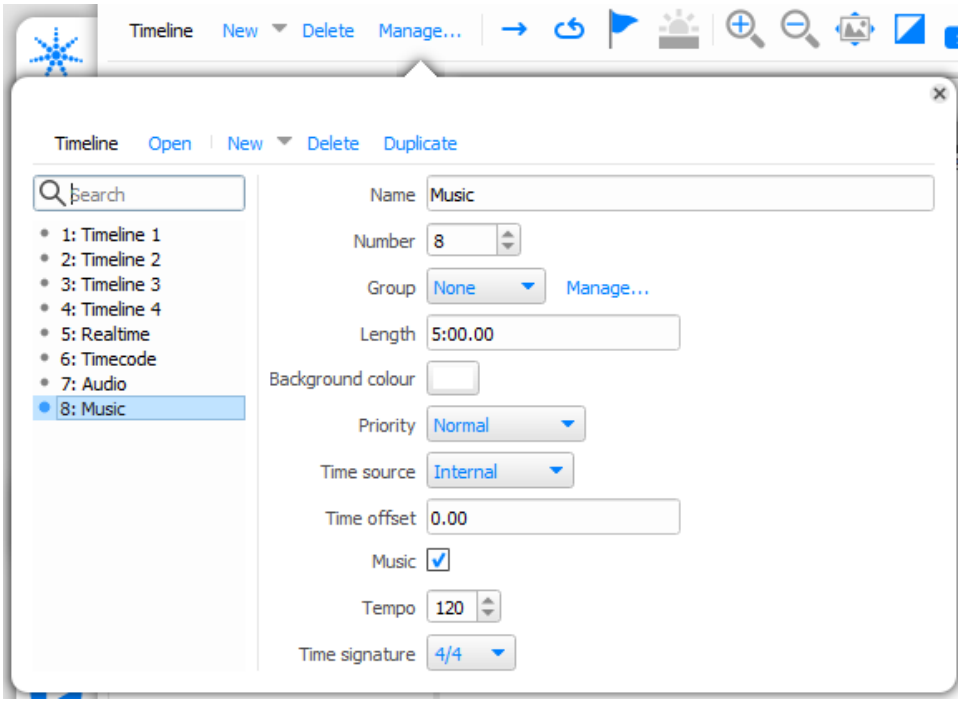
Audio Buses

The four Audio Buses are internal buses to which one patches the external audio sources available to the system via one or more RIO As. You can use the Audio Feedback window available from the Main menu to monitor each Audio Bus:



Working with Music

By selecting Music, the properties pane will give further options:



Tempo

Specify the number of beats per minute (BPM) on the timeline

Time signature

Specify the number of beats per bar

- None – use a sequence of beats with no bars
- 3/4
- 4/4
- 5/4
- 6/8

- 7/8
- 12/8

Timelines with music tools enabled show bars and beats in the timeline header. The cursor snaps to musical divisions when moving and adjusting presets. Simulation audio will automatically be enabled.

With music tools enabled, preset timing can be set in musical terms. These use the format Bars | Beats . PercentOfBeat and can be typed using just full stops. For example, entering a preset length of 4.2.5 will set the length to four bars, two and a half beats. Music timelines start at time 1 | 1.00.

The metronome button in the timeline toolbar can be used to activate a metronome click track which plays through simulation audio. The metronome will not be audible on controller output.

Designer can analyse music and detect a tempo. Right-click on any audio track and choose ‘Get tempo’. The tempo of the current timeline will be updated with the detected tempo. Designer may occasionally detect a tempo which is half or two times the correct tempo. If you suspect this has happened, tempo can be manually adjusted from the timeline's properties pane.

Changing the Timeline and Preset Defaults

Use Main Menu > Preferences on the main toolbar and select the Timelines tab to change these defaults. Here you can change the default background colour, timeline length and fade & release time of newly placed presets. See Preferences.

Working with Presets

Applying presets

At its most basic let's, for example, make a fixture or group of fixtures go green. To apply a preset to a timeline you will need to select it and then click to drop it onto the timeline. Select the "Colour" preset and drop this onto the appropriate timeline row so that it starts at the required time, say at 0 seconds. A 10 second long red strip (the default length and colour) will appear already selected for manipulation via the Preset Properties pane on the right. "Colour" preset properties are limited to colour and timing, use the colour picker to select green and set a fade time and skew as required.If you have the Simulate window open you can now simply click Start and you'll see these fixtures fade to green using the time and transition you have just entered and, after 10 seconds, fade back to black using the default release time of 2 seconds.

Presets can be moved and stretched on the timeline using the mouse to vary their start, end and length or alternatively you can type exact values into the Timing fields top right. Click View Transitions to display the fade and release timing graphically which can also be stretched using the mouse as an alternative to typing fade and release time values into the Timing pane.

So getting slightly more adventurous let's say you want the fixtures to remain green for longer, say 20 seconds, and then fade to a rainbow effect. Firstly either drag the end of the green preset to 20 seconds on the timeline or set the end or length value to be 20 seconds via the Timing fields. Now select the "Rainbow Effect" preset and drop this onto the timeline immediately following the green preset (so it starts at 20 seconds) and set its period to be 2 seconds with a "Spread" offset type. Again, use the Simulate window to view this new programming (click Reset then Start).

Tip: Hold Shift while dragging for finer resolution (centisecond). Hold Ctrl (Cmd) while dragging to snap to the start/end of other placed presets.

Programmed groups, fixtures or elements - i.e. those with at least one preset applied - are shown in blue in the Browser, unprogrammed remain black.

NOTE: When adding presets to the timeline, the timelines rows will change appearance to indicate which rows the preset can be dropped on. Darker rows can have the preset dropped onto them, lighter rows cannot.

Pre-configuring a preset

You can configure a preset before applying it to a timeline. When you select the preset from the preset browser, you can set the parameters as discussed below. When you then add the preset to the timeline it will have these properties.

This allows you to drop multiple presets with the same parameters onto multiple timeline rows.

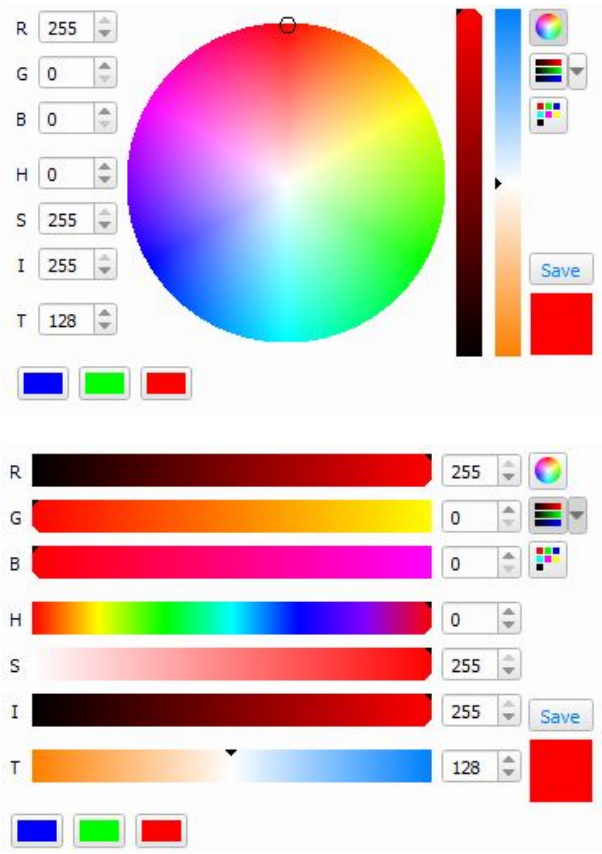
Filtering Presets

The Preset browser displays all of the available presets of the requested type (Built-in, User etc.). The Built-in and User browsers can filter the available presets to make it easier to find the preset you are looking for.

The filter button beside the search lets you choose the types of preset (Groups, Pixel Matrix, VLC) to display in the browser.

Colour picker & user palette

The majority of presets allow you to select one or more colours and so the colour picker and user palette is used to select a colour either graphically or numerically, two modes are provided:



A user colour palette is available with the third button to the right of the colour picker. The palette comes pre-populated with the primary and secondary colours, along with black and white. To add your own colour simply mix it using the picker and click the Save button. This will store the colour within your user palette The user palette is stored with the project.

Variable White fixtures

The temperature slider takes effect on fixtures with warm white and cold white control channels and on fixtures with a single colour temperature channel.

Transparency

Some preset types support transparency. This is where one of the colours within the effect is specified as transparent, allowing whatever programming is running underneath to be seen. It is a very powerful feature allowing some very specific effects to be achieved that would otherwise be impossible.

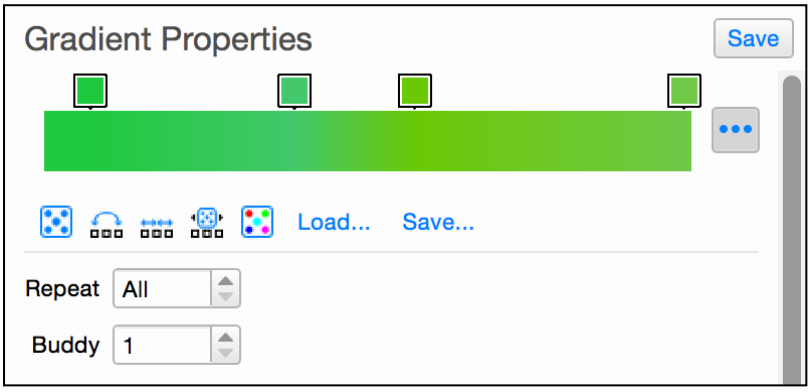
However it also allows you to break some of the usual rules of Pharos playback and so it may need some extra thought or experimentation to get the result you are looking for. Here are some tips on how to get the best from this feature:

- Before using transparency make sure there isn't a way to achieve what you want using the existing opaque presets. If there is another way to get what you want then that may make life simpler. Use transparency for those very specific effects that cannot be achieved any other way.
- It can sometimes be difficult to make transparent effects behave tidily when a timeline loops. Use it in timelines that don't loop or hold at end when you can, or make a point of turning off the transparent effect before the point when it loops.
- It may be easier to get the result you want by using several timelines. Often problems can be avoided by having one timeline that contains the transparent effects and putting the background non-transparent effects into a separate timeline.

- When you are using multiple timelines, don't forget about the timeline priority setting. This can be a way of ensuring that transparent effects stay on top while you change the background underneath.
- Custom presets and more complex presets cannot have transparency encoded into them; this is only available to some of the in-built preset types.

Gradient Options

When a preset includes a Gradient Property, additional properties can be shown using the advanced feature button •••



Random - Will create a random gradient, and open some options to better specify the random gradient

Reverse - Will flip the gradient horizontally

Distribute - Will spread out the colour stops evenly

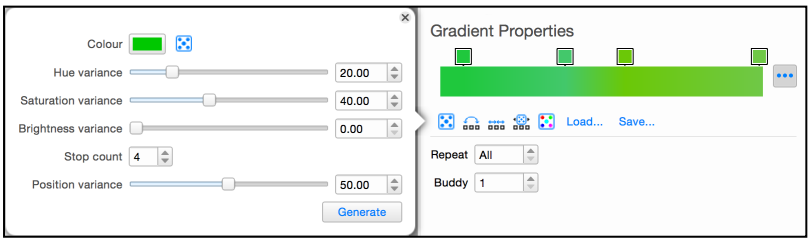
Randomise positions - Will set the colour stops to random positions, without changing the colours

Shuffle colours - Will randomly set the colours of the colour stops to one of the current colours without affecting the positions

Load - Allows you to load a saved gradient into the editor

Save - Allows you to save the gradient for future use

Random Gradient



Colour - The seed colour for the random colour, can be randomised.

Hue variance - Maximum variance of the gradient colour's hue

Saturation variance - Maximum variance of the gradient colour's saturation

Brightness variance - Maximum variance of the gradient colour's brightness

Stop count - Number of colour stops in the gradient

Position variance - Variance of the position of the colour stops from an even distribution

Saving Presets

When you have created a preset with settings that you may want to reuse, you can save it to your User presets.

The Save button will store the settings for the current preset and allow you to rename the preset.

These saved User presets can be found in the User tab of the preset browser, and added to the timeline in the same way as the Built-in presets. Any saved preset is also stored as a file within your documents folder, this can be useful should you wish to transfer or share them; they can be found at the following locations:

- Windows: \Documents\Pharos Controls\Designer 2\Timeline Presets
- Mac: /Documents/Pharos Controls/Designer 2/Timeline Presets

Scenes

Scenes can be dropped onto the Scene rows at the required time to synchronise their control with the rest of the presentation.

Tip: Scene rows are automatically added when you add a Scene to a row so that you have an empty row.

NOTE: Scene rows have the lowest priority within a timeline, so programming applied to a fixture in the Group section will always override its programming within a Scene.

DALI presets

DALI presets can be dropped onto the DALI ballasts and groups for each DALI interface in the project, see DALI. DALI presets should be thought of as commands instructing the DALI ballasts or groups to fade to a level or scene with the ballasts retaining this level or scene indefinitely regardless of the state of the timeline. Unlike DMX fixtures, there exists no notion of a released, default state and so DALI ballasts must be explicitly turned off with a preset. Beware that timelines set to loop will repeatedly run any placed DALI presets and thus reissue these commands until the timeline is released.

Tip: It may be simpler to separate DALI programming onto dedicated timelines and use triggers to synchronize them to the other fixture programming.

Copying presets

Presets can be copied (right-click>Copy) from one position and pasted (right-click>Paste) into a different position on the same timeline, on another timeline in the project or in a timeline in a different project, this helps speed up the process of applying programming from one set of fixtures onto another; preset parameters, timing and transitions will all be copied. Note that copying presets in this way creates brand new instances of presets that operate independently of each other.

Tip: Hold Shift while selecting Paste to place the copy with fisner resolution (centisecond).

Linking presets

If, however, you want to add more fixtures to an existing preset so as to operate on them all as one then drag the top or bottom edge of that preset up or down to include more rows of fixtures, this operation creates a linked preset. Note that any skewed timing or effects within the preset will now be rendered over the new, larger fixture selection - use Repeat to compensate if required.

A linked preset can be unlinked if desired by using right-click>Unlink to yield separate, identical instances.

Only presets on fixtures or groups can be linked.

Deleting presets

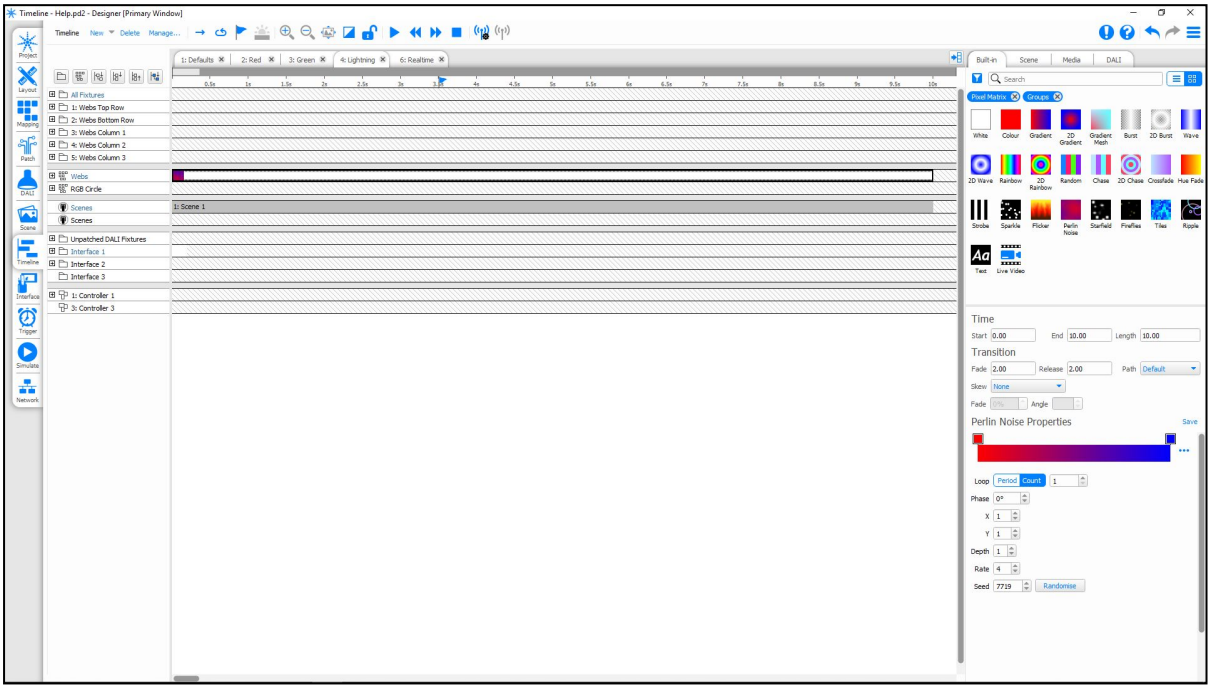
Presets can be deleted by pressing Delete, Backspace or using right-click>Delete.

Selecting multiple presets

To select more than one preset at a time for moving and editing properties or timing hold Ctrl (Cmd) while clicking to build the selection or Ctrl (Cmd) + A to select all.

Preset Types and Properties

The Timeline window is where you put your presentation together by dragging and dropping the built-in effects and your User Presets, Scenes, Media and Custom Presets onto your Fixtures, Groups and Pixel Matrices:



The window comprises 4 sections: On the left is the Browser, in the middle the Timeline editing area. top right are the folders of Built-in, User, Media, Scene, DALI and Custom Presets (although not all folders may be displayed). Below this is the Preset Properties pane which is divided into Timing, Transition, Name and Properties all of which you use to manipulate how a preset placed on a timeline is rendered.

Before creating a timeline it is worth covering the six preset types:

Built-in Presets

Designer comes with a range of Built-in presets which can be used to create a range of static, dynamic, wave based, and 2D effects over a group or array of fixtures. The Presets can be used on Fixtures, Groups, Pixel Matrices or VLC/VLC+ Content Targets.

White

Use with: Fixtures, Groups, Pixel Matrices, VLC/VLC+ Primary/Secondary

The most basic preset, sets an intensity and colour temperature.

You can also animate the white by selecting a wave shape.

- Shape - choose None (static intensity) or a dynamic effect (Sine, Cosine, Square, Triangle or Ramp Up)
- Base - The centre of the wave
- Size - the amplitude of the effect
- Period - the period of the effect in seconds
- Count - the number of times that the effect should repeat over the length of the preset
- Offset Style - choose None (all elements are the same intensity), Spread (the effect is spread over space as well as time) or Once (as None, but will stop after one period)

- Reverse - reverses the direction of the effect
- Repeat - the number of elements to repeat the effect over
- Buddy - the number of elements that will be set to the same intensity (if Buddy is greater than 1, the number of elements that are repeated over is Repeat multiplied by Buddy)

Colour

Use with: Fixtures, Groups, Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders a static colour fill. Use the colour picker, user palette or text entry fields (RGB or HSI) to select the colour.

You can also animate the colour by selecting a wave shape.

- Shape - choose None (static colour) or a dynamic effect (Sine, Cosine, Square, Triangle or Ramp Up)
- Base - The centre of the wave
- Size - the amplitude of the effect
- Period - the period of the effect in seconds
- Count - the number of times that the effect should repeat over the length of the preset
- Offset Style - choose None (all elements are the same intensity), Spread (the effect is spread over space as well as time) or Once (as None, but will stop after one period)
- Reverse - reverses the direction of the effect
- Repeat - the number of elements to repeat the effect over
- Buddy - the number of elements that will be set to the same intensity (if Buddy is greater than 1, the number of elements that are repeated over is Repeat multiplied by Buddy)

Direct Colour

Use with: Fixtures, Groups

Renders a static colour fill. Use the sliders to set the specific value for each emitter.

This preset only becomes active if there are fixtures in the project that support Direct Colour.

- Direct Colour - define the desired static colour using the intensity sliders for each emitter channel.

Note that the preset will display available colour channels based on all fixtures introduced within the project layout rather than on a per fixture/group basis. This means that the preset can be applied to a fixture/group that has fewer emitter channels, however, only the available channels will take effect on each fixture/group.

Crossfade

Use with: Fixtures, Groups, Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders a linear crossfade from the start colour to the end colour:

- Start Colour - the colour to start the crossfade in
- End Colour - the colour to end the crossfade in

Gradient

Use with: Fixtures, Groups

Renders a static multi-colour gradient over a group of fixtures:

- To change a colour, press on the coloured button, select a colour and press Ok
- To move a colour, click and drag the coloured button
- To add a new colour, click anywhere on the slider where there is no button
- To remove a colour, right-click on the coloured button
- Repeat - the number of elements between the start and end of the fan
- Buddy - the number of elements that will be set to the same colour in the fan (if Buddy is greater than 1, the number of elements that are repeated over is Repeat multiplied by Buddy)

2D Gradient

Use with: Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders a static multi-colour gradient on a matrix:

- To change a colour, press on the coloured button, select a colour and press Ok
- To move a colour, click and drag the coloured button
- To add a new colour, click anywhere on the slider where there is no button
- To remove a colour, right-click on the coloured button
- Type - the shape of the rainbow effect (Linear, Radial, Conical, Square, Noise, Perlin Noise or Bilinear)

If the Type is Linear, Radial, Conical, Square or Bilinear, the properties Angle, Repeat and Count are available:

- Repeat - the repeat style (None, Sawtooth, Triangle)
- Count - the number of repeats
- Angle - the angle in degrees of the gradient (Linear, Conical & Bilinear only)

If the type is Noise:

- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new seed

If the type is Perlin Noise:

- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new seed
- X - the horizontal scale (higher numbers will produce more variation horizontally)
- Y - the vertical scale (higher numbers will produce more variation vertically)
- Depth - the coarseness of the noise

Gradient Mesh

Use with: Pixel Matrices

Renders a dynamic gradient effect onto a Matrix or Target. Gradients are produced between multiple points on a grid

- Random - generate random settings for the effect
- Colours - pressing the colour buttons will prompt for a new colour for that step
- Interpolate - when checked the colours are used as points in a gradient to generate the colours in the mesh
- Colours - the number of colours used in the mesh
- Colour changes - the number of control points across the gradient
- Colour change variance - how evenly the colour points are spread across the gradient
- Weight variance - determines the effect each grid point has on the gradients around it

- Dynamic weights - determines whether the weights should change
- Dynamic weight rate - determines how much the weights vary
- Period - the number of seconds that the sequence takes to complete
- Count - the number of times that the sequence should complete over the length of the preset
- X points - the number of grid points along the X axis
- Y points - the number of grid points along the Y axis
- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new random seed

Burst

Use with: Fixtures, Groups

Creates a single wave in the chosen shape

- Base colour - the base colour
- Top colour - the colour of the pulse
- Transparency - select Opaque for none, Base or Top Transparent to superimpose the effect onto other programming
- Period - the number of seconds that the sequence takes to complete
- Count - the number of times that the sequence should complete over the length of the preset
- Phase - the offset of the pulse in degrees
- Shape - the shape of the pulse (Sine, Triangle, Square, Ramp Up or Ramp Down)
- Pulse width - the size of the pulse
- Pulse speed - the rate of the pulse
- Reverse - reverse the direction of the pulse
- Repeat - the number of elements to repeat the pulse over
- Buddy - the number of elements that will be set to the same colour in the pulse (if Buddy is greater than 1, the number of elements that are repeated over is Repeat multiplied by Buddy)

2D Burst

Use with: Pixel Matrices, VLC/VLC+ Primary/Secondary

Creates a single wave in the chosen 2D shape on a matrix

- Base colour - the base colour
- Top colour - the colour of the pulse
- Transparency - select Opaque for none, Base or Top Transparent to superimpose the effect onto other programming
- Period - the number of seconds that the sequence takes to complete
- Count - the number of times that the sequence should complete over the length of the preset
- Pattern - the type of pulse (Linear, Bilinear, Radial or Square)
- Angle - the angle in degrees of the pulse (Linear & Bilinear only)
- Reverse - reverse the direction of the pulse
- Shape - the shape of the pulse (Sine, Triangle, Square, Ramp Up or Ramp Down)
- Pulse width - the size of the pulse
- Pulse speed - the rate of the pulse

Wave

Use with: Fixtures, Groups

Renders a dynamic pulse of colour passing over another colour:

- Base colour - the base colour
- Top colour - the colour of the pulse
- Transparency - select Opaque for none, Base or Top Transparent to superimpose the effect onto other programming
- Period - the number of seconds that the sequence takes to complete
- Count - the number of times that the sequence should complete over the length of the preset
- Repeat - the number of elements to repeat the pulse over
- Buddy - the number of elements that will be set to the same colour in the pulse (if Buddy is greater than 1, the number of elements that are repeated over is Repeat multiplied by Buddy)
- Shape - the shape of the pulse (Sine, Triangle, Square, Ramp Up or Ramp Down)
- Pulse Width - the width of the pulse in percent (1 > 200%, if 100%, the pulse is half of the element width)
- Phase - the offset of the pulse in degrees
- Reverse Direction - reverses the direction of the pulse
- Invert Pulse - changes the starting position of the pulse

2D Wave

Use with: Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders a dynamic pulse of colour over another colour on a matrix:

- Base colour - the base colour
- Top colour - the colour of the pulse
- Transparency - select Opaque for none, Base or Top Transparent to superimpose the effect onto other programming
- Period - the number of seconds that the sequence takes to complete
- Count - the number of times that the sequence should complete over the length of the preset
- Pattern - the type of pulse (Linear, Radial, Conical, Square, Noise, Perlin Noise or Bilinear)
- Shape - the shape of the pulse (Sine, Triangle, Square, Ramp Up or Ramp Down)
- Pulse Width - the width of the pulse in percent (1 > 200%, if 100% the pulse fills half of the matrix)
- Reverse Direction - reverses the direction of the pulse
- Invert Pulse - changes the starting position of the pulse

If the Type is Linear, Radial, Conical, Square or Bilinear, the properties Angle, Repeat and Count are available:

- Repeat - the repeat style (None, Sawtooth, Triangle)
- Count - the number of repeats
- Angle - the angle in degrees of the pulse (Linear, Conical & Bilinear only)

If the type is Noise:

- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new seed

If the type is Perlin Noise:

- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new seed
- X - the horizontal scale (higher numbers will produce more variation horizontally)
- Y - the vertical scale (higher numbers will produce more variation vertically)
- Depth - the coarseness of the noise

Rainbow

Use with: Fixtures, Groups, Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders a dynamic rainbow effect cycling through hue:

- Colour - specifies the start colour of the rainbow (the saturation and intensity are maintained throughout the cycle)
- Period - the number of seconds that the rainbow takes to complete one cycle
- Count - the number of times that the rainbow should cycle over the length of the preset
- Reverse Colour - reverses the direction around the hue circle (default (unchecked) is clockwise)
- Offset Style - Choose None (all elements are the same colour) or Spread (the rainbow is spread over space as well as time)
- Scale - The relative size to scale the rainbow to before spreading it over the group (2 will make the spread double to group size, so half the rainbow is visible at a time)
- Reverse Direction - if Offset Style is Spread, reverse the direction of the spread in space
- Repeat - the number of elements between the start and end of the hue circle
- Buddy - the number of elements that will be set to the same colour in the rainbow (if Buddy is greater than 1, the number of elements that are repeated over is Repeat multiplied by Buddy)

2D rainbow

Use with: Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders a dynamic rainbow effect on a matrix:

- Colour - specifies the start colour of the rainbow (the saturation and intensity are maintained throughout the cycle)
- Period - the number of seconds that the rainbow takes to complete one cycle
- Count - the number of times that the rainbow should cycle over the length of the preset
- Pattern - the shape of the rainbow effect (Linear, Radial, Conical, Square, Noise, Perlin Noise or Bilinear)
- Reverse - reverses the direction of the wave

If the Type is Linear, Radial, Conical, Square or Bilinear, the properties Angle, Repeat and Count are available:

- Repeat - the repeat style (None, Sawtooth, Triangle)
- Count - the number of repeats
- Angle - the angle in degrees of the wave (Linear, Conical & Bilinear only)

Note that setting Repeat to None will only have an apparent effect when the Type is Radial. It behaves like Sawtooth with a Count of 1, except that the area outside the unit circle is filled with the same colour as the edge of the unit circle, rather than the effect continuing beyond a Count of 1.

If the type is Noise:

- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new seed

If the type is Perlin Noise:

- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new seed
- X - the horizontal scale (higher numbers will produce more variation horizontally)

- Y - the vertical scale (higher numbers will produce more variation vertically)
- Depth - the coarseness of the noise

 Random

Use with: Fixtures, Groups, Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders a dynamic chase through a random sequence of colours:

- Start colour - specifies the first colour of the sequence, all subsequent colours are relative to the start colour in a pseudo-random way (saturation and intensity levels are maintained)
- Steps - the number of steps in the sequence
- Seed - the seed of the pseudo-random sequence (copying this value to another preset will create the same random sequence)
- Randomise - picks a new seed
- Period - the number of seconds that the sequence takes to complete
- Count - the number of times that the sequence should complete over the length of the preset
- Offset Style - choose None (all elements are the same colour) or Spread (the sequence is spread over space as well as time)
- Reverse - reverses the direction of the wave
- Repeat - the number of elements to repeat the pulse over
- Buddy - the number of elements that will be set to the same colour in the pulse (if Buddy is greater than 1, the number of elements that are repeated over is Repeat multiplied by Buddy)
- Fade - the fade time in seconds between each colour in the sequence
- Hold - the time that each colour in the sequence is not fading

 Chase

Use with: Fixtures, Groups, VLC/VLC+ Primary/Secondary

Renders a dynamic chase through a user-specified sequence of colours:

- Colours - pressing the colour buttons will prompt for a new colour for that step
- Steps - the number of steps in the sequence
- Direction - choose Forwards, Backwards or Bounce (the latter uses two periods to complete)
- Period - the number of seconds that the sequence takes to complete
- Count - the number of times that the sequence should complete over the length of the preset
- Offset Style - choose None (all elements are the same colour) or Spread (the sequence is spread over space as well as time)
- Reverse - Reverse the direction of the pulse
- Repeat - the number of elements to repeat the chase over
- Buddy - the number of elements that will be set to the same step (if Buddy is greater than 1, the number of elements that are repeated over is Repeat multiplied by Buddy)
- Fade - the fade time in seconds between each colour in the sequence
- Hold - the time that each colour in the sequence is not fading

 2D Chase

Use with: Pixel Matrices, VLC/VLC+ Primary/Secondary

Creates a Chase, with multiple colours, and renders it using the same 2D options as the 2D Wave.

To change the colours, selected the block above the colour display and chose a new colour.

- Steps - the number of colour steps in the Chase
- Direction - the order the chase works through the specified colours (Forwards, Backwards, or Bouncing)
- Period - the number of seconds that the sequence takes to complete

- Count - the number of times that the sequence should complete over the length of the preset
- Phase - the position on the underlying waveform that the effect starts at.
- Pattern - the type of pulse (Linear, Bilinear, Radial, Conical, Square, Noise or Perlin Noise)
- Angle - the angle in degrees of the pulse (Linear, Conical & Bilinear only)
- Curve - the amount that each repeat of the Conical effect curves round. (Conical only)
- Repeat - the repeat style (None, Sawtooth, Triangle)
- Count - the number of repeats
- Reverse - reverse the direction of the pulse
- Fade - the crossfade time from each colour to the next.
- Hold - the time that each colour holds for before starting the next fade

Hue Fade

Use with: Fixtures, Groups, Pixel Matrices, VLC/VLC+ Primary/Secondary

Performs a fade in hue between two defined points:

- Start colour - defines the hue at the start of the preset, and the saturation and brightness throughout the preset
- End colour - defines the hue at the end of the preset; saturation and brightness will be the same as the start colour
- Reverse - reverses the direction of the hue fade

The start and end colours will share the same saturation and brightness; editing the saturation or brightness for one colour will edit the other as well.

The fade time between the colours is determined by the length of the preset on the timeline.

Strobe

Use with: Fixtures, Groups, Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders a dynamic colour strobe effect on black:

- Colour - specifies the flash colour
- Transparency - select Opaque for none, Base Transparent to superimpose the effect onto other programming
- Period - the interval in seconds between the start of each flash
- Duration - the length in seconds of the flash

Sparkle

Use with: Fixtures, Groups, Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders a dynamic random sparkle effect:

- Base colour - the colour of the background
- Spark colour - the colour of the spark
- Transparency - select Opaque for none, Base or Spark Transparent to superimpose the effect onto other programming
- Period - the rate of the effect (larger numbers are slower)
- Density - the density of the effect in percent (higher numbers, more sparks)

 Flicker

Use with: Fixtures, Groups, Pixel Matrices

Renders dynamic, random flickering over a colour gradient:

- To change a colour, press on the coloured button, select a colour and press Ok
- To move a colour, click and drag the coloured button
- To add a new colour, click anywhere on the slider where there is no button
- To remove a colour, right-click on the coloured button
- Period - the period of the effect in seconds
- Sub - the amplitude of the low frequency perturbation
- First - the amplitude of the fundamental flicker frequency
- Second - the amplitude of the second harmonic
- Third - the amplitude of the third harmonic
- Seed - used to offset the effect; click the Randomise button to generate a random value
- Uniform - apply the Seed value as the offset for all fixtures in the group, or use it as a seed to generate random offsets for each fixture in the group

Each of the sliders corresponds to a sine wave of a specific frequency. The frequency of Sub is defined by the Period - the default is 30 seconds (1/30Hz). First will be twice as fast, Second twice as fast again and Third twice as fast as Second. The value of each sine wave is used to fetch a value from a set of pre-generated random values and the four results are summed. The sum is used to select a position in the colour gradient to output to the fixtures. Mix the different frequency components using the sliders to select how much of each component you want.

So if you are looking for a relatively steady flicker you might have a lot of Sub, with a little bit of Third to stop it looking too regular. If you want a more chaotic looking flicker then you might have less of Sub and First and more of Second and Third. It really is something you have to experiment with. If you want the overall flicker to have a different speed change the Period and everything will shift accordingly.

If you've got a set of slider values that you like and you want to copy the effect to another group, but not have both groups flickering identically, then just click the Randomise button to change the offset.

 Perlin noise

Use with: Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders a smoothly-varying noise effect:

- To change a colour, press on the coloured button, select a colour and press Ok
- To move a colour, click and drag the coloured button
- To add a new colour, click anywhere on the slider where there is no button
- To remove a colour, right-click on the coloured button
- Period - the number of seconds that the noise takes to loop
- Count - the number of times that the noise should loop over the length of the preset
- Phase - the phase shift of the underlying wave
- X - the horizontal scale (higher numbers will produce more variation horizontally)
- Y - the vertical scale (higher numbers will produce more variation vertically)
- Depth - the coarseness of the noise
- Rate - the rate at which the noise varies
- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new seed

Starfield

Use with: Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders a radiating star field:

- Space colour - the colour of the background (on the VLC this can be set to transparent)
- Star colour - the colour of the stars (on the VLC the opacity can be set for this)
- Speed - the speed of the stars
- Star Count - the number of stars to show
- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new seed

Opacity on stars will show through the Space colour and transparency on Space will show through any effect playing below the starfield.

Fireflies

Use with: Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders an effect of "fireflies" randomly flying about

- Period - the number of seconds that the fireflies take to loop
- Count - the number of times that the fireflies should loop over the length of the preset
- Background - the colour of the background (on the VLC this can be set to transparent)
- Start colour - the starting colour of the fireflies
- End colour - the end fade colour of the fireflies
- Opacity - the opacity of the fireflies (100% is fully opaque)
- Speed - the speed that the fireflies move at
- Duration - the lifetime of the fireflies
- Fireflies - the number of fireflies visible at any one time
- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new seed

Opacity on fireflies will show through the background colour and transparency on background will show through any effect playing below the fireflies.

Nebula

Use with: VLC/VLC+ Primary/Secondary

Renders an effect of a nebulous gas cloud

- Period - the number of seconds that the nebula take to loop
- Count - the number of times that the nebula should loop over the length of the preset
- Background - the colour of the background (on the VLC this can be set to transparent)
- Start colour - the starting colour of the particles
- End colour - the end fade colour of the particles
- Opacity - the opacity of the particles (100% is fully opaque)
- Speed - the speed that the fireflies move at
- Duration - the lifetime of the particles
- Particles - the number of particles visible at any one time

- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new seed

Opacity on particles will show through the background colour and transparency on background will show through any effect playing below the nebula.

Tiles

Use with: Pixel Matrices, VLC/VLC+ Primary/Secondary

Renders a random geometric tile arrangement with varying colours.

- Gradient:
  - To change a colour, press on the coloured button, select a colour and press Ok
  - To move a colour, click and drag the coloured button
  - To add a new colour, click anywhere on the slider where there is no button
  - To remove a colour, right-click on the coloured button
- Tile width - the maximum width of a tile
- Tile height - the maximum width of a tile
- Tile splits - the maximum number of times each tile could be randomly split in half
- Steps - the number of colour steps across the gradient to use to set the tile colours
- Period - the number of seconds that the colour steps takes to loop
- Count - the number of times that the colour steps should loop over the length of the preset
- Fade - the crossfade time from each colour to the next.
- Hold - the time that each colour holds for before starting the next fade
- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new seed

Ripple

Use with: Pixel Matrices

Renders ripples onto the matrix:

- Gradient - the colours that the ripples fade through
- Background colour - the background of the effect
- Antialiasing - Enable to add fades around the edge of the ripples
- Ripple fade out - the fade path of the ripple
  - None - the ripple doesn't fade out
  - Linear - the opacity of the ripple decreases linearly with time.
- Period - the number of seconds that the ripples take to loop
- Count - the number of times that the ripples should loop over the length of the preset
- Lifetime - the length of time that the ripple exists for
- Speed - the rate of growth of the ripple
- Type - the type of colour fill to use
  - Solid - fade from the start of the gradient to the end over the lifetime of the ripple
  - Gradient - apply the gradient to the body of the ripple
  - Random - randomly select a single colour from the gradient for each ripple
- Filled - when enabled, a filled circle is rendered, when disabled a hollow circle is rendered
- Ripple width - the thickness of the ripple (when filled, the gradient is rendered over this distance)
- Ripples - the number of ripples to create during the loop
- Seed - the seed of the pseudo-random noise (copying this value to another preset will create the same noise)
- Randomise - picks a new seed

Text

Use with: Pixel Matrices, VLC/VLC+ Primary/Secondary/Overlay

Renders a scrolling text message on a matrix:

Renders a text message which can be changed at runtime:

- Base colour - the base colour
- Text colour - the colour of the text
- Transparency - select Opaque for none, Base or Text Transparent to superimpose the effect onto other programming
- Period - the number of seconds that the message takes to scroll over the matrix
- Count - the number of times that the message should scroll over the length of the preset
- Text - the text to render
- Font - the font to use to render the text (see the Fonts dialog below)
- Scroll - the scroll direction of the text
- Blend - the amount of crossfade at each end of the matrix
- Seamless loop - if the text is set to scroll, setting this will remove the gap between the end and the start
- Align - if the text is set to not scroll, this is the alignment of the text (Left, Centre, Right)
- Orientation - the rotation of the text
- Flip - flips the text top to bottom
- Mirror - flips the text left to right
- Offset - set a offset amount on the Y axis of the matrix

To configure the font used by the Text preset, press the Edit... button next to the font picker to open the Fonts dialog:

- Select a font from the Font picker
- Press New to create a new font
- Press Delete to delete the selected font (note that you cannot delete a font that is in use in the project)
- Set the font's name in the Name property
- Use Family, Size, Bold and Italic to set the appearance of the font
- Press Ok to close the Fonts dialog

NOTE: Editing a font will change all Dynamic text presets that use that font not just the currently selected preset(s).

The Text preset allows you to change the text after uploading the project to a Controller. To do this, you need to specify which parts of the text are going to change and which parts will remain the same.

For example, to show the opening time of a venue, you might set the Text property to "Opening Time: <open>". This creates a text slot called 'open' which you can change the value of. You can have more than one slot specified in the Text property, for example "Opening Time: <open> Closing Time: <close>".

To set the initial text for a text slot, press the Configure... button next to the Text property to open the Text Slot Configuration box:

- Click in the Default Value cell of a slot to edit the text stored in that slot
- You can remove unused text slots by pressing Remove
- Press Ok to save changes and Cancel to discard changes

The Set Text Slot trigger action allows you to change the value of a text slot from a trigger.

There are two built-in slots, <time> and <date>, which show the current time and date respectively. You can change the format of how the time and date are displayed in the Text Slot Configuration box. Press the

Configure... button next to the Text property to open this dialog. At the bottom of the dialog you can select from some standard time and date formats, or type your own using the following codes:

%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%c	Date and time representation
%d	Day of the month (01-31)
%H	Hour in 24h format (00-23)
%I	Hour in 12h format (01-12)
%j	Day of the year (001-366)
%m	Month as a decimal number (01-12)
%M	Minute (00-59)
%p	AM or PM designation
%S	Second (00-61)
%U	Week number with the first Sunday as the first day of week one (00-53)
%w	Weekday as a decimal number with Sunday as 0 (0-6)
%W	Week number with the first Monday as the first day of week one (00-53)
%x	Date representation
%X	Time representation
%y	Year, last two digits (00-99)
%Y	Year
%Z	Timezone name or abbreviation
%%	A % sign

All other text is used verbatim. The computed output will be truncated to 255 characters.

Live video

Use with: Pixel Matrices, VLC/VLC+ Primary/Secondary

Displays live video on a matrix (LPC X only) or VLC/VLC+ layout:

- X&Y offset - map to the top left pixel of interest on the incoming DVI image
- Black level - the intensity level below which rgb(0,0,0) should be output

To use Live video, the video input settings will need to be set.

User Presets

User Presets will show any preset configurations that you have saved.

User presets can be renamed by right-clicking on them and selecting Rename, and deleted by right-clicking and selecting Delete

Scene

Scenes can be added to timelines as a way of generating reference palettes, or controlling advanced parameters of complex fixtures.

Scene (User named)

Use with: Scenes

The presets that you optionally created using Scene to create static effects to play back within the project.

NOTE: If no Scenes have been created, the Scene folder will not be displayed.

Media Presets

Video (User named)

Use with: Pixel Matrices, VLC/VLC+ Content Target

The presets that you optionally created using the Media window to import still and moving images into your project. These presets have spatial awareness when applied to Pixel Matrices and VLCs in that the media clip will be resized to fit the Pixel Matrix's Render Window or the VLC/VLC+ Content Target.

If applied to a VLC or VLC+ Content Target, the media preset can be cropped using the Crop top, Crop bottom, Crop left and Crop right to specify the area of the media to be output to the Content Target. The crop value determines the number of pixels that are cropped off each side of the media preset.

Media clips in the Preset Browser can be managed in the same way as in Mapping Mode, by Right-clicking on the media clip, and New Media clips can be added using the New button at the top of the Preset Browser.

Media Presets have the following properties:

- Period - the number of seconds that the media plays for within the preset
- Count - the number of times that the media should play during the preset
- Start - set the in point of the media
- End - set the out point of the media
- Direction - the direction to play the media in (forwards or backwards)
- Temperature - Adjust the colour temperature of the media
- Flip - reflect the media vertically
- Mirror - reflect the media horizontally
- Crop left - the number of pixels to crop off the left hand side
- Crop right - the number of pixels to crop off the right hand side
- Crop top - the number of pixels to crop off the top side
- Crop bottom - the number of pixels to crop off the bottom side
- Black level - the level below which rgb(0,0,0) should be output

NOTE: If no Media or Audio Presets have been created, the Media Presets folder will not be displayed.

Audio (User named)

Use with: Audio Rows

The presets that you optionally created using the Media window to import audio into your project. These presets can be used for Simulation Audio or Controller Audio (on LPC X, VLC or VLC+).

Audio Presets have the following properties:

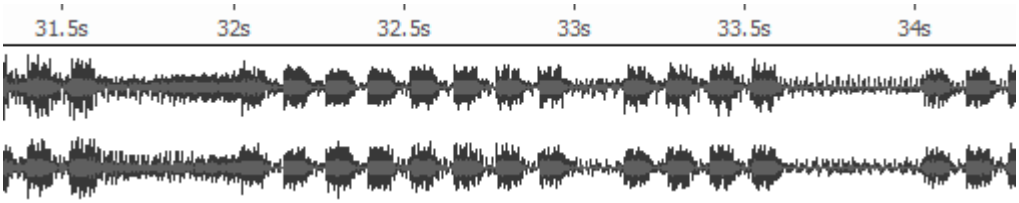
- Start - set the in point of the media
- End - set the out point of the media

The Audio Preset on the timeline will display a representation of the waveform of the audio file in the preset.

The top and bottom half of the waveform in dark grey show the maximum and minimum amplitude of the wave at that point respectively. The root mean square (RMS) value of the wave is shown in light grey.

Mono audio files are represented as a single waveform.

Stereo audio files are represented as a single waveform but will expand to show individual left and right waveforms when timeline row height is increased. The left channel is shown at the top and right at the bottom.



NOTE: If no Audio or Media Presets have been created, the Media Presets folder will not be displayed.

DALI Presets

Like Scenes, DALI presets do not have a length, only a transition, with the settings persisting until another DALI preset is encountered.

However, unlike Scenes, DALI presets will persist even if the timeline is released. Indeed, since they are just commands to tell the DALI ballasts what to do, even power-cycling the Controller will make no difference; the settings will persist until a new command is issued or the ballasts themselves power-cycled.

Set level

Use with: DALI ballasts, groups or interfaces

Used to set a DALI fixture or user created group to a level (0>254, 255), and select a fade time from the pull-down list of DALI fade times. See DALI regarding creating DALI groups.

Set Colour

Use with: DALI ballasts, groups or interfaces (that support Colour commands)

Renders a static colour fill. Use the colour picker, user palette or text entry fields (RGB or HSI) to select the colour.

DALI fade time can be set as part of the preset.

Set Colour Temperature

Use with: DALI ballasts, groups or interfaces (that support Colour Temperature commands)

Renders a static colour temperature fill. Use the level and colour temperature picker to select the colour temperature.

DALI fade time can be set as part of the preset.

DALI Scene (user named)

Use with: DALI ballasts, groups or interfaces

Used to recall a DALI scene that you created and uploaded, and select a fade time from the pull-down list of DALI fade times. See DALI regarding creating DALI scenes.

NOTE: If there are no DALI fixtures in the project, the DALI Presets folder will not be displayed.

Recordings

Recordings made in DMX Record can be used on timelines.

Recording (user named)

Use with: Fixtures, Groups, Pixel Matrices

Recordings created using DMX Record can be imported and added to timelines.

Recordings apply to the timeline row on which they are placed. Only the channels patched to that row will respond to the recording, allowing selective playback of components of a larger recording.

Recordings in the preset browser can be managed by right-clicking on the recording. New recordings can be added using the import button at the top of the preset browser.

Recordings have the following properties:

- Period - the number of seconds that the recording plays for within the preset
- Count - the number of times that the recording should play during the preset
- Start - set the in point of the recording
- End - set the out point of the recording

The preset browser will also display recording metadata including the recording's name, description, source, duration and frame rate.

Custom Presets

Custom Preset (user named)

Use with: Pixel Matrices

Renders a Custom Preset that you have optionally created using the Media window:

- Period - the number of seconds that the effect takes to complete one cycle
- Count - the number of times that the effect should cycle over the length of the preset

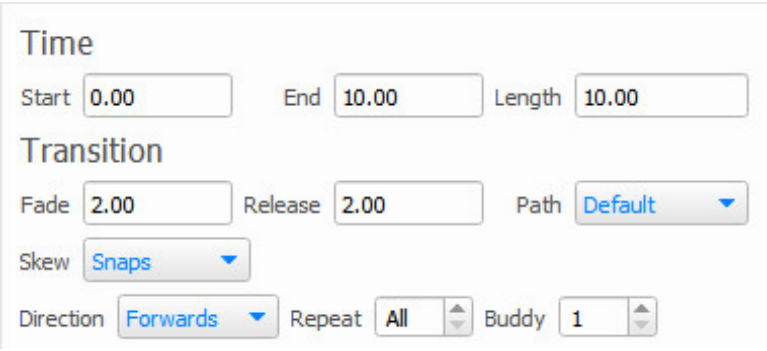
In addition, Custom Presets may define a number of properties that can be set for each instance of that Preset on the timeline.

However, a drawback of the custom presets is that they cannot have transparency encoded into them. Effects with transparency can be layered on top of the preset, but any effect "below" the preset will be overwritten.

NOTE: If no Custom Presets have been created, the Custom Presets folder will not be displayed.

Timing, Transitions & Precedent

It is often said that good lighting is as much about timing as anything else so it is important to understand the concepts of timing and transitions used throughout Designer:



Timing

Timing values pertain to presets placed on timelines and determine the Start, End and Length times and may be numerically set as an alternative to dragging.

Transition timing

Transition values pertain to presets placed on timelines and determine what sort of cross fade is rendered. The use of interesting transitions can transform your project so it is well worth spending some time experimenting to see what can be achieved using these properties:

Fade

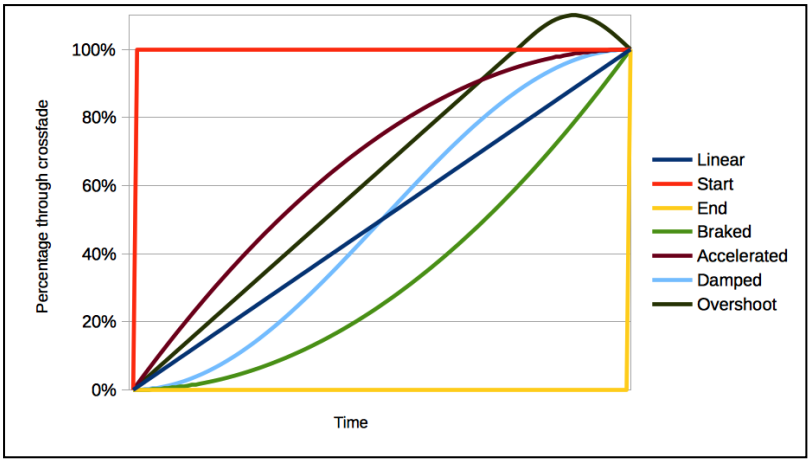
Sets the overall length of the transition, default value is 2 seconds.

Release

Sets the time used for a preset to “release” its fixtures when it completes, default value is 2 seconds. This value is not used if another preset is placed on the timeline immediately after this one - in that case the fade time of the second preset will be preferred.

Path

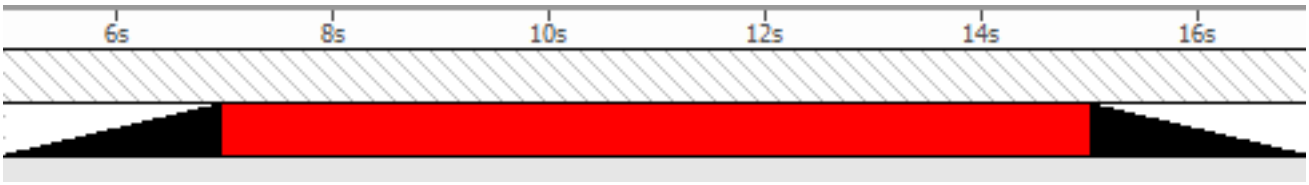
Specifies the cross fade path, default is Default meaning that each pixel or parameter will use the library default path (or Dimmer Curve if specified), typically Linear for intensity/position and Start for colour/gobo wheels. A variety of paths are provided which provide overall crossfade paths.



Additional paths add the ability to specify the order that colour and intensity channels crossfade in, such that the colour can fade then the intensity, for example.

- Col at Start
- Col at End
- Int at Start
- Int at End
- Colour First
- Intensity First

Viewing Transitions



The View Transitions button can be used to display the transitions for each preset on the timeline.

Transition Skews

Skews are transitional effects that can be applied to the fade-in time for built-in presets (1D & 2D). Skews work by changing (or skewing) the timings when the fade-in is applied to groups and arrays of fixtures, thereby creating a variety of possible transition effects. Each skew type will affect both how the fade effect applies to each fixture, and how it propagates through the group or matrix.

There are two completely separate lists of available skew types for 1D and 2D built-in effects.

Group & Scenes (1D)

Skew types applied to 1D built-in presets work by propagating through a group's list of fixtures. The effects will propagate across the group's list order end-to-end, so it is important to pay close attention to the fixture-listing-order within each group to ensure the skew transition applies as desired.

Skew Type

Using skews with 1D built-in presets can create "multi-part" fades so that fixtures and elements fly into the scene at different speeds. The skew type and fade time can allow fixtures to come on from as slow as one by

one, all the way to a smooth fade in across them all.

None

The default skew type is None which causes all the fixtures/elements to fade in together over the course of the fade time. Each subsequent skew type will bring each fixture within a group online at an increasing speed, see below for more details:

Snaps

As the name suggests each fixture will snap from 0% (off) to 100% (on) instantly in turn, the skew propagates down the list of fixtures one-by-one. When there are fewer fixtures within a group this skew type is very effective at bringing online each fixture one-by-one. The fewer the fixtures, and the longer the fade time (and duration of the built-in preset on the timeline), the more dramatic the effect will be.

Individually

This skew type gradually fades each fixture from off to on individually. The effect propagates down the fixture group each fixture in turn. Similarly to snaps this skew works effectively with fewer fixtures that are to be brought online one-by-one but with an added fade-in effect.

Note: the Snaps and Individually skew types complete on a single fixture before starting on the next. If there are many fixtures over a short fade time the two effects will be indistinguishable as the effect has to propagate too quickly to reveal the fade effect of the Individually skew type.

Wave, Staggered, Spread, and Multispeed

These 4 Skew types propagate through a list of grouped fixtures at varying rates thereby producing different fade-in effect across the affected fixtures. The visual difference between each Skew relates to how many of the fixtures in the group are actively fading-in at the same time as well as at what rate.

The effects of each Skew are best demonstrated by the below graphic showing the 4 Skews applied to a 10s fade-in effect.

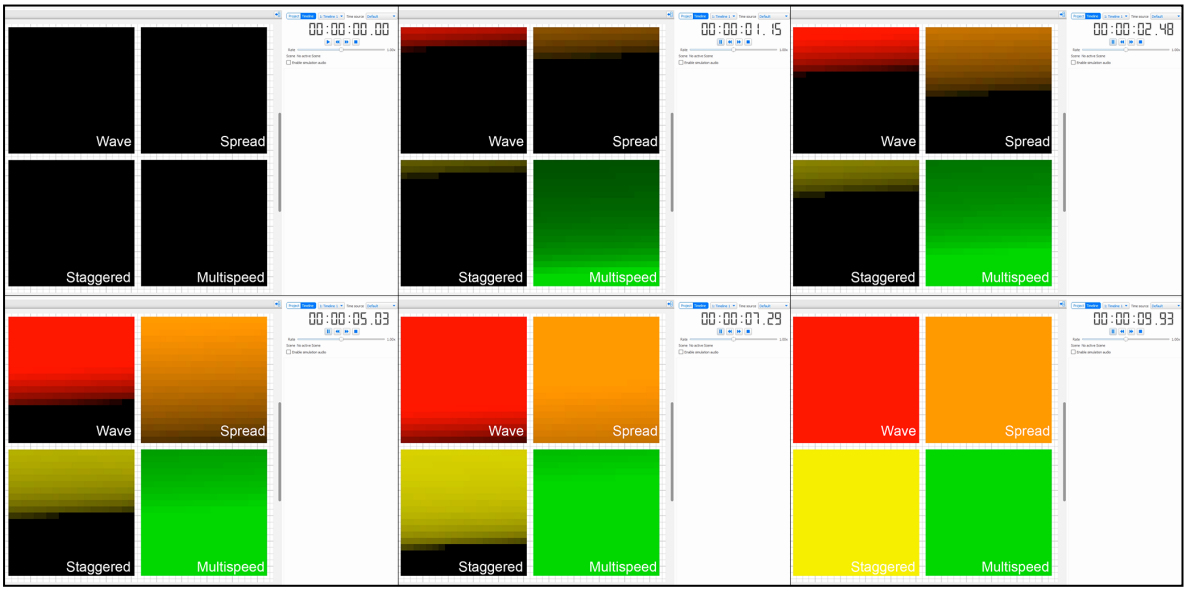


Figure: Wave, Spread, Staggered, and Multispeed skew comparisons over a 10 second fade time.

Below is a more technical overview for each Skew covering what is happening to each fixture's fade within the affected group.

NOTE: With these transitions we highly recommend experimenting using Simulate as the results can often be surprising. The Skew feature is also greatly enhanced when paired with the Direction.

For **Wave**, the fade-in time for all fixtures within the affected group is equal to a quarter of the set total fade time. This will typically produce an effect similar to a wipe transition with a narrow band of fixtures that change at the same time.

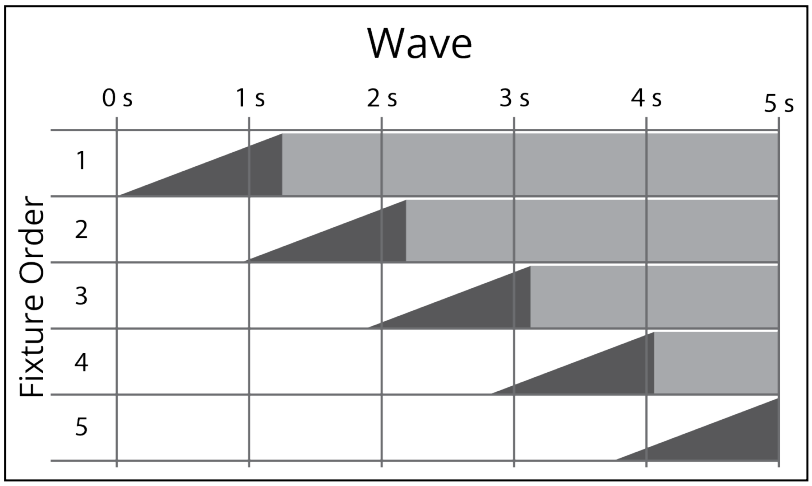


Figure: Wave skew applied to 5 fixtures over a 5 second fade time. Notice that the last fixture will begin its fade, which lasts a quarter of 5s (1.25s), so as to finish by the end of the 5s fade time. All other fixtures distribute the start time of their fade based on the beginning times for the first and last fixture.

For **Spread**, the fade-in time for all fixtures within the affect group is equal to half the set total fade time. This, like wave, also produces an effect similar to a wipe transition, but with a broader band of fixtures that are changing at the same time (compared to Wave).

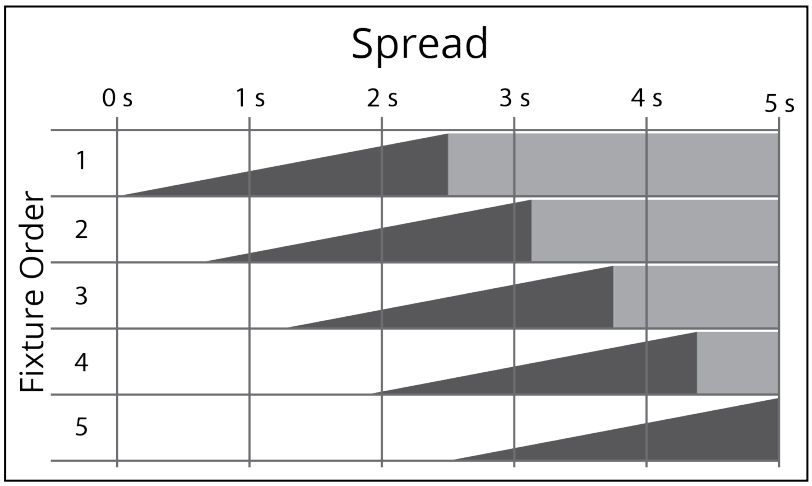


Figure: Spread skew applied to 5 fixtures over a 5 second fade time. Notice that the last fixture will begin its fade, which lasts half of 5s (2.5s), so as to finish by the end of the 5s fade time. All other fixtures, like for Wave, distribute the start time of their fade based on the beginning times for the first and last fixture.

For **Staggered**, each fixture will start fading in turn - but they will have different fade times so that they all finish fading together. This will produce an effect that will start as a more gradual transition that appears to accelerate towards the end.

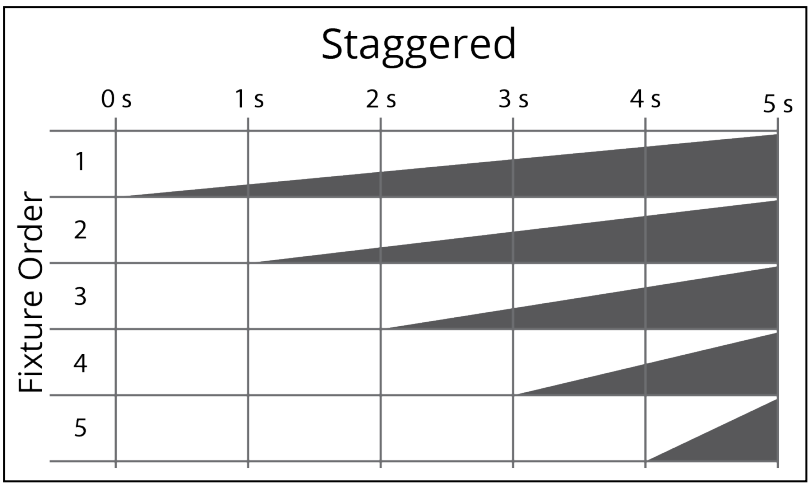


Figure: Staggered skew applied to 5 fixtures over a 5 second fade time. Notice that the fade start time for all fixtures is distributed evenly across the total fade, so in this case each fixture begins to fade 1s apart from one another.

For **Multispeed**, all the fixtures will start fading at the same time, but fixtures at the top of the group will complete the fade more slowly than the fixtures at the end of the group (see figure below). So it will start as a rapid transition that appears to slow and linger towards the end.

NOTE: Multispeed has an opposite direction to Wave, Spread, and Staggered.

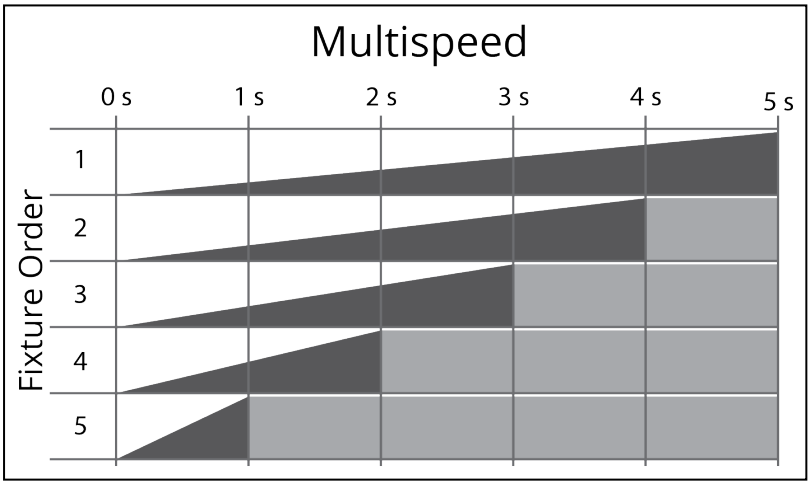


Figure: Multispeed skew applied to 5 fixtures over a 5 second fade time. Notice that the fade end time is distributed evenly across the total fade time, but for this skew the last fixtures in the group will be the first to complete their fade.

Direction

The ordering of a skewed transition depends on the fixture/element ordering within the group. The Skew Direction drop-down provides further ordering options such as Forwards and Backwards for additional flexibility. Also included are In and Out options that split the group into two halves in opposing order; In will begin at both ends of the fixture group and work the skew inwards to the middle, while Out will do the opposite, starting in the middle of the group working out to the ends. There is also a Shuffle option that applies the skew following a random order.

Additional groups can be created with different fixture/element ordering to achieve other skewed effects.

Repeat

Specifies the number of adjacent fixtures/elements over which a skewed transition is repeated, default is All meaning that the skew will span the entire selection. Typically you set this value to be equal to the number of elements in a compound fixture or the number of fixtures in a zone or on a truss, experimentation is recommended as interesting effects can be achieved.

Buddy

Specifies the number of fixtures/elements that will fade together within a skewed transition, default is 1 meaning that each fixture/element will fade independently. Set to 2 to make pairs fade together, 3 for threesomes etc. Again, experimentation recommended.

Matrix & Media presets (2D)

Skew types applied to 2D built-in presets work off of the mapping array for pixel matricies and it thus takes into consideration the location of each fixture to produce the effects.

Skew Type

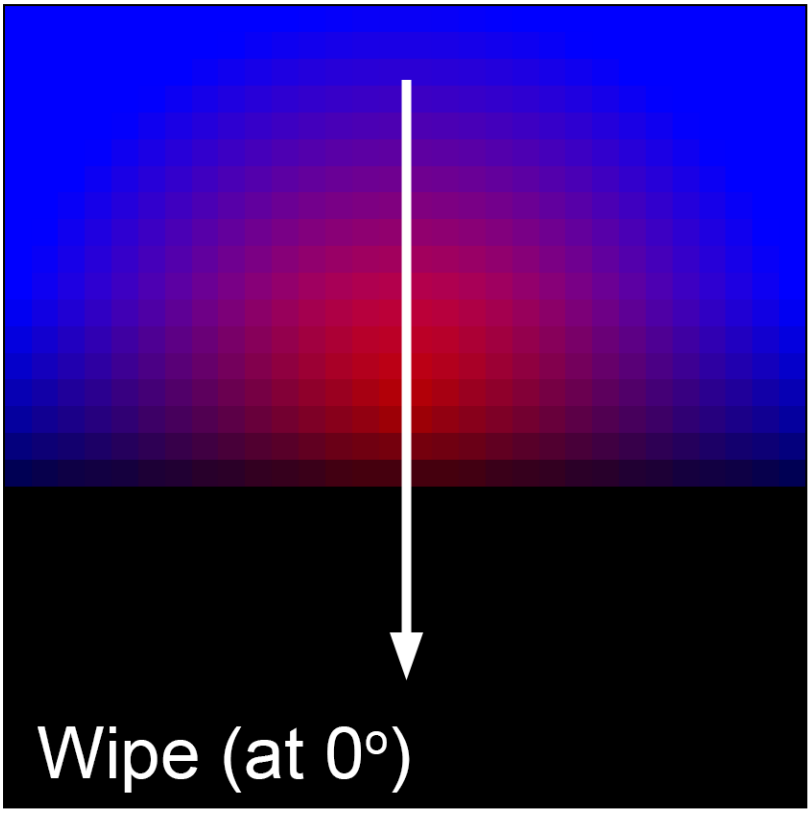
Since the Pixel Matrices onto which you place Matrix and Media presets have spatial awareness, the available skews are more powerful than those for 1D built-in presets. They are akin to video wipes in their functionality. Combined with the % Fade (which describes the feathering of the transition edge) and Angle properties it is possible to obtain a variety of visual effects.

None

The default skew type is None which causes all the fixtures/elements to fade in together over the course of the fade time.

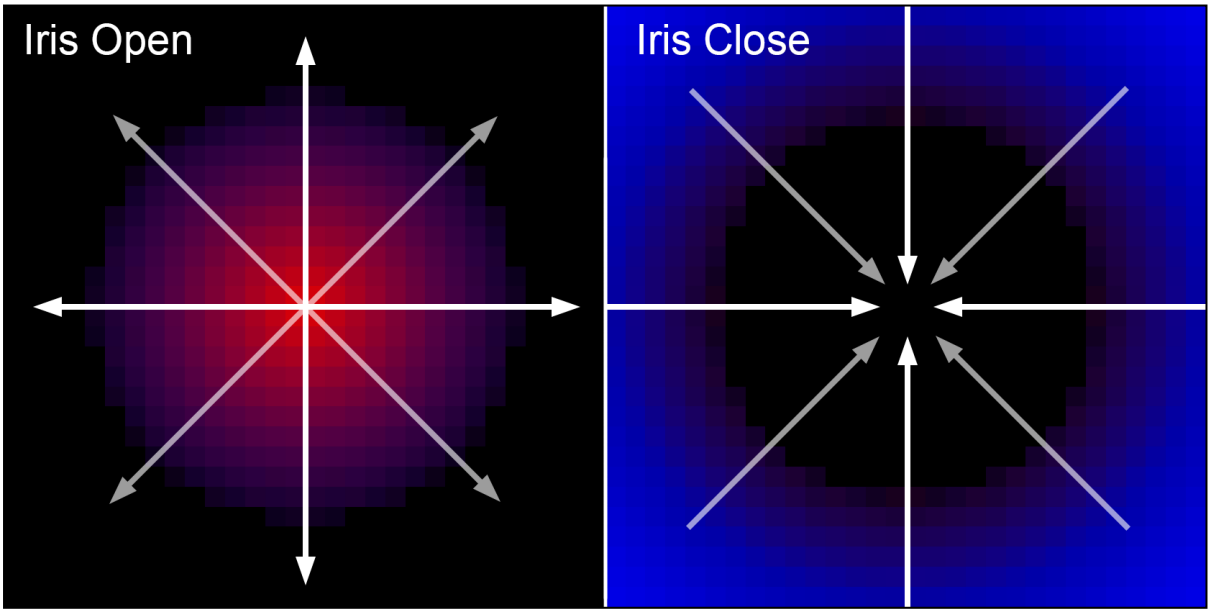
Wipe

This is a wipe that fades in row by row from one side of the canvas to the opposite.



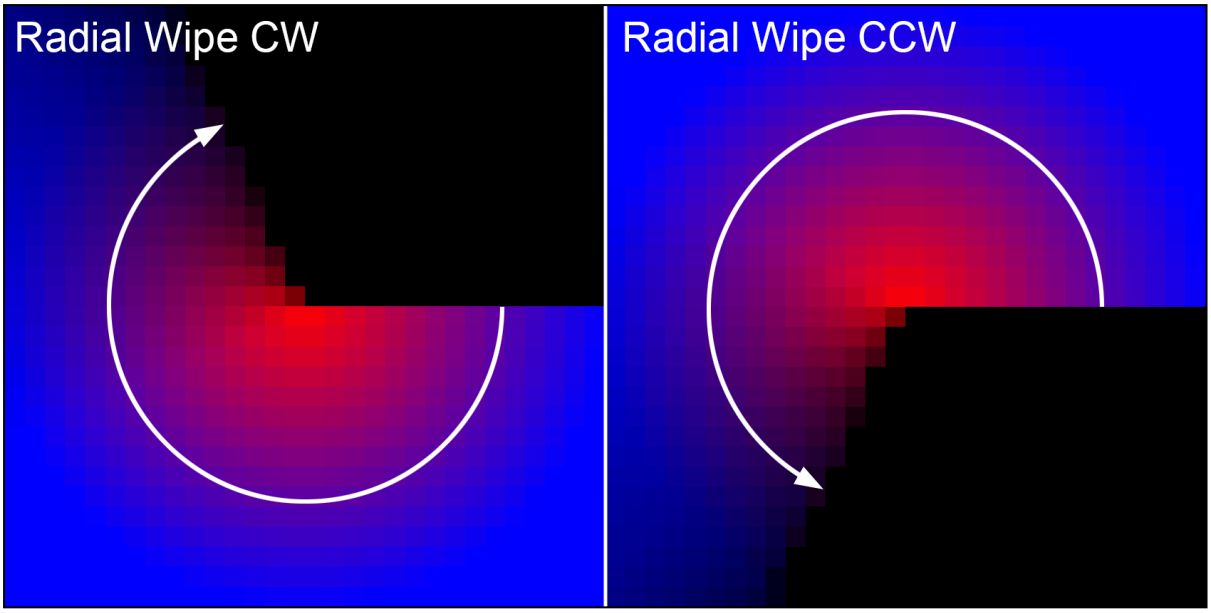
Iris Open & Close

This skew type will create a radiating fade that will start from the center of the canvas and propagate outwards (Iris Open), or the opposite starting at the edges and propagating inwards to the center (Iris Close). Angle doesn't have an effect on this skew type as its effect is perpendicular to the angel.



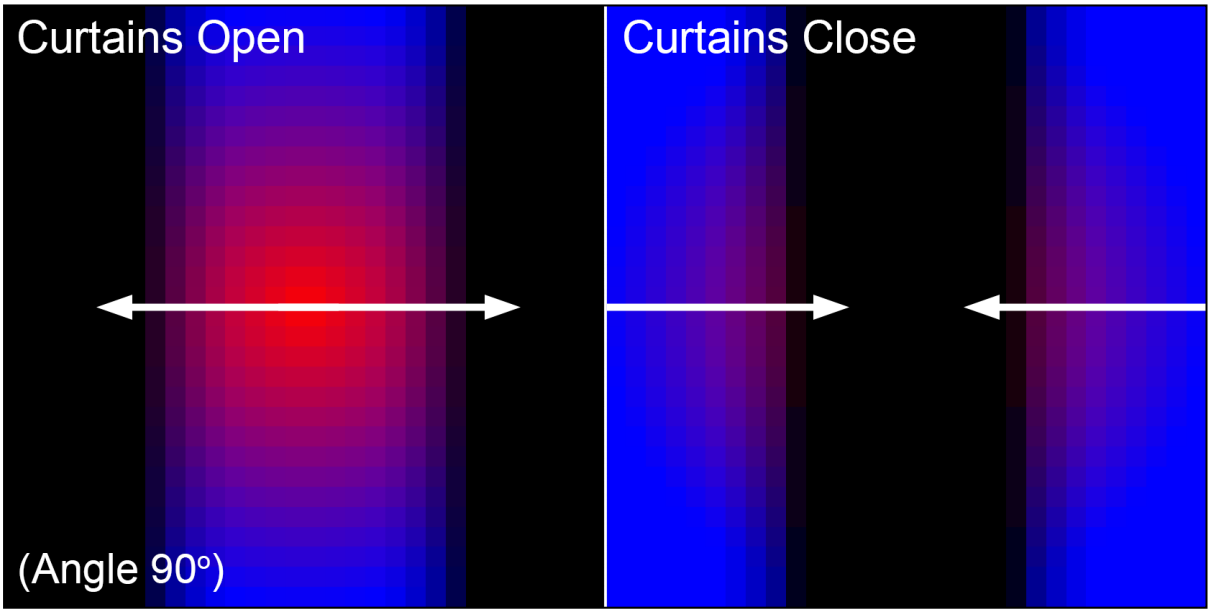
Radial Wipe Clockwise (CW) & Counter Clockwise (CCW)

The radial wipe creates a fade wipe that propagates following a clock hand as it revolves around the center-point either clockwise, or counter clockwise. By changing the angle the point at which the wipe starts can be modified. The wipe starts at the specified angle.



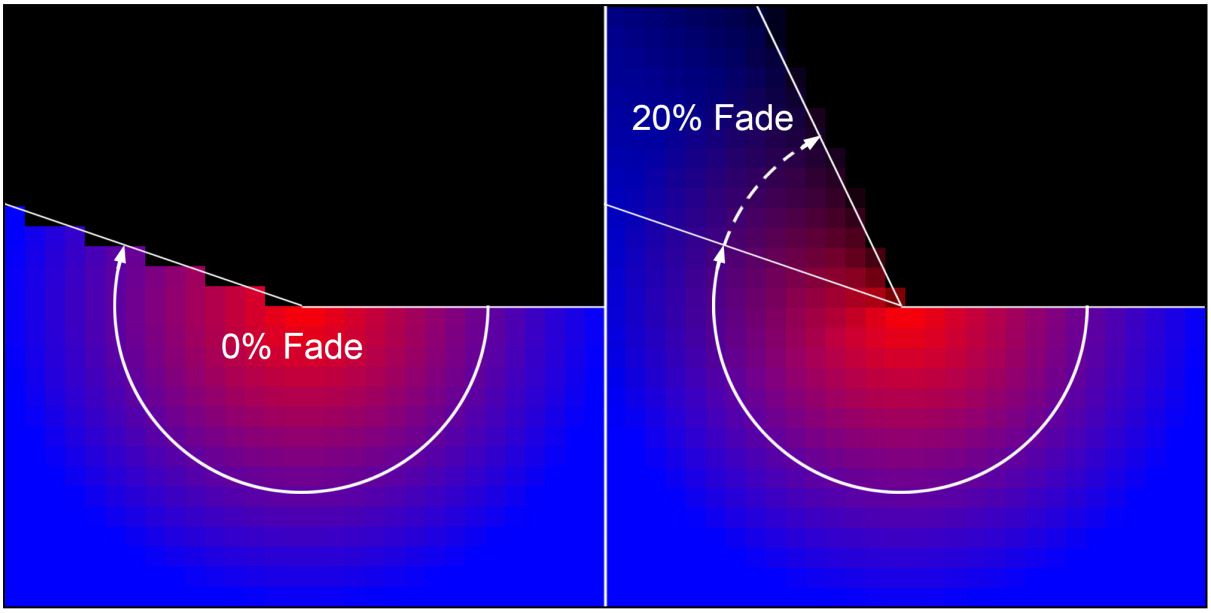
Curtains Open & Close

This skew type will create a fade-in revealing the content as if curtains were being opened or closed. Curtains open skew creates a fade-in begin in the middle and propagate outwards from the center, whereas the curtains close skew creates a fade-in that begins at the edges and propagates inwards. The direction of the propagation is perpendicular to the angle, so at angle = 0 the effect comes from the top and bottom.



% Fade

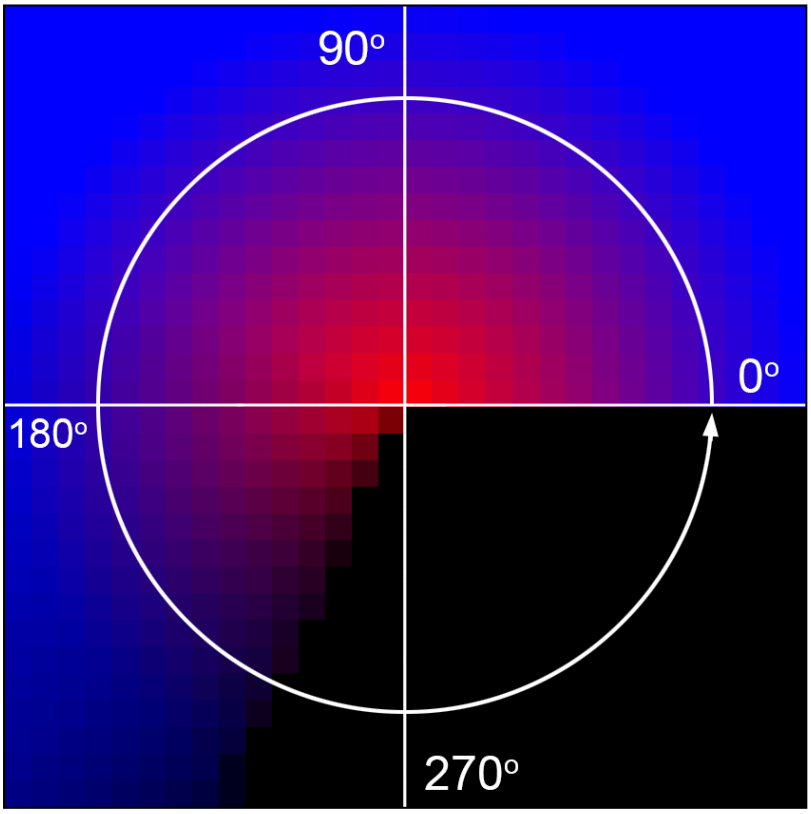
Sets the hardness of the of the transition edge for 2D skew types - akin to feathering is some image editing software. The hardness ranges from hard (0%) to soft (100%). When the % Fade is very hard the rows of fixtures will turn on one after the other in a way that creates a harsh edge. With a softer (higher than 0%) % Fade the transition smooths across more rows making it softer. See below for a representation of this effect.



NOTE: If % Fade is put to 100% (i.e. softest) the skew effect will not take place as it is the same has having it set to none.

Angle

Some skews (for example Linear or Radial Wipe) can optionally accept an angle value that alters the direction or start point of the transition.



Specifying times

Timing fields display times in the format hh:mm:ss.cc (hours:minutes:seconds.centiseconds), although leading zeros are not displayed. 24 hours (24:00:00.00) is the maximum timeline length and thus timing duration.

When setting times you can enter in this format directly (omitting leading zeros) or you can use h, m & s to specify your units and Designer will reformat accordingly. Furthermore, any number input without separators (h, m, s or :) is taken literally if it is valid as such or converted if not, here a decimal point will always denote centiseconds.

For example:

00:01:30.00	1 minute and 30 seconds	(00:01:30.00)
1:30	1 minute and 30 seconds	(00:01:30.00)
90s	1 minute and 30 seconds	(00:01:30.00)
1h2.5m	1 hour, 2 minutes and 30 seconds	(01:02:30.00)
2h7m45.5s	2 hours, 7 minutes, 45 seconds and 50 centiseconds (half a second)	(02:07:45.50)
99	1 minute, 39 seconds ("99" not valid so converted)	(00:01:39.00)
100	1 minute,0 seconds ("100" valid so taken literally)	(00:01:00.00)
2020	20 minutes, 20 second	(00:20:20.00)
30.1	30 seconds and 10 centiseconds (tenth of a second)	(00:00:30.10)

Precedent

The Pharos Controllers use the Latest Takes Precedent Plus (LTP+) system (popularised by Flying Pig Systems in the early 1990s) to determine what to output to a fixture (or, strictly speaking, fixture element or parameter) at playback runtime. LTP+ was an enhancement of the standard LTP system and was designed to incorporate automated lighting control. The "rules" of the LTP+ system are as follows:

1. After system initialisation, and prior to any preset (on a timeline) running, the output will be in a default, "released" state. This does not mean that all DMX channels will be zero however as this default state is determined by each fixture's library definition that will set parameters to sensible "home" positions, for example pan and tilt to midway, irises and gobo/colour flags to open white.
2. The output will respond to the latest preset activated regardless of the preset data (so a preset programmed to black will override a colour).
3. When a preset expires or is explicitly released the output will revert to the prior state which may be an overridden preset (if any) or the default state.
4. Fixture parameters are grouped by kind (Intensity, Colour, Beam Image, Beam Shape, Position & Control) and so preset programming and thus precedent operates at this level - multiple presets can thus be responsible for the output of a fixture with multiple parameter kinds e.g. a moving light or conventional fixture with a scroller.

Since the Controllers can run multiple timelines simultaneously then some consideration should be given as to how best structure the project. This is particularly important if the project calls for random triggering of timelines, for example from a Button Panel Station (BPS), since there is no way of knowing in advance what state the output will be in (i.e. which timelines have already been triggered) when a new timeline triggered. This interaction of timelines can yield unexpected results unless care is taken when programming.

Timeline Audio

It is possible for the LPC X, VLC, VLC+ to output Audio from the Stereo connectors on the rear.

This audio will then be played synchronously with the lighting on the timeline.

Managing Timeline Audio

to Enable Timeline Audio

Timeline Audio is a feature that must be enabled from the Project Features page, and requires a suitable controller in the project.

To Import Audio

Audio is imported in the same way as video clips; from Mapping or the Media Preset library.

To mute timeline audio during simulation

While simulating a timeline, it may be desirable to mute the audio temporarily. This can be done by selecting the  Mute button on the Mode toolbar.

To Set the Default volume of the Timeline Audio

The volume of the Timeline Audio output is managed at a controller level.

To set the Default level, go to the controller's interface settings. This can be adjusted at runtime using the Set Volume Action.

Imported audio should be normalised to avoid having to chance volume for different timelines.

Timeline Audio Properties

When an Audio Media preset is added to a timeline, the following properties will be available:

- Start
- End
- Audio category

Start

This is the start point of the audio, this can be used to not play the start of the audio track

End

This is the end point of the audio, this can be used to not play the end of the audio track

Audio category

There are two categories available, which treat the audio in different ways:

Background

A background audio track will be played back normally, and if another background track is started, the first will be stopped.

Alert

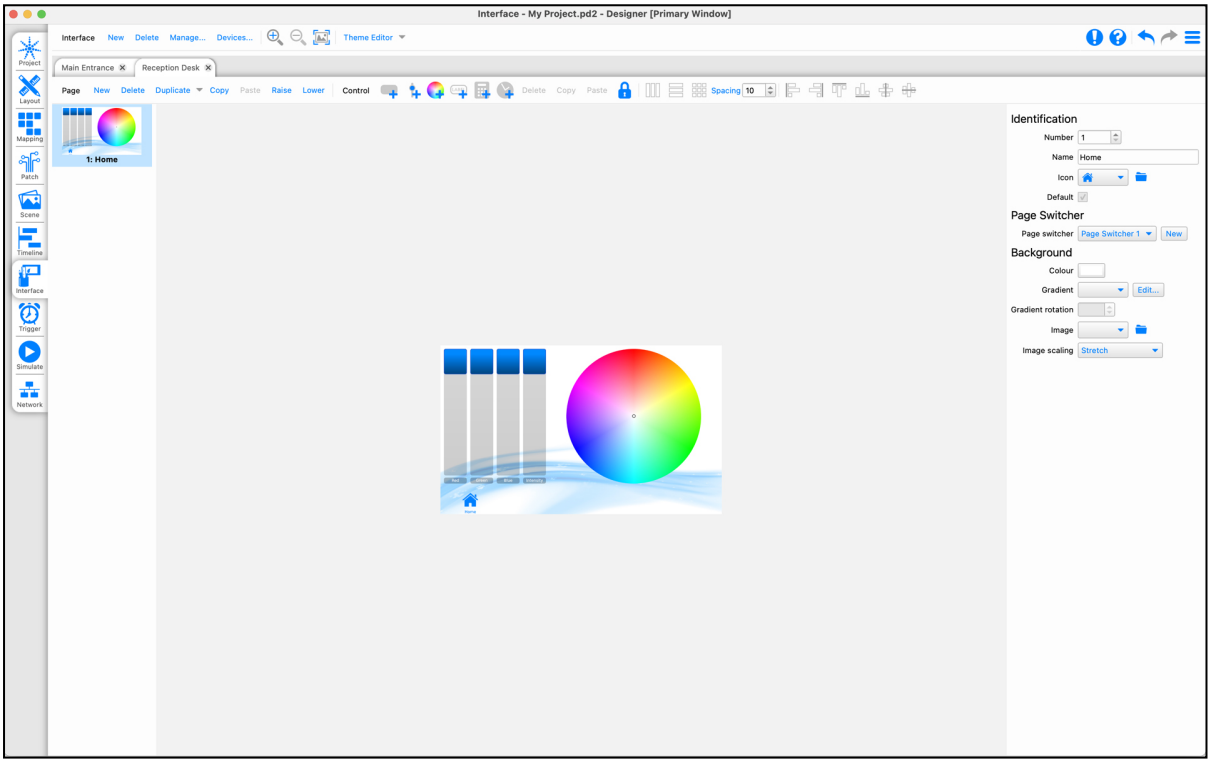
An Alert track will playback over the top of a Background track, and will stop a playing alert track if another alert track is started.

Interface Overview

Keyboard Shortcuts

Ctrl+N	Create a new Interface
Ctrl+I	Show interface properties
Alt+ *select Colour Picker*	Sets the startup colour of the colour picker to the selected colour
Ctrl+drag *on one or more control*	Creates a duplicate of the selected control/s

The Interface View is used to create custom User Interfaces for a Pharos Touch Panel Device, including the TPC, TPS, TPS 5 and TPS 8.



Working with Interfaces

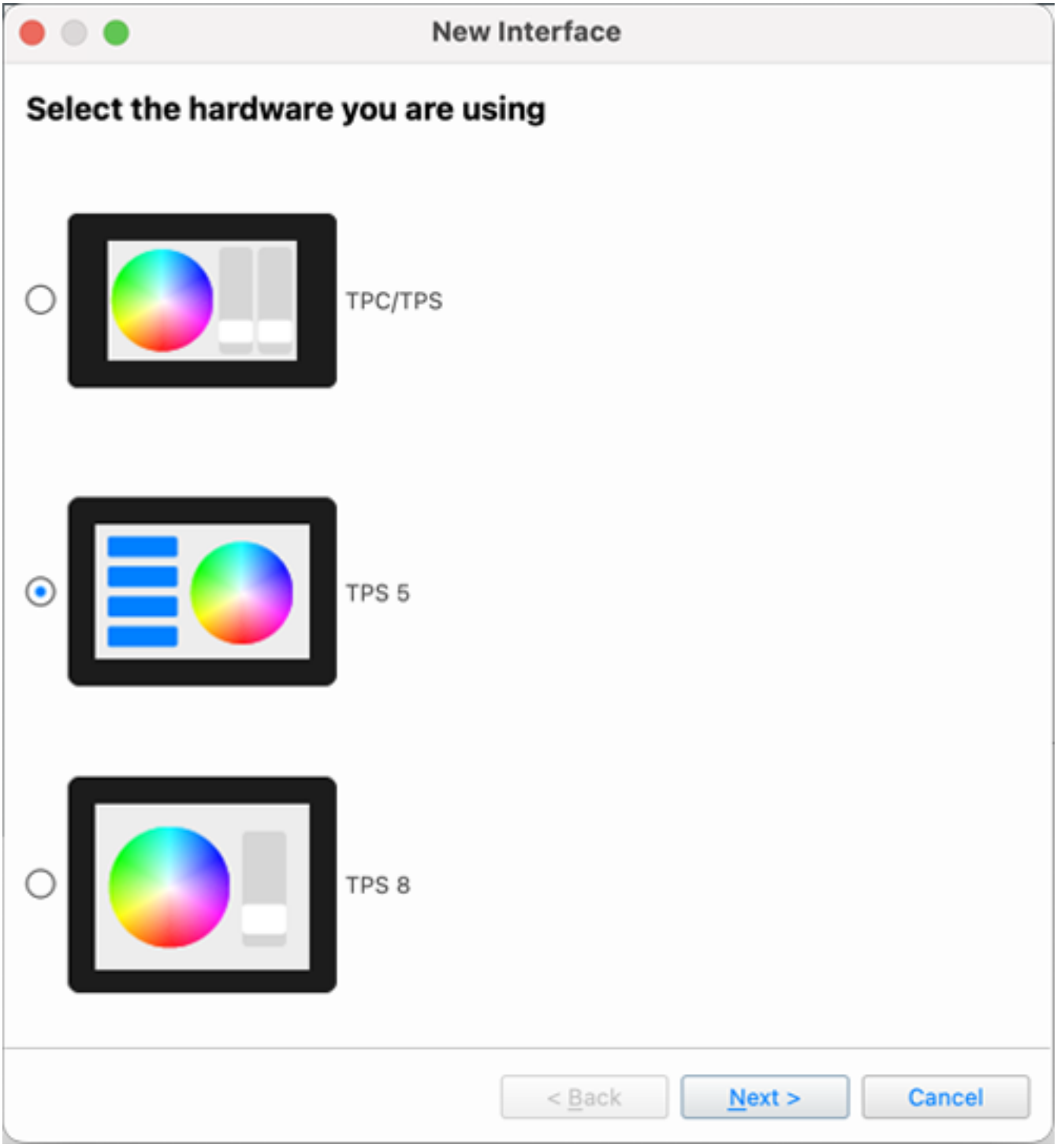
An interface is a group of pages which can be displayed on a TPC, TPS, TPS 5 or TPS 8.

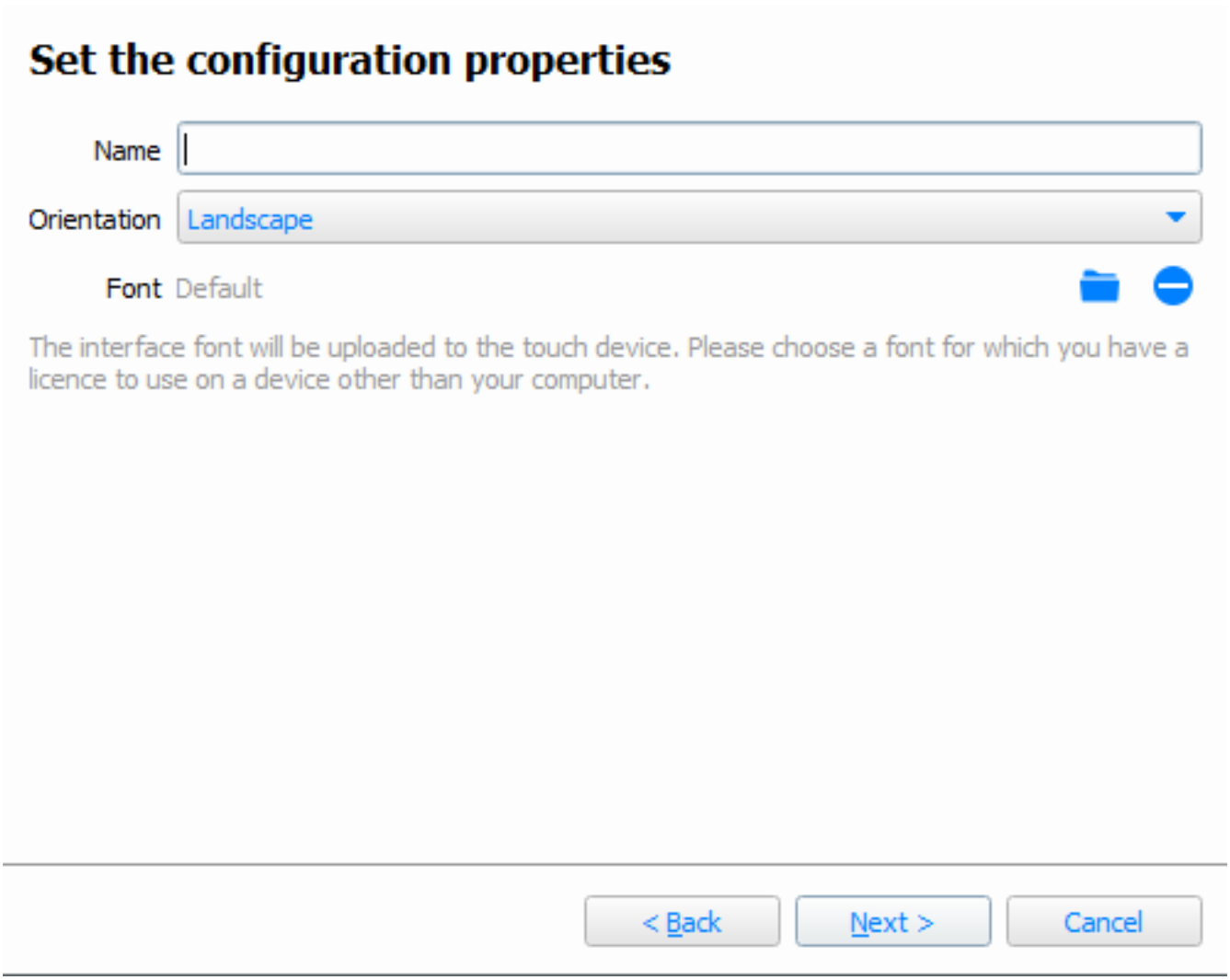
Managing Interfaces

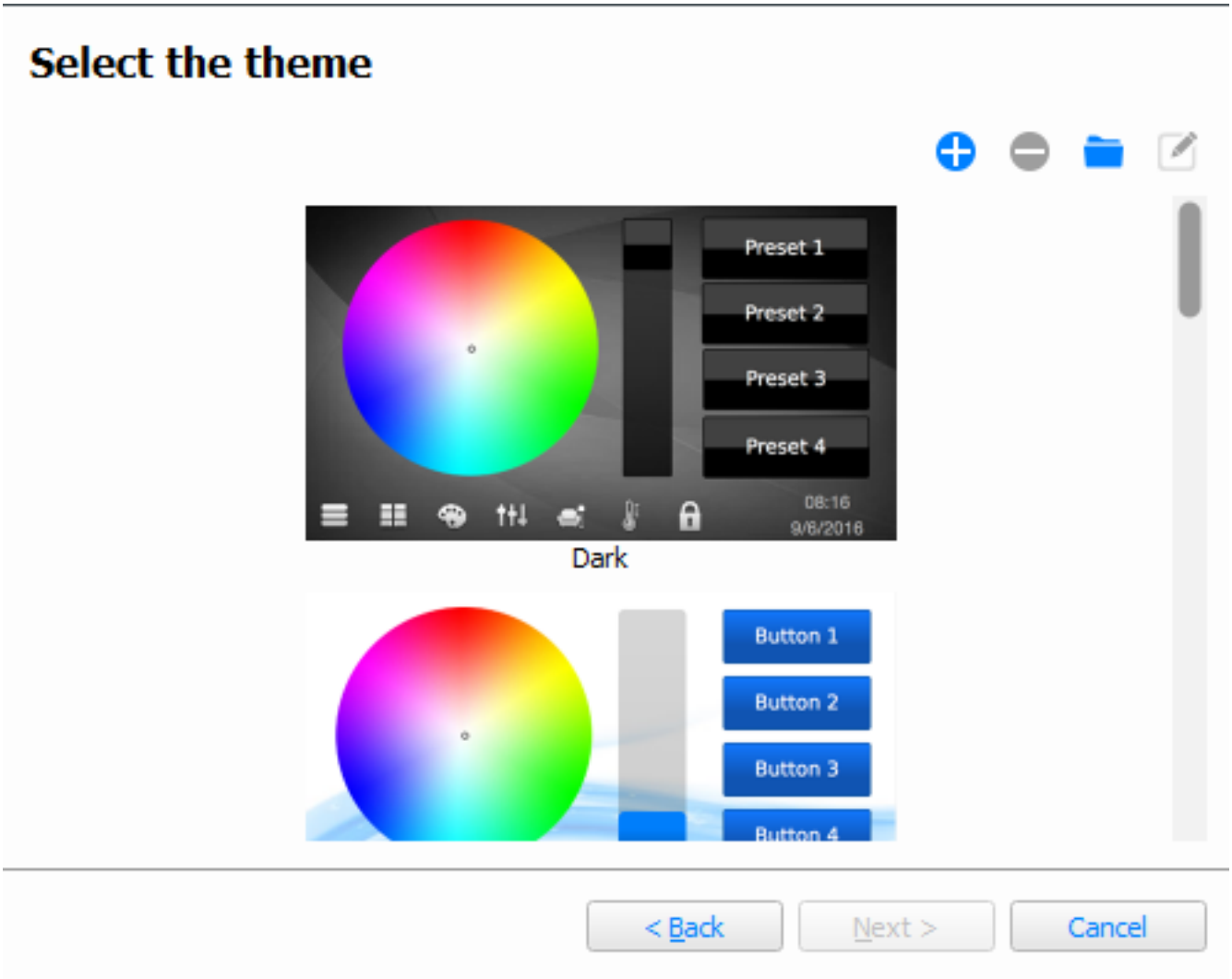
Managing of Interfaces is handled from the Interface toolbar

New Interface

The New button allows you to create a new interface for your project. This will open a tool to configure the interface properties:


New Interface
Select the hardware you are using
TPC/TPS
TPS 5
TPS 8
< Back
Next >
Cancel


Set the configuration properties
Name
Orientation
Landscape
Font
Default
The interface font will be uploaded to the touch device. Please choose a font for which you have a licence to use on a device other than your computer.
< Back
Next >
Cancel



Name

A user readable name for the interface

Orientation

The format in which to create the interface (Landscape or portrait)

Font

The font used for the Interface can be set here. You will need to locate a font file (*.ttf, *.ttc,*.otf, *.otc) on your system. This font will be used for all text in your Interface.

Theme

The theme for the interface. See Built in Themes.

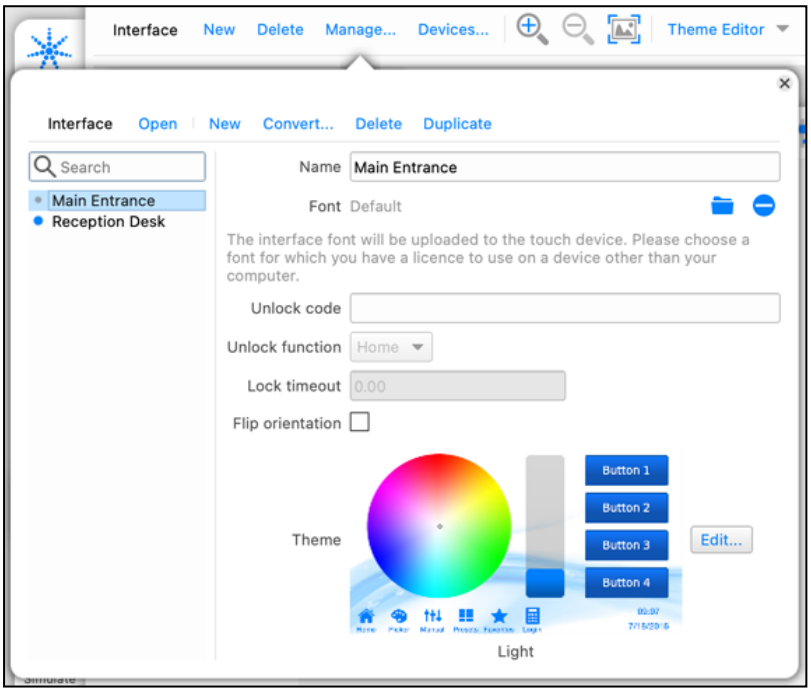
NOTE: Once the interface has been configured, the tool will prompt you to create the first page. See here for more details.

Delete Interface

You can delete an Interface either through the Delete button on the toolbar for the active interface or through the Manage dialog for inactive interfaces

Manage Interfaces

The Manage option allows you to Open interfaces which have previously been closed, along with deleting and accessing the properties for inactive interfaces.



NOTE: Use the Convert option to open the wizard to change the hardware type for an Interface.

Interface Properties

The Properties tool allows you to rename the interface at any time.

You can also setup the Lock functionality within the Interface.

Lock

When a Touch Device hasn't been touched for a period of time, it can be configured to go into a Lock state. This displays a keypad on the screen which allows an unlock code to be input. This will then return the Touch Device to the normal interface.

Unlock Code

The numerical code which needs to be input to unlock the Touch Device.

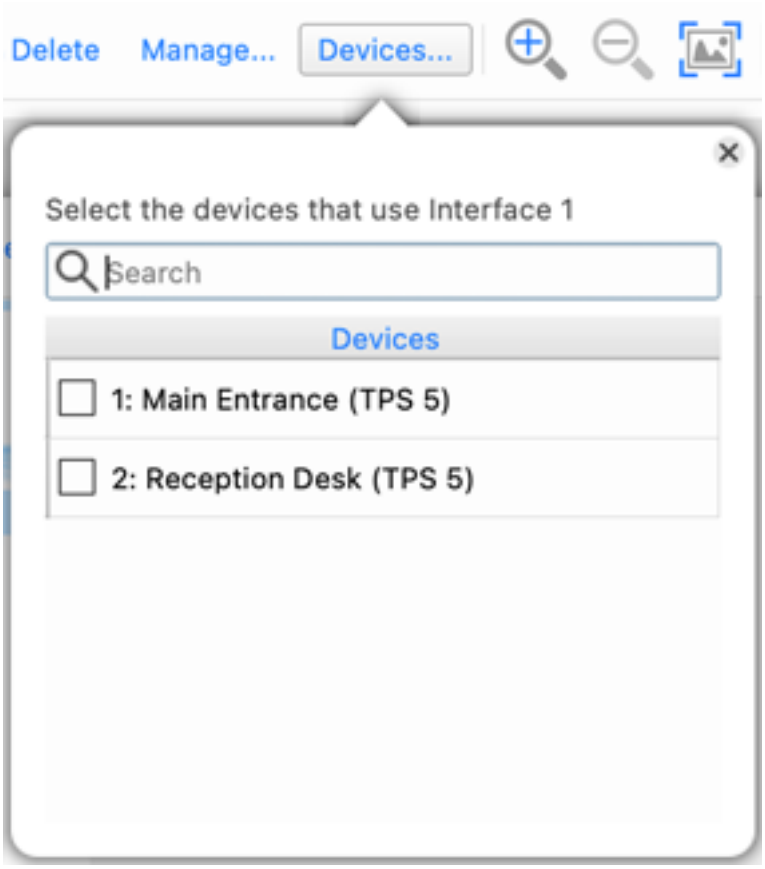
Unlock Function

Define whether the interface should return to the Home page or the page that was displayed when it locked.

Lock Timeout

Define the period of Inactivity before the Touch Device becomes locked.

Linking Interfaces to Touch Devices



To link an Interface to a TPC, TPS, TPS 5 or TPS 8 use the Devices... option in the toolbar.

This will then list all touch devices in the project and allow you to check all the touch devices which should display the active interface.

Working with Pages

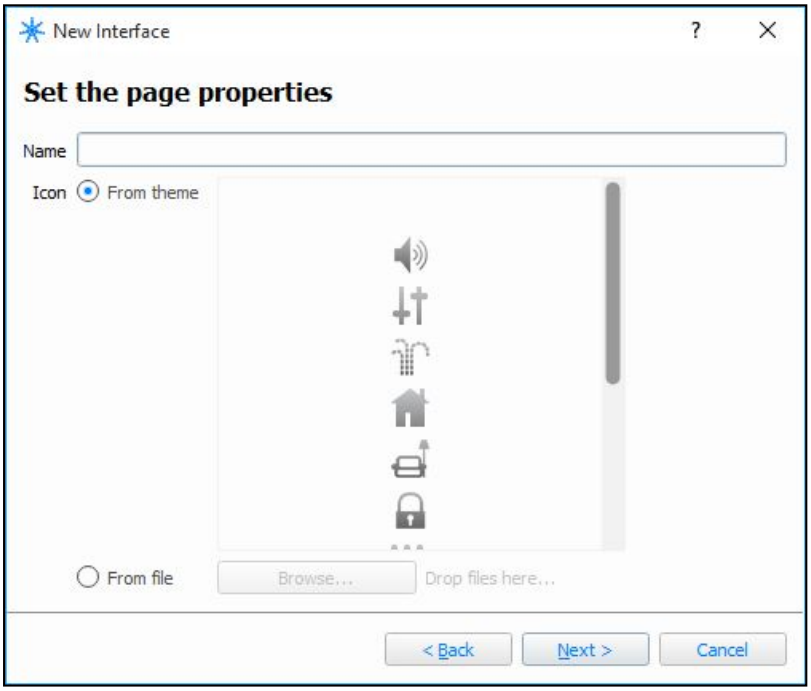
Pages are the core of a Touch Device Interface; they contain all the elements which can be used to control your project.

Managing Pages

New Page

When you create a new Interface, you must create the first page of the interface. Once the interface and first page have been created, you can create additional pages with the New button on the Page toolbar.

These will both bring up a tool to create the page:



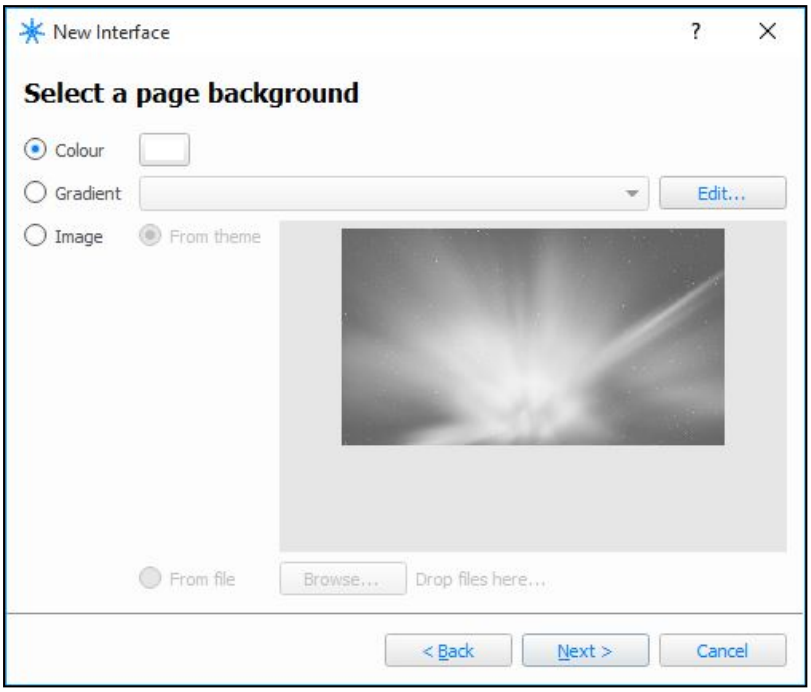
Name

A user readable name for the page

Icon

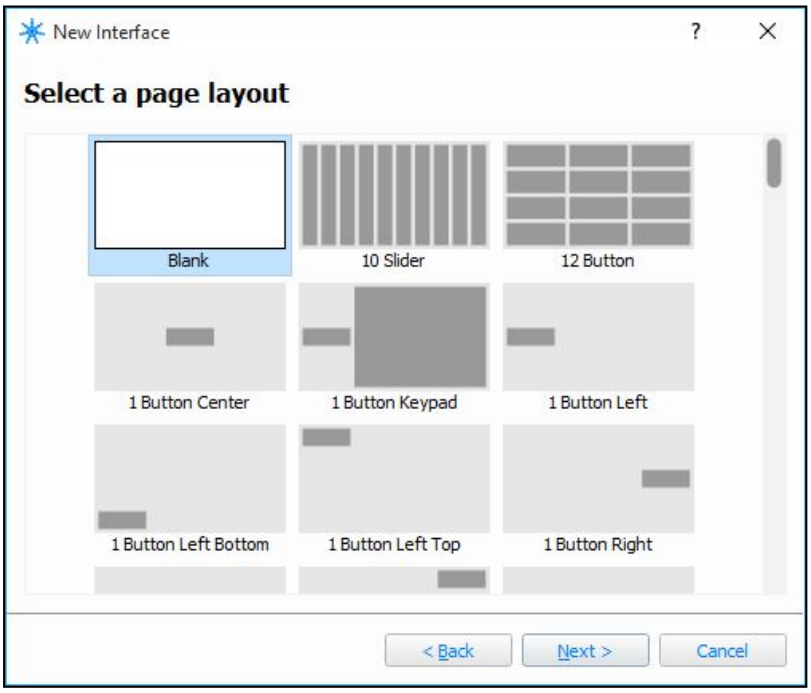
Each page can have an icon attributed to it. This is shown on Page Switchers. The icons offered will be from the chosen theme, but you may click the Browse button to choose your own.

Page Background



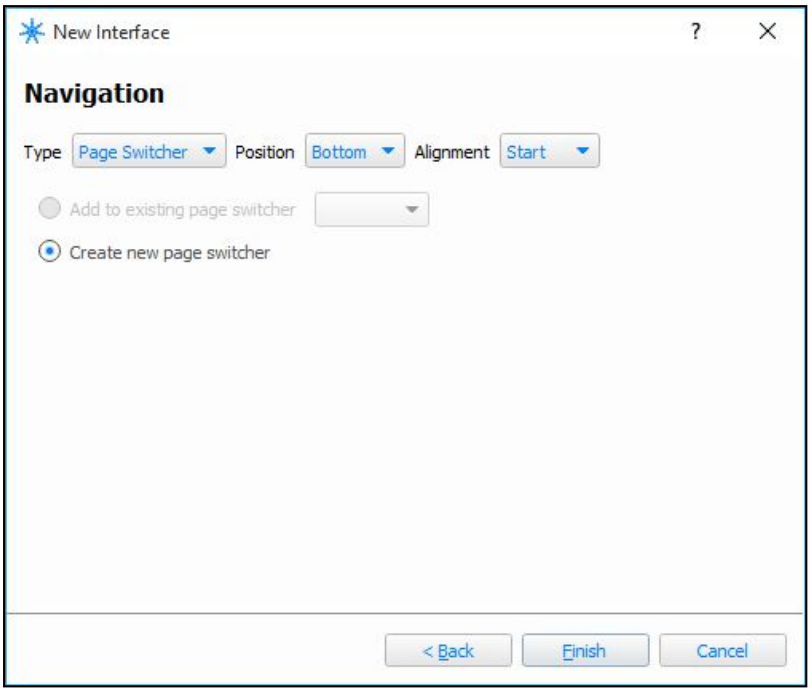
Set the page background, using either a colour, gradient or image. Some gradients are included with the application, but a gradient editor is provided for you to create your own. The images offered will be from the chosen theme, but you may click the Browse button to choose your own.

Page Layout



Pharos Designer comes preloaded with some layouts of controls. These will pre-populate your page with a set of controls. Different page layouts are presented depending on interface orientation and screen size.

Navigation



Select a navigation type for moving between pages, choosing from a page switcher or navigation buttons.

Delete Page

You can delete the active page in the editor using the Delete button on the Page toolbar.

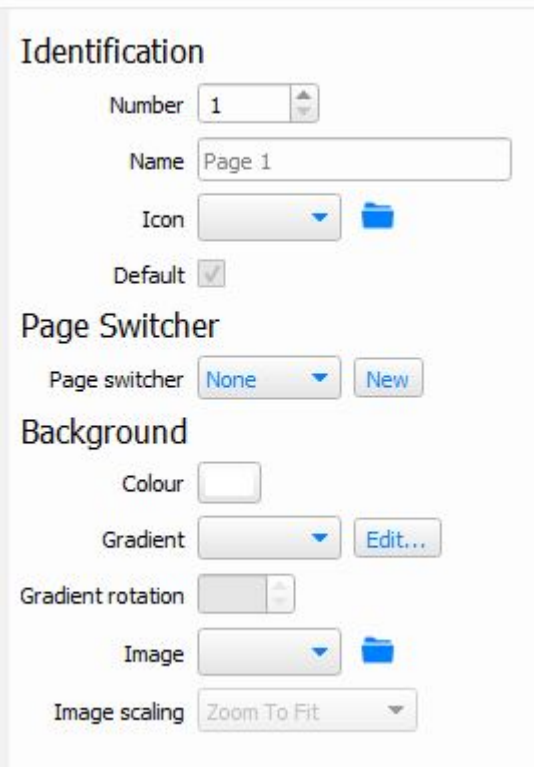
Duplicate Page

You can duplicate the active page in the editor using the Duplicate button on the Page toolbar.

By default, this will create a new set of controls on the page. If you want to copy the page and keep the control keys the same, then there is a dropdown beside the Duplicate button which allows you to Duplicate (Keep Control Keys)

Page Properties

When a page is selected, the properties browser displays the properties of the page:



Number: A unique number used to reference the pages within an interface.

Name: A user readable name to enable easy identification of the page in the page browser

Icon: The icon displayed in a page switcher

Default: The default page that should be shown when the interface loads.

Page Switcher: Define and edit the page switcher that should be used on this page.

Background Colour: A single flat colour can be chosen with the colour picker

Background Gradient: A user configurable gradient can be applied as the background, and the rotation angle of the gradient can be specified.

Background Image: A static image can be used as the background, with image scaling options to stretch, zoom or tile the image on the background.

Working with Controls

Adding Controls

To add new control items to a page, simply select which control you would like to add then drag and release on the page where you would like the control to be. You can add buttons, sliders, colour pickers, labels, keypads and clocks.

Editing Controls

To edit controls that are already on the page, click  and then select the controls you would like to edit. You can now edit the controls by using your mouse to move and resize the controls or move the controls by pressing the arrow keys on your keyboard.

Duplicating Controls

It is possible to create duplicates of controls by copying and pasting, or using Ctrl+drag(Cmd + drag) to create a duplicate of the selected control/s.

Deleting Controls

You can delete controls by selecting the controls and then clicking Delete.

Editing Layout of Controls

With multiple controls selected you can use a variety of tools to alter their layout:

Icon:	Layout control:	Effect:
	Layout selected controls horizontally	Moves and resizes the selected controls to fill the selection box with spacing between controls dictated by the spacing value. Controls will be laid out horizontally.
	Layout selected controls vertically	Moves and resizes the selected controls to fill the selection box with spacing between controls dictated by the spacing value. Controls will be laid out vertically.
	Layout selected controls in a grid	Moves and resizes the selected controls to fill the selection box with spacing between controls dictated by the spacing value. Controls will be laid out in a grid. This grid layout supports controls that span multiple rows or columns.
	Align selected controls to the left	Moves the controls to the left-most point of the selection box. Does not effect the Y axis or control size.
	Align selected controls to the right	Moves the controls to the right-most point of the selection box. Does not effect the Y axis or control size.

	Align selected controls to the top	Moves the controls to the top of the selection box. Does not effect the X axis or control size.
	Align selected controls to the bottom	Moves the controls to the bottom of the selection box. Does not effect the X axis or control size.
	Align selected controls to the middle in a vertical line	Moves the controls to the centre of the selection box in a vertical line. Does not effect the Y axis or control size.
	Align selected controls to the middle in a horizontal line	Moves the controls to the centre of the selection box in a horizontal line. Does not effect the X axis or control size.

NOTE: The separation of the controls when using the layout options is defined by the Spacing option

Editing Control's Properties

The properties for a control can be edited using the Property Editor, when one or more controls are selected. If multiple controls are selected, only common properties will be displayed and any changes will be applied to all selected controls.

Common Properties

Caption - the text that appears on a button, defining its purpose. It is possible to include a line break in this caption using "\n". If you require "\n" in your caption, you should use "\\n" to escape the first backslash. The caption of a control can be changed via the Set TPC Control Caption trigger action in Trigger - see triggers for more information.

Key - the reference for the control within Trigger. By default this will be set to <control type>XXX, where <control type> is 'button' or 'slider', etc. and XXX is a unique number for the control, which starts at 001 for a new project, e.g. button123. Setting the control key to be the same for two controls will mean that they will fire the same trigger in Designer. A single TPC trigger in Designer can match multiple control keys through the use of variables. See Variables for more information on using variables with TPC triggers.

Startup State - choose which state the item should be in when the Controller starts up.

X - The position, in pixels, of the control on the horizontal axis of the screen relative to the top left corner of the control.

Y - The position, in pixels, of the control on the vertical axis of the screen relative to the top left corner of the control.

Width - The width of the control in pixels.

Height - The height of the control in pixels.

Button Properties

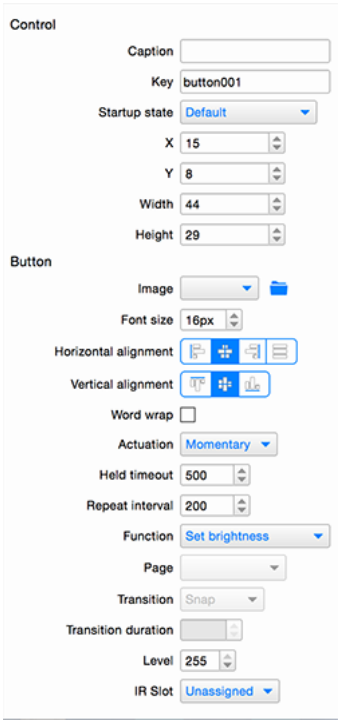


Image - choose an image to display instead of the themed shape of the button.

Either choose from button images already used in the project or click ▫▫▫ to browse for a new image. Images will be stretched to fill the area of the button. Transparency in images is supported. Overall transparency of the button will still be determined by the current theme. Click ↩ to remove the image and return the button to the themed shape.

Font Size - set by default from the theme; size of the font used to display the button caption.

Horizontal alignment - The horizontal alignment of the text in the button

Vertical alignment - The vertical alignment of the text in the button

Word Wrap - set by default from the theme; determines whether the caption of a button will flow onto multiple lines if necessary.

Actuation - can be set to Momentary or Maintained. Momentary indicates the button will trigger a 'press' and 'release' every time it's touched; Maintained indicates the button will remain depressed when tapped once, and will only release when tapped again.

Held Timeout, Repeat Interval - specify the length of time the button must be held before 'repeat' triggers begin firing and how rapidly 'repeat' triggers fire.

Function - can be set to: None, Next Page, Previous Page, Back, Go To Page, Increase Brightness, Decrease Brightness, Set Brightness. Each function has associated sub-properties. For example, in the screenshot below the Next Page transition can be set to None, Pan Left or Pan Right and a transition duration can be set.

Page - if function is set to Go To Page, select the page to go to

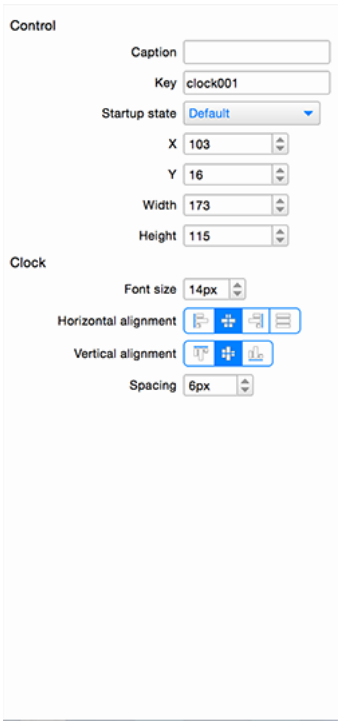
Transition - if function is set to Next Page, Previous Page, Go To Page or Back, select the required transition effect

Transition duration - if function is set to Next Page, Previous Page, Go To Page or Back, select the required transition duration time

Level - if function is set to Set brightness, set the required brightness.

IR Slot - This associates an IR slot with the button. The IR slot can be activated by an IR remote control, mimicking a button being tapped.

Clock Properties



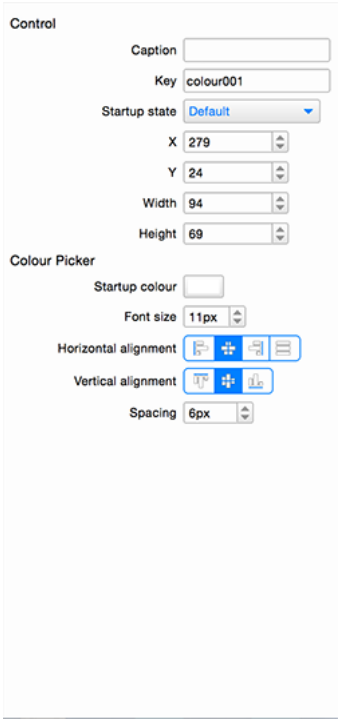
Font Size - set by default from the theme; size of the font used to display the colour picker caption.

Horizontal alignment - The horizontal alignment of the text in the caption

Vertical alignment - The vertical alignment of the text in the caption

Spacing - set by default from the theme; spacing between the colour picker wheel and the caption text.

Colour Picker



Startup Colour - select the colour that you want the colour picker to be in at startup. This can also be set by clicking on the colour picker with Alt held down.

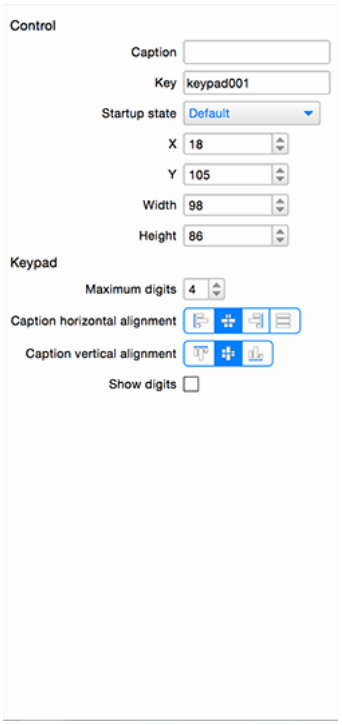
Font Size - set by default from the theme; size of the font used to display the colour picker caption.

Horizontal alignment - The horizontal alignment of the text in the caption

Vertical alignment - The vertical alignment of the text in the caption

Spacing - set by default from the theme; spacing between the colour picker wheel and the caption text.

Keypad Properties



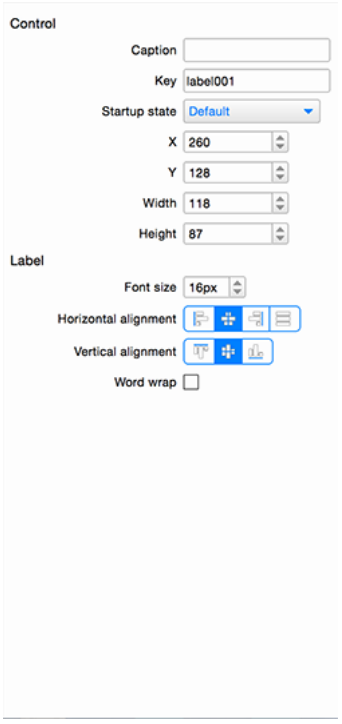
Max Digits - set the maximum amount of characters that may be entered into a keypad by the user at a time.

Caption horizontal alignment - The horizontal alignment of the text in the caption

Caption vertical alignment - The vertical alignment of the text in the caption

Show digits - choose whether the characters entered into a keypad are hidden or shown.

Label Properties



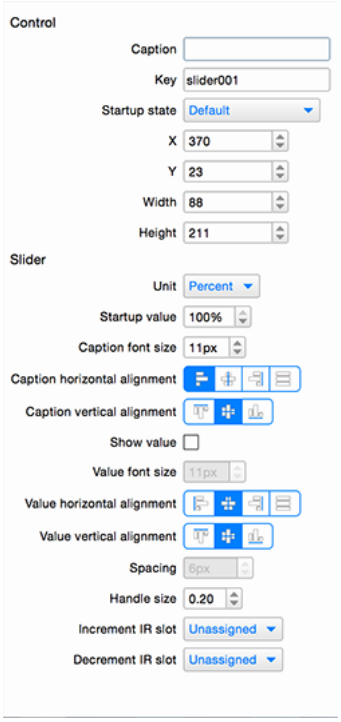
Font Size - set by default from the theme; size of the font used to display the caption text in the label.

Horizontal alignment - The horizontal alignment of the text in the label

Vertical alignment - The vertical alignment of the text in the label

Word Wrap - set by default from the theme; determines whether the caption of a label will flow onto multiple lines if necessary.

Slider Properties



Unit - this sets whether the value should be displayed as a percentage or 8-bit value (0-255).

Startup Value - this sets where the slider is positioned at startup

Caption Font Size - set by default from the theme; size of the font used to display the slider caption.

Caption horizontal alignment - The horizontal alignment of the text in the caption

Caption vertical alignment - The vertical alignment of the text in the caption

Show Value - whether the value of the slider is displayed next to it.

Value Font Size - set by default from the theme; size of the font used to display the slider value.

Value horizontal alignment - The horizontal alignment of the value

Value vertical alignment - The vertical alignment of the value

Spacing - set by default from the theme; spacing between the slider and the first line of text, and the spacing between the caption and value.

Handle Size - set by default from the theme; fraction of the slider track that is occupied by the slider handle (0.05 - 0.95).

Increment IR Slot - this allows an IR slot to be associated with incrementing the slider level.

Decrement IR Slot - this allows an IR slot to be associated with decrementing the slider level.

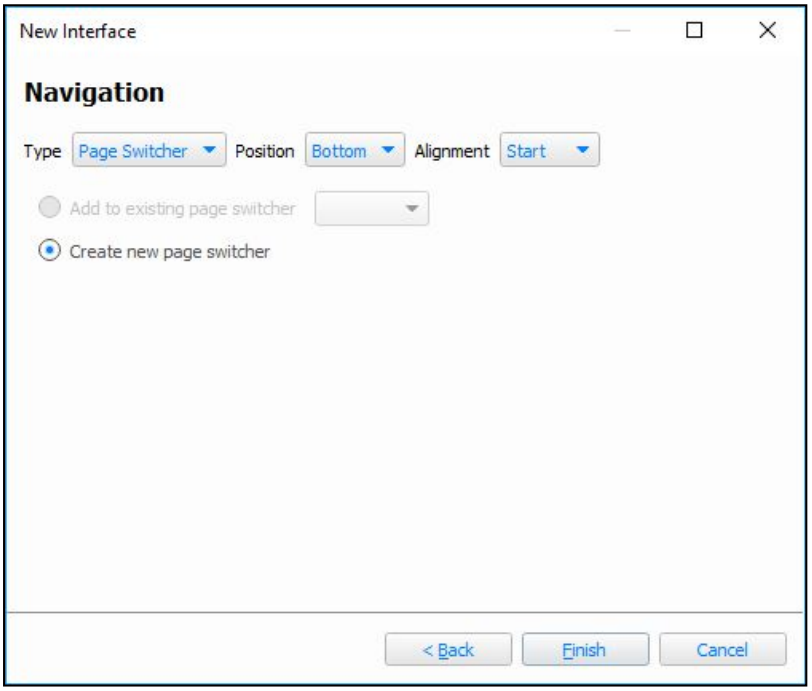
Page Navigation

Configuring Page Navigation

There are two methods for managing navigation between pages for projects that contain multiple pages:

- Page Switchers
- Navigation Buttons

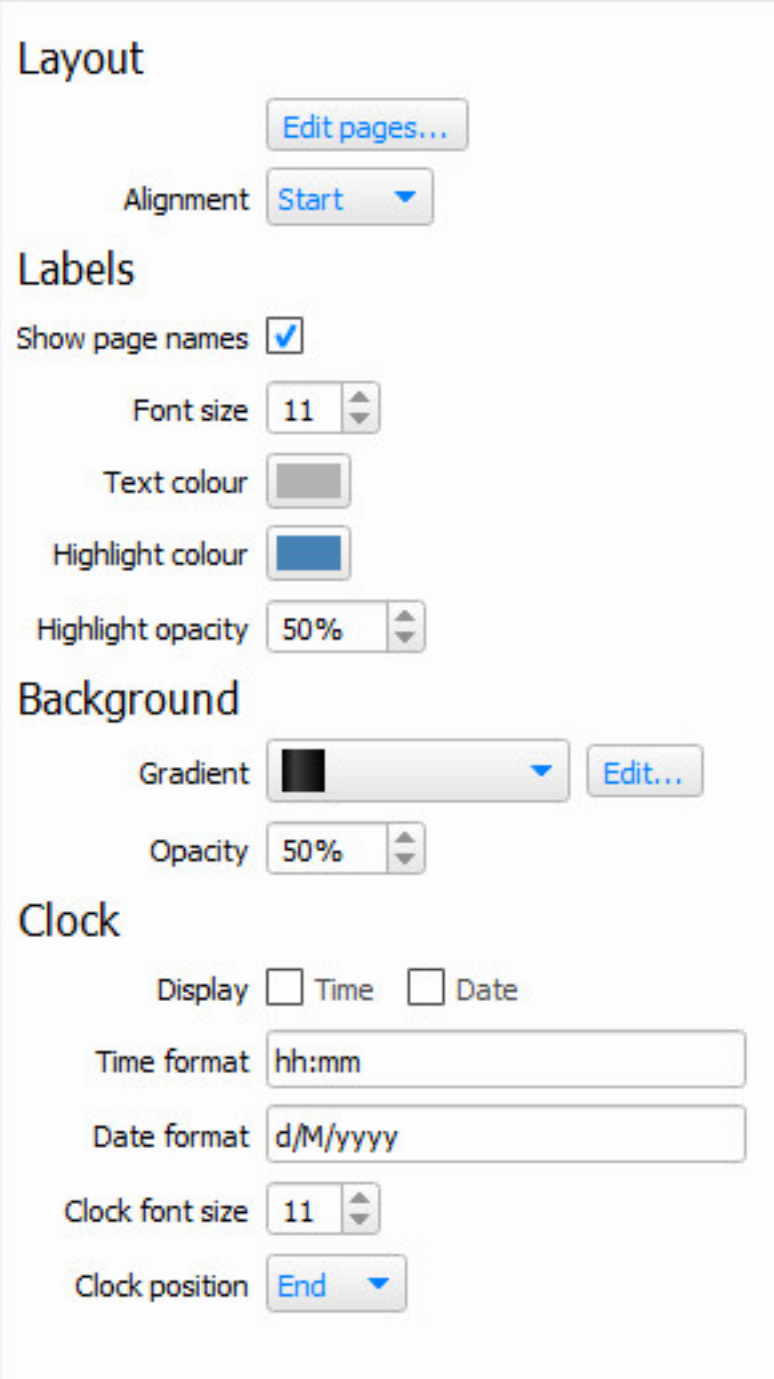
These can be created from the Navigation step of the new page wizard or by pressing New in the Page properties



Page Switchers

The position of a page switcher on the screen can be set along with its alignment. It is possible to use an existing page switcher from another page, or alternatively you may create a new page switcher.

Properties



Layout

Edit pages...: Adjust the pages included in the switcher and their order

Alignment: The alignment of the buttons within the switcher

Labels

Show page names: Should the page name be shown with the icon

Font size: The size of the text labels

Text colour: The colour of the labels on the switcher

Highlight colour: The colour highlight for the current page

Highlight opacity: Set the transparency of the highlight for the current page

Background

Gradient: The gradient to use on the switcher

Opacity: The opacity of the switcher's background

Clock

Display: Choose whether to display the time and/or date

Time/Date format: Specify the format of the displayed date and time

Clock font size: Font size for the clock

Clock position: Where to put the clock on the switcher

The pages in the page switcher can be adjusted later by right-clicking the page switcher in the Page Preview window and selecting Edit Page Switcher.

Navigation Buttons

Navigation buttons can be positioned at the top or bottom of a page. Alignment options are Start, End, Center or Spread. A maximum of three buttons can be added and each button's function can be set from the following list:

- Next Page (go to the page after this one, governed by the order shown in the page browser)
- Previous Page (go to the page before this one, governed by the order shown in the page browser)
- Back (go to whichever page was shown before the current page)
- Go To Page

The function of navigation buttons can be adjusted at any time by selecting the button in Page Preview and changing the Local Function in the Property Editor:

Built-In Themes

Pharos Designer comes with some built-in themes that you may use directly in your projects, or edit with the Theme Editor as required. Knowledge of the states in a theme for each item (e.g. buttons, sliders, etc.) is useful when using the Set TPC Control State action in Designer. Changing the state of an item will change its appearance, and this allows you to provide feedback in your interface.

The built-in themes are as follows:

- Dark
- Light
- Aurora
- City
- Lite

Extra TPS Themes are available from our website.

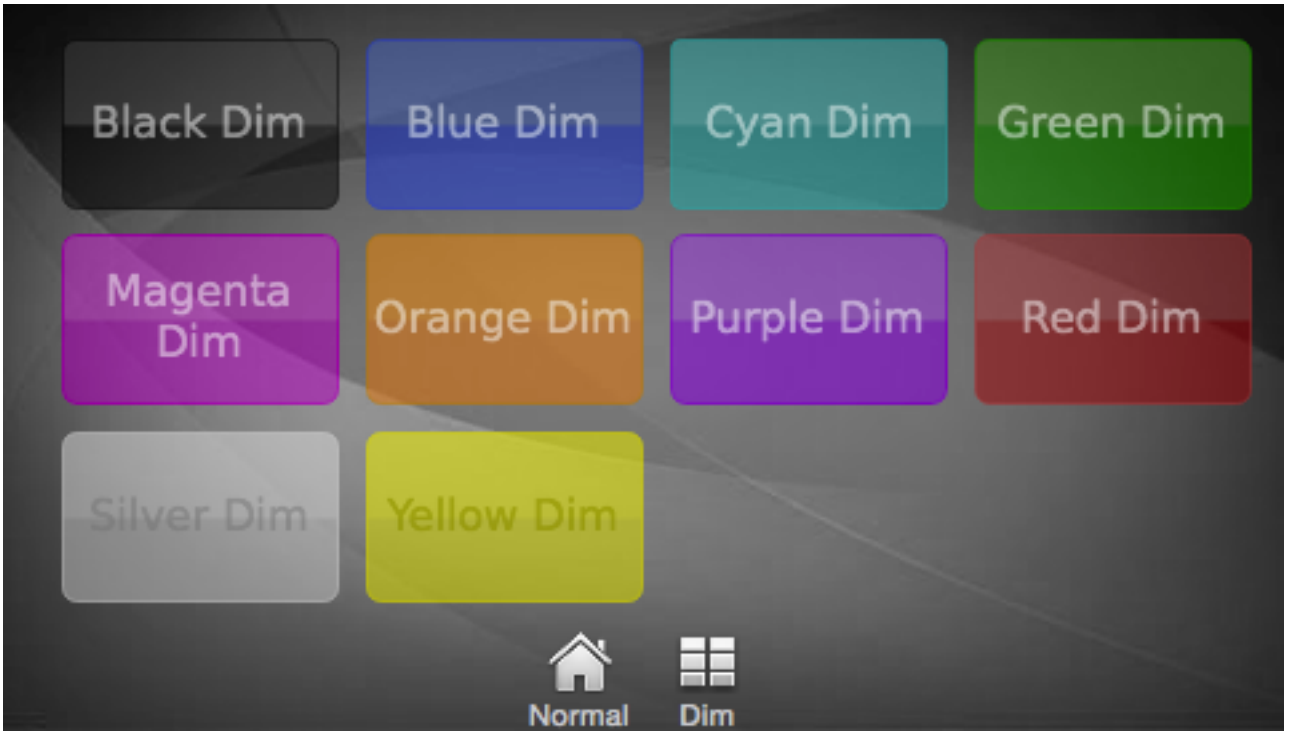
Dark Theme

The Dark theme is included with Designer. It has the following states for items:

Button States

The following states are shown in the image above:

- Black (default)
- Blue
- Cyan
- Green
- Magenta
- Orange
- Purple
- Red
- Silver
- Yellow



The following states are shown in the image above:

- Black Dim
- Blue Dim
- Cyan Dim
- Green Dim
- Magenta Dim
- Orange Dim
- Purple Dim
- Red Dim
- Silver Dim
- Yellow Dim

Slider States

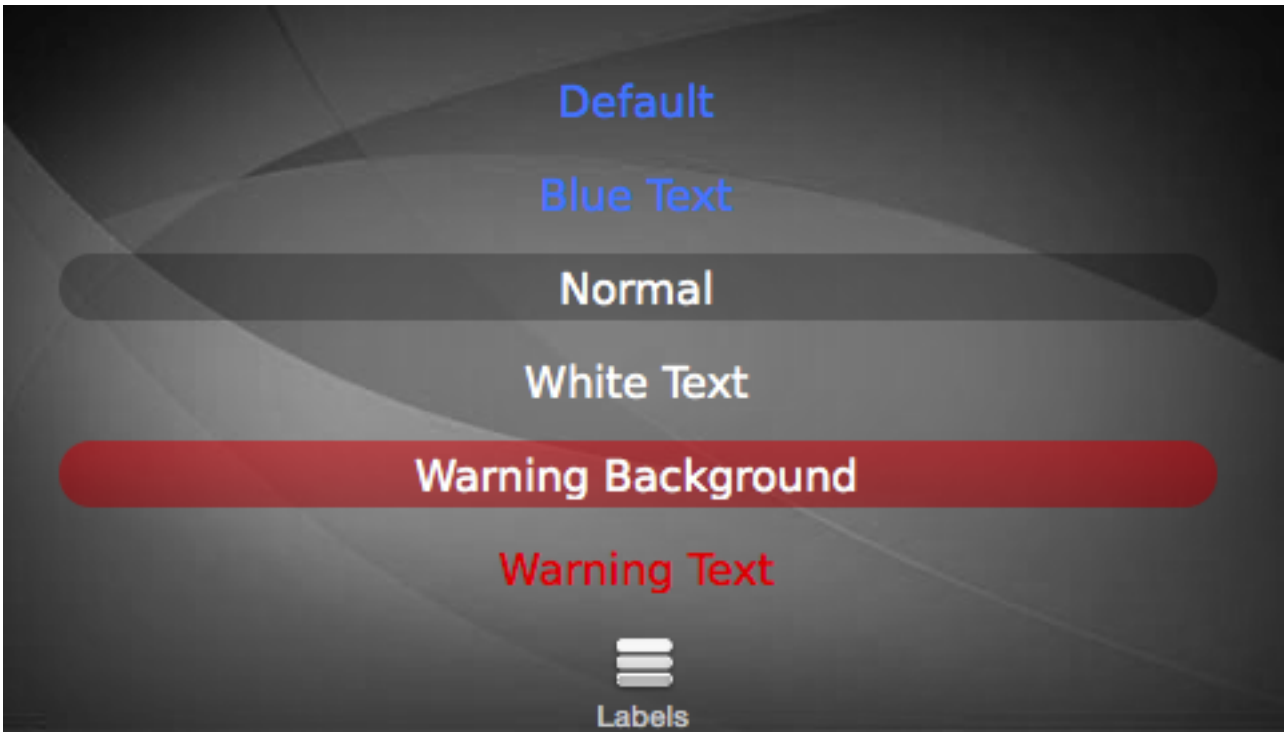
The following states are shown in the image above:

- Black (default)
- Blue
- Cyan
- Green
- Magenta
- Orange
- Purple
- Red
- Silver
- Yellow

The following states are shown in the image above:

- Black (default)
- Blue
- Cyan
- Green
- Magenta
- Orange
- Purple
- Red
- Silver
- Yellow

Label States



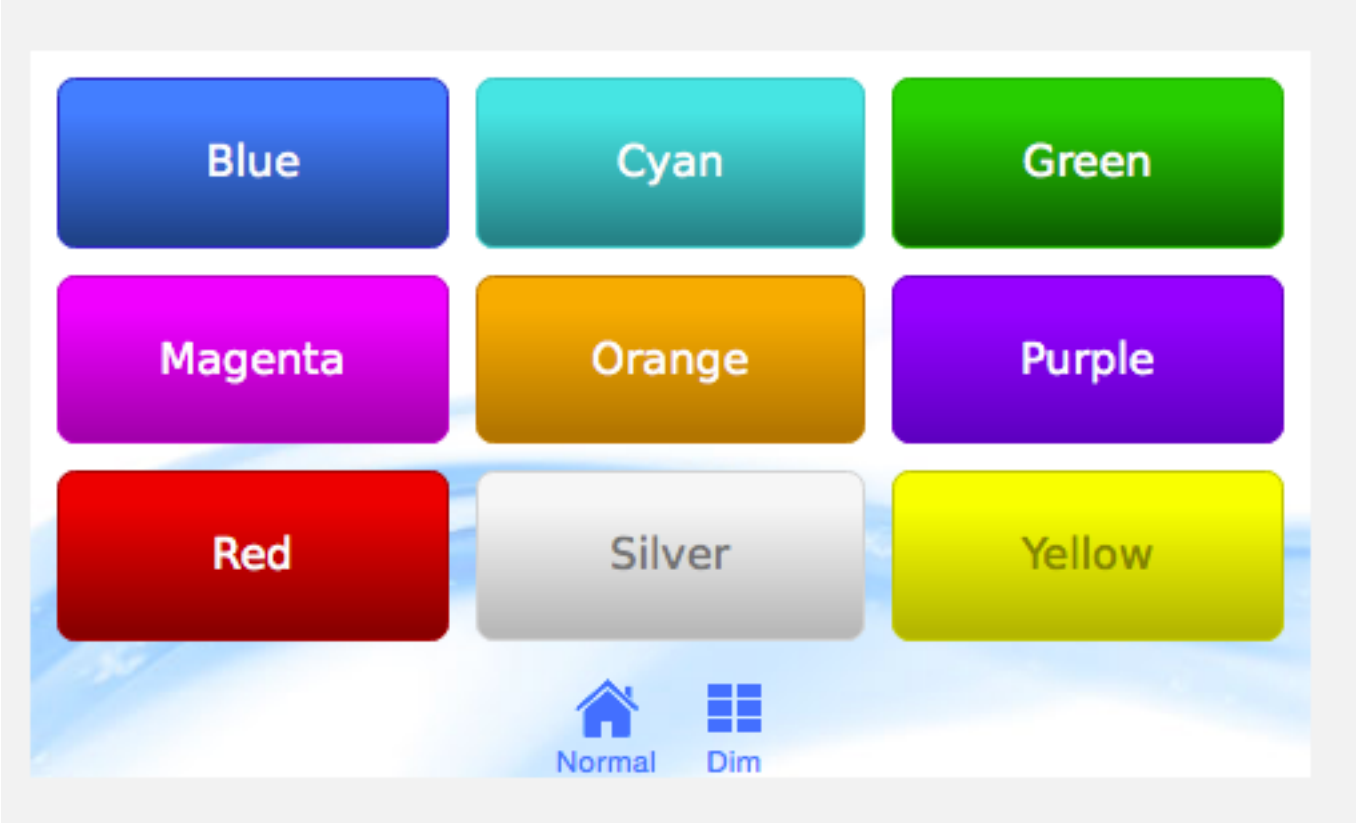
The following states are shown in the image above:

- Blue Text (default)
- Normal
- White Text
- White Background
- Warning Text

Light Theme

The Light theme is included with Designer. It has the following states for items:

Button States



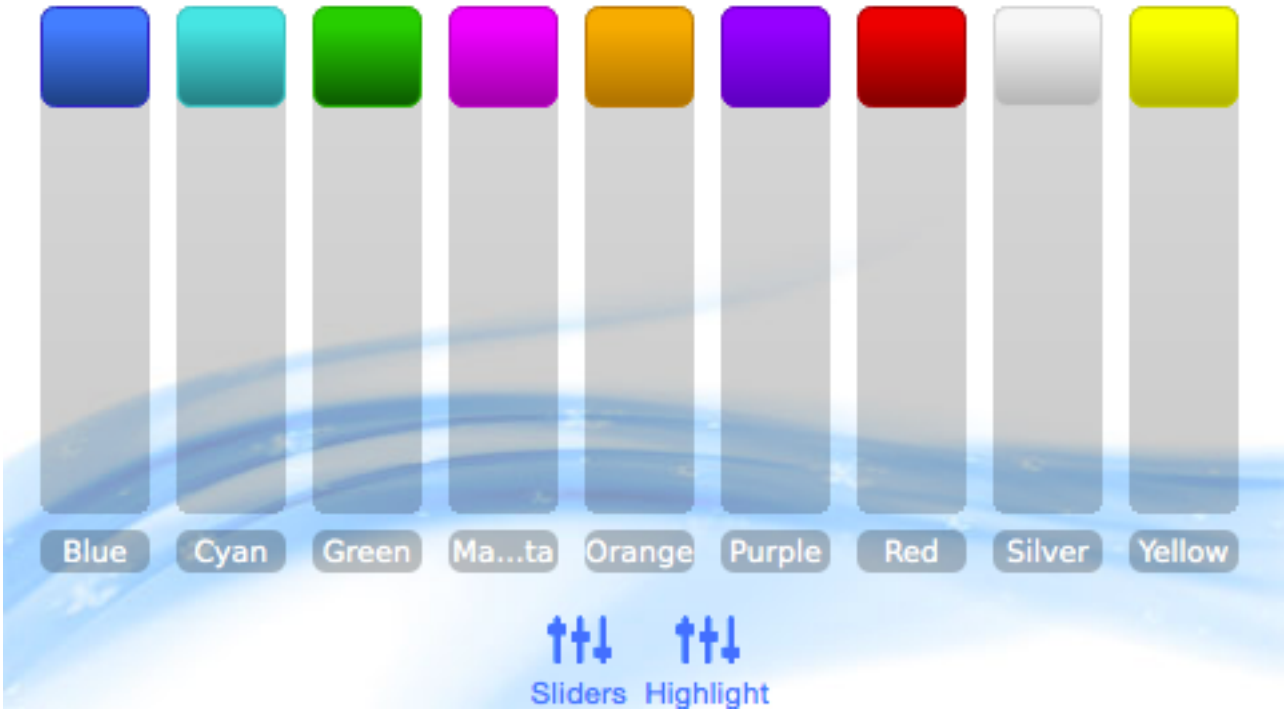
The following states are shown in the image above:

- Blue(default)
- Cyan
- Green
- Magenta
- Orange
- Purple
- Red
- Silver
- Yellow

The following states are shown in the image above:

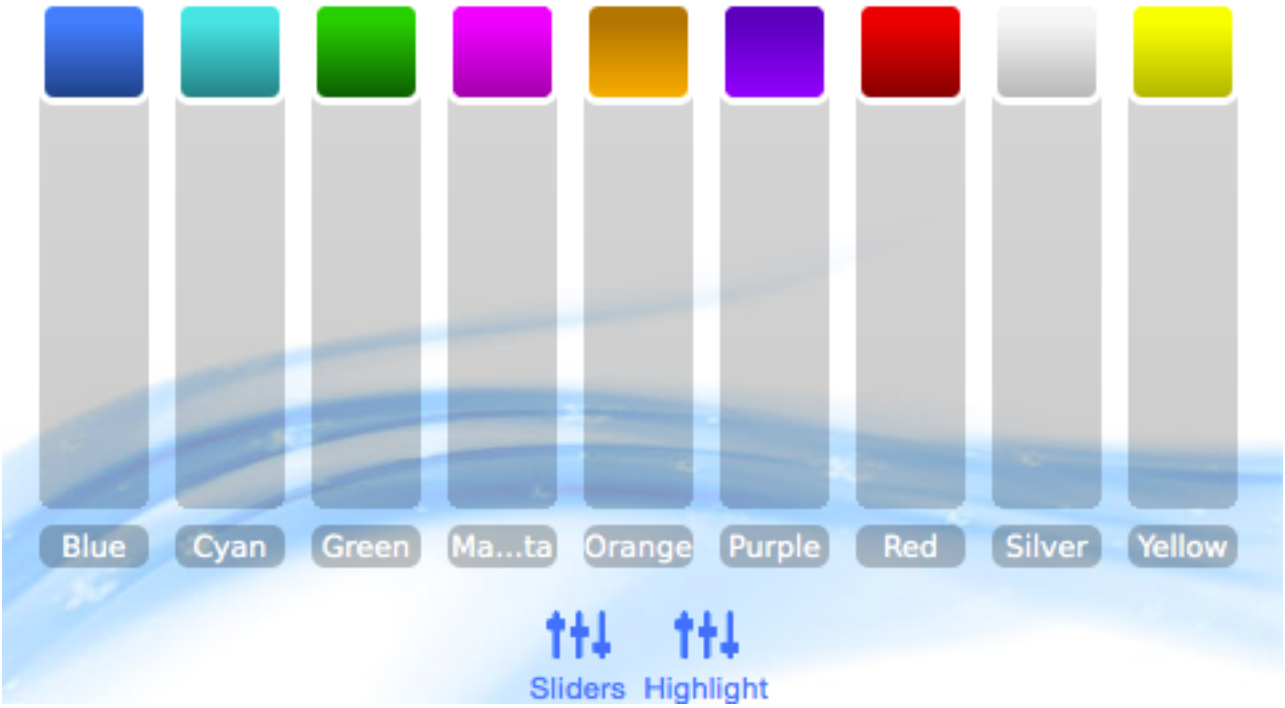
- Blue Dim
- Cyan Dim
- Green Dim
- Magenta Dim
- Orange Dim
- Purple Dim
- Red Dim
- Silver Dim
- Yellow Dim

Slider States



The following states are shown in the image above:

- Blue (default)
- Cyan
- Green
- Magenta
- Orange
- Purple
- Red
- Silver
- Yellow



The following states are shown in the image above:

- Blue
- Cyan
- Green
- Magenta
- Orange
- Purple
- Red
- Silver
- Yellow

Label States

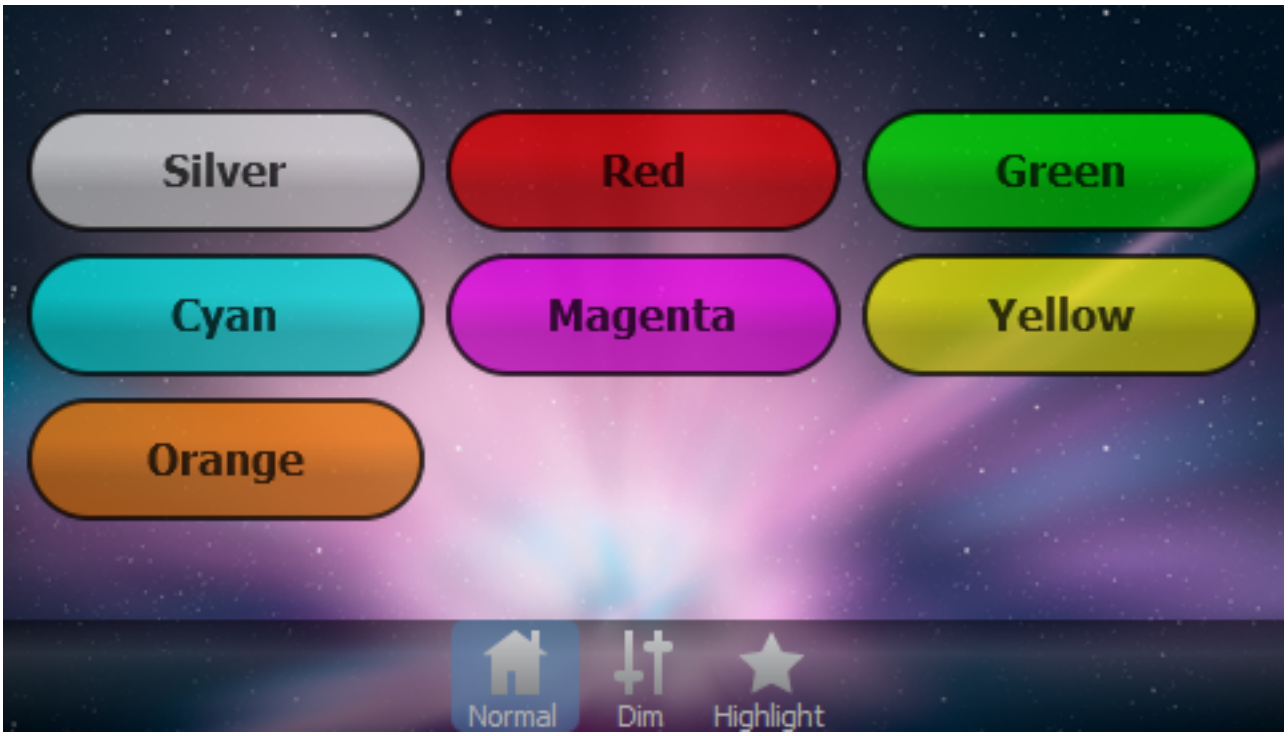
The following states are shown in the image above:

- Blue Text (default)
- Normal
- White Text
- White Background
- Warning Text

Aurora Theme

The Aurora theme is included with Designer. It has the following states for items:

Button States



The following states are shown in the image above:

- Silver (default)
- Red
- Green
- Cyan
- Magenta
- Yellow
- Orange

The following states use the same colours as the above, but they cause the opacity of the button to vary over a period of 1 second to attract attention.

- Silver Flashing
- Red Flashing
- Green Flashing
- Cyan Flashing
- Magenta Flashing
- Yellow Flashing
- Orange Flashing

The following states are shown in the image above:

- Silver Dim
- Red Dim
- Green Dim
- Cyan Dim
- Magenta Dim
- Yellow Dim
- Orange Dim

The following states are shown in the image above:

- Silver Highlight
- Red Highlight
- Green Highlight
- Cyan Highlight
- Magenta Highlight
- Yellow Highlight
- Orange Highlight

Slider States

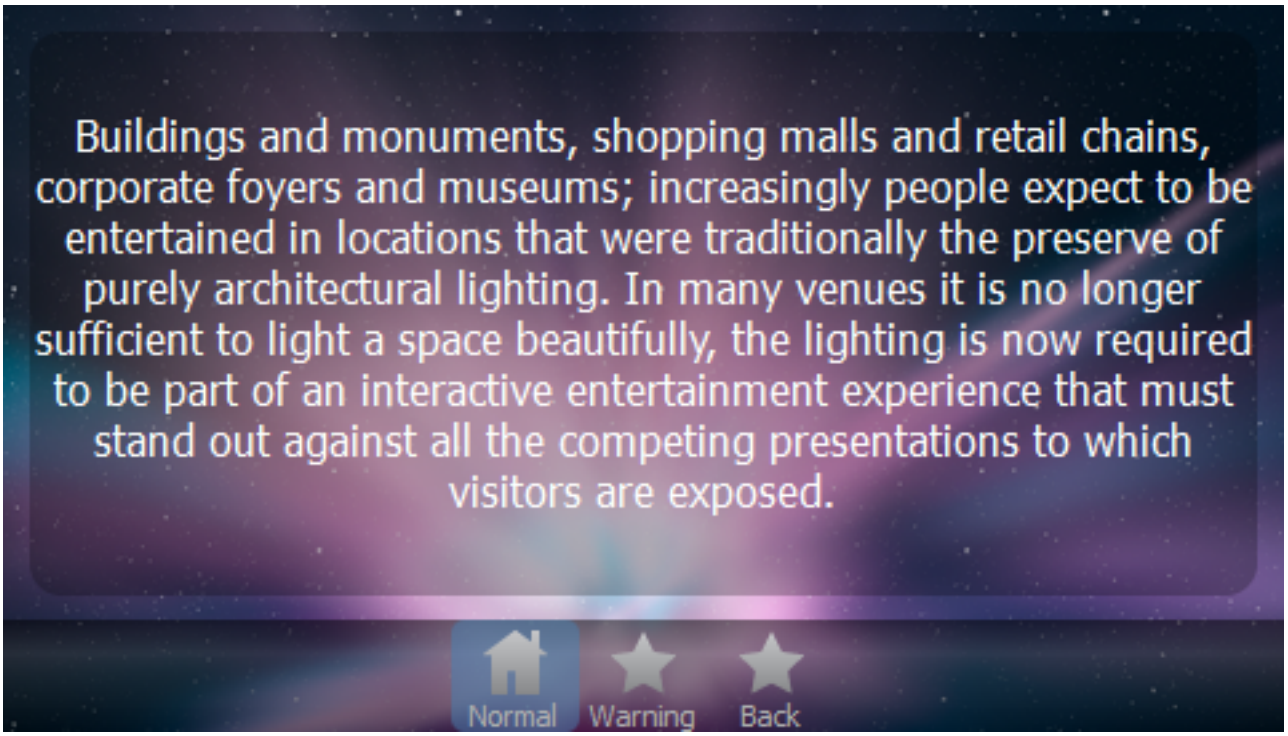
The following states are shown in the image above:

- Silver (default)
- Red
- Green
- Cyan
- Magenta
- Yellow
- Orange

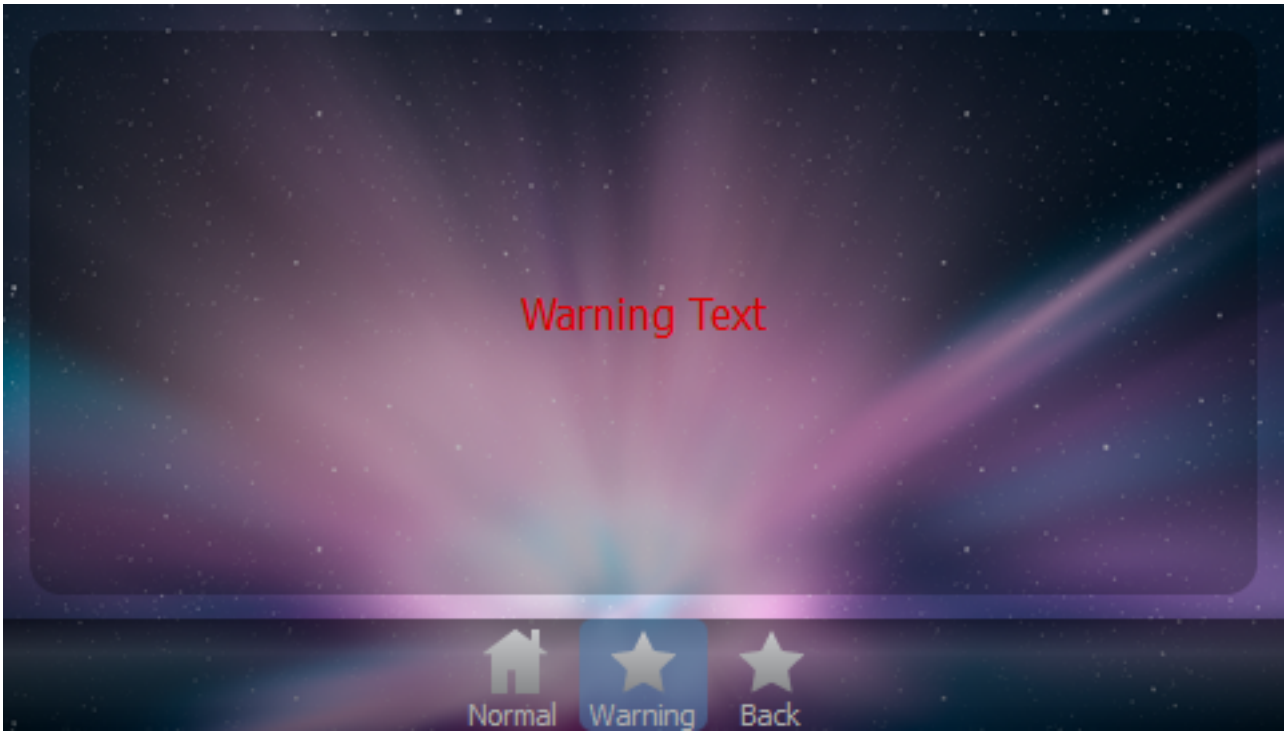
The following states are shown in the image above:

- Silver Highlight
- Red Highlight
- Green Highlight
- Cyan Highlight
- Magenta Highlight
- Yellow Highlight
- Orange Highlight

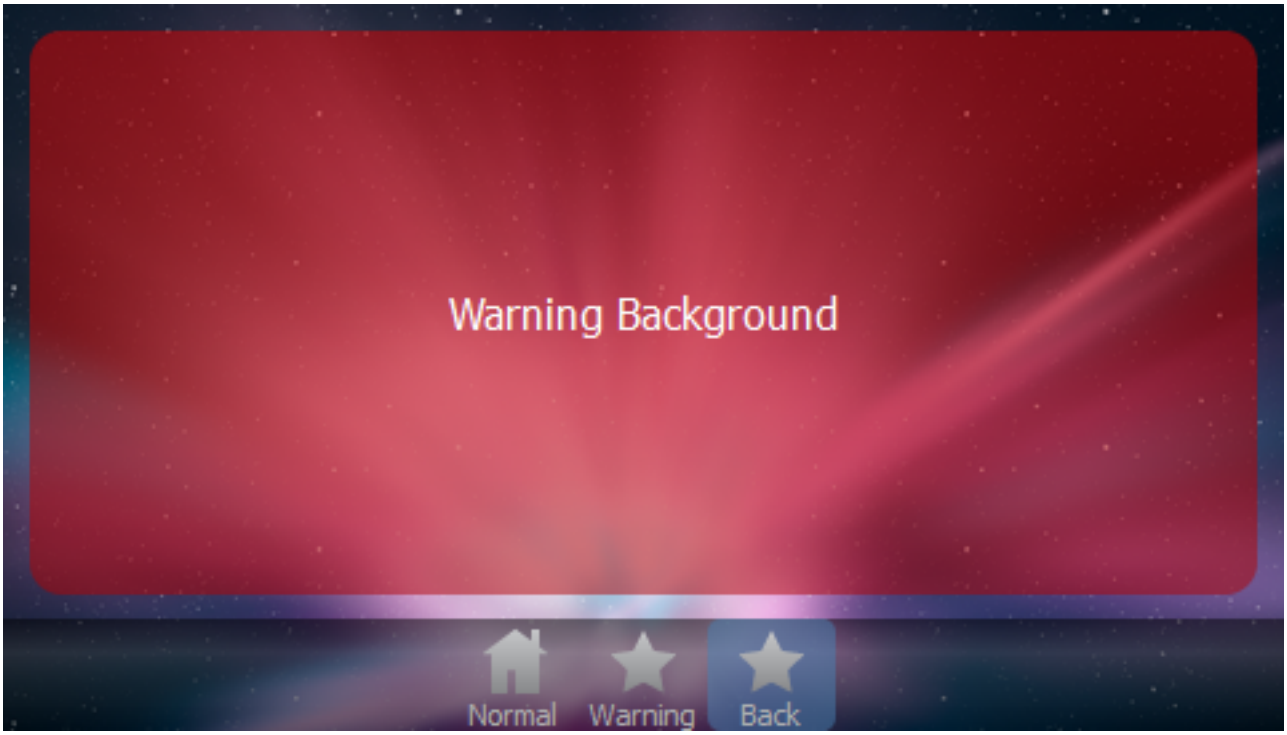
Label States



The **Normal** state (default) is shown in the image above.



The **Warning Text** state is shown in the image above.



The **Warning Background** state is shown in the image above.

City Theme

The City theme is included with Designer. It has the following states for items:

Button States

The following states are shown in the image above:

- Sky Blue (default)
- Lilac Tint
- Pale Lavender
- Light Red
- Light Pink
- Soft Amber
- Sea Green
- Pale Green
- Silver

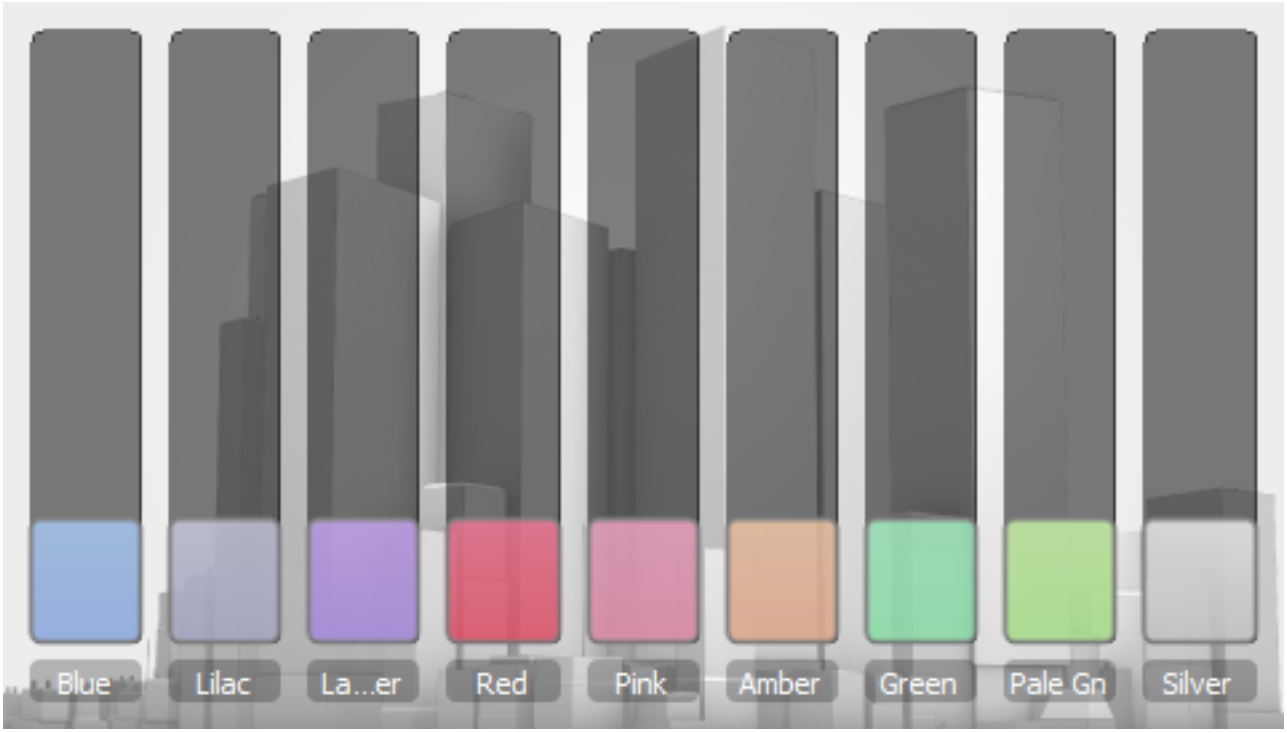
The following states use the same colours as the above, but they cause the opacity of the button to vary over a period of 1 second to attract attention.

- Sky Blue Flashing
- Lilac Tint Flashing
- Pale Lavender Flashing
- Light Red Flashing
- Light Pink Flashing
- Soft Amber Flashing
- Sea Green Flashing
- Pale Green Flashing
- Silver Flashing

The following states are shown in the image above:

- Sky Blue Dim
- Lilac Tint Dim
- Pale Lavender Dim
- Light Red Dim
- Light Pink Dim
- Soft Amber Dim
- Sea Green Dim
- Pale Green Dim
- Silver Dim

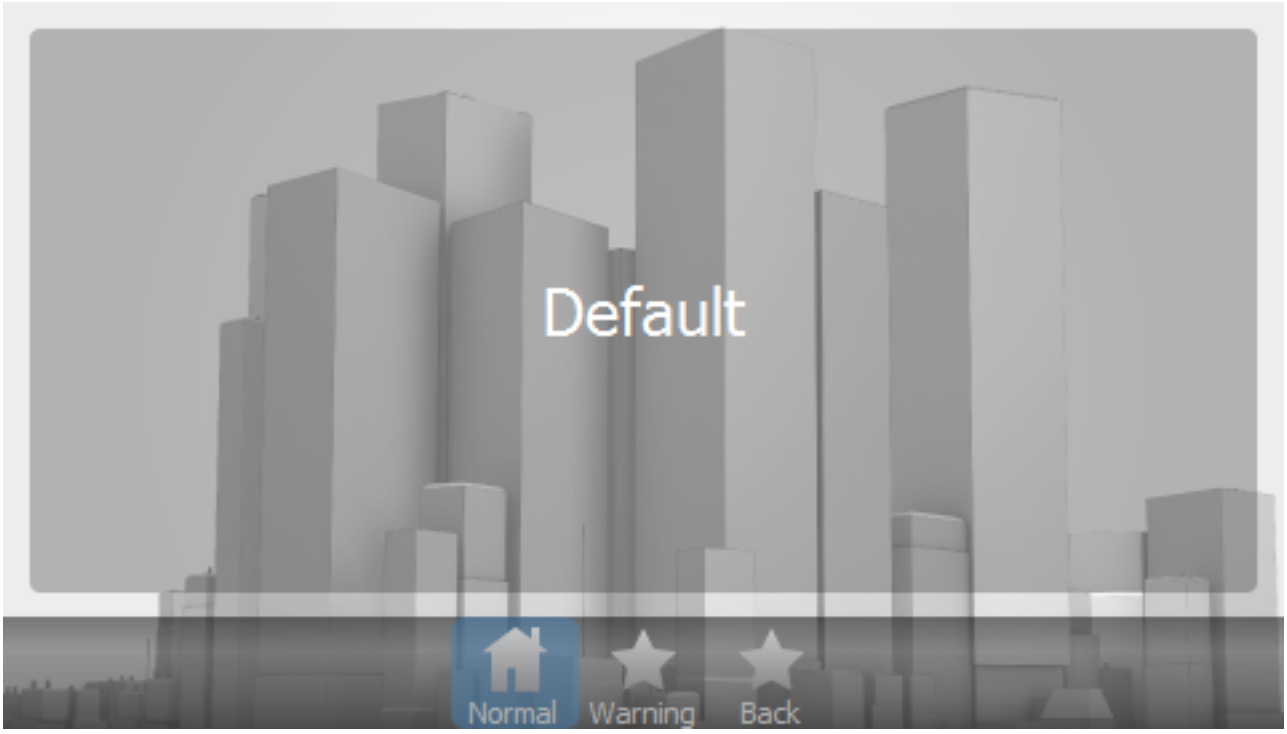
Slider States



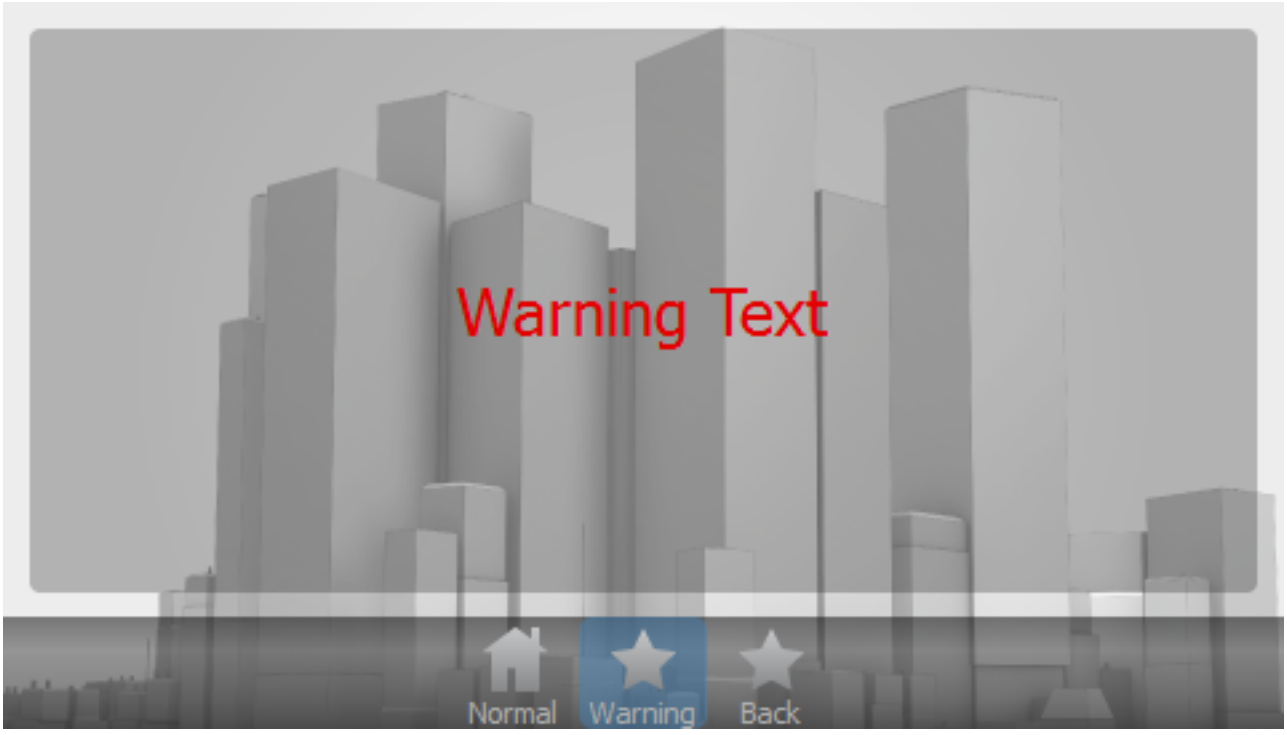
The following states are shown in the image above:

- Sky Blue (default)
- Lilac Tint
- Pale Lavender
- Light Red
- Light Pink
- Soft Amber
- Sea Green
- Pale Green
- Silver

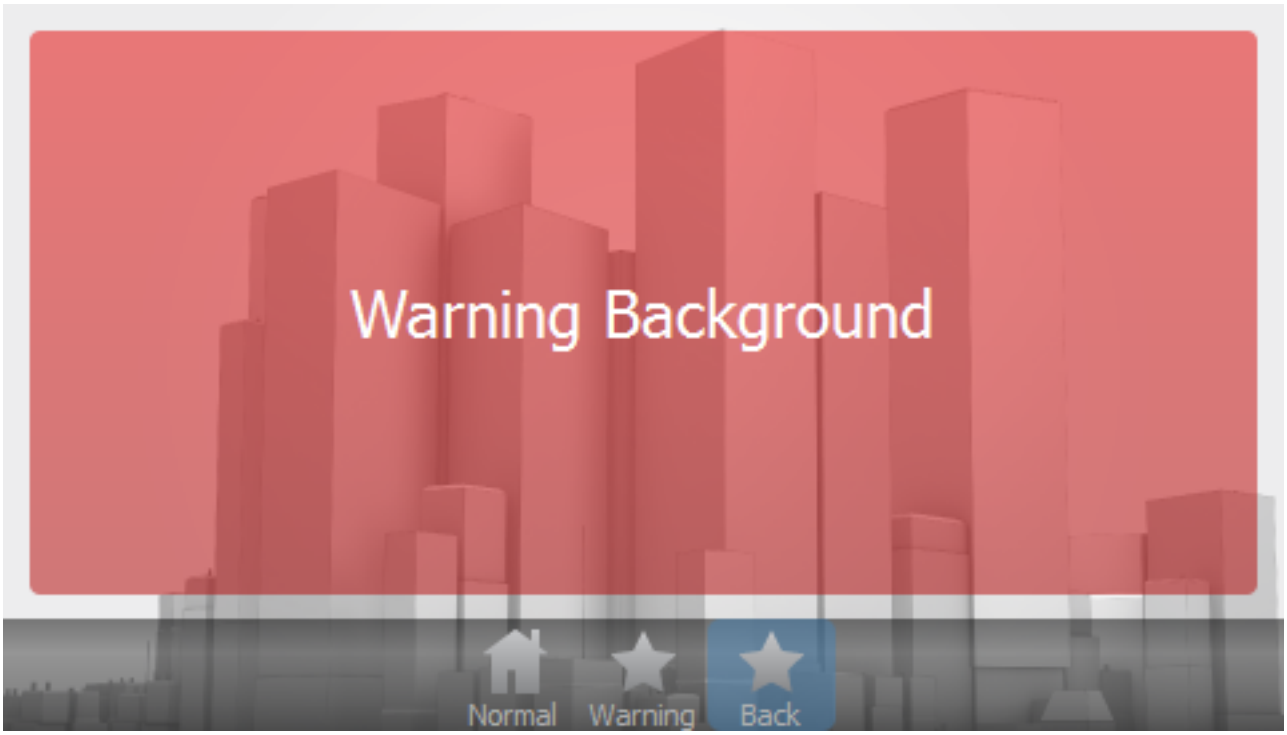
Label States



The **Default** state is shown in the image above.



The **Warning Text** state is shown in the image above.

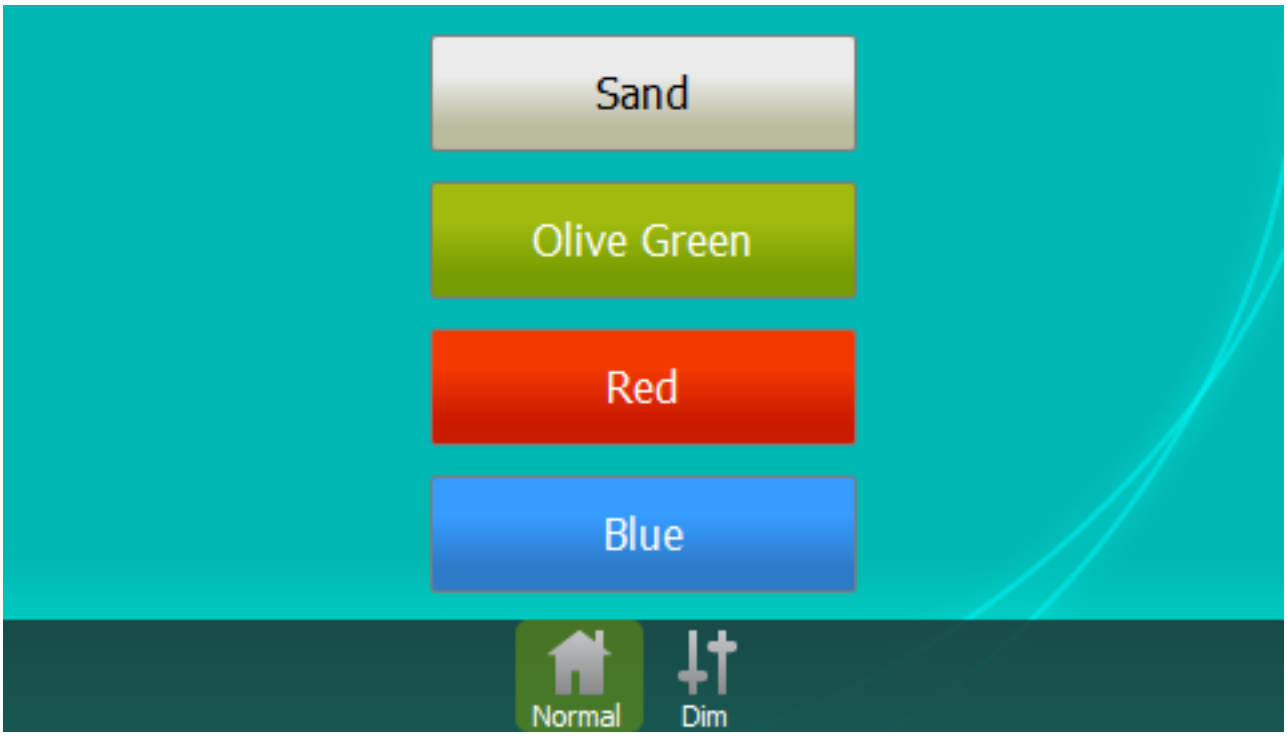


The **Warning Background** state is shown in the image above.

Lite Theme

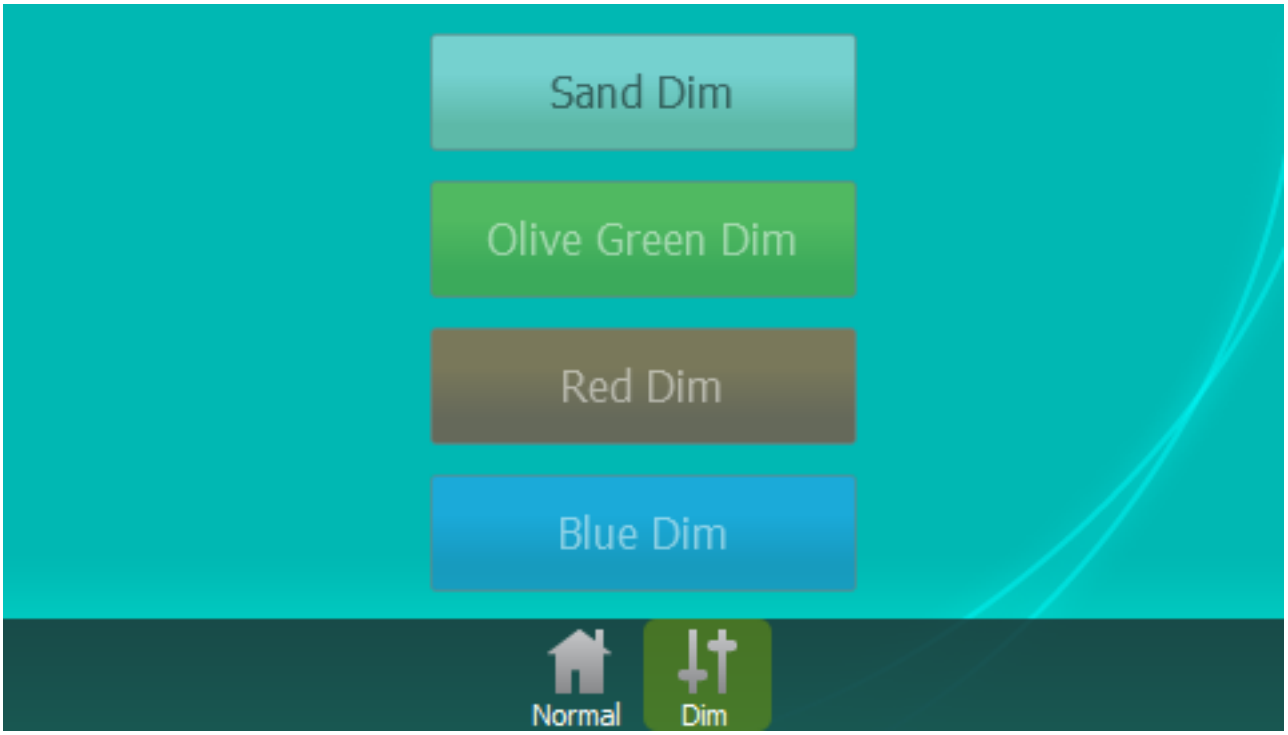
The Lite theme is included with Designer. It has the following states for items:

Button States



The following states are shown in the image above:

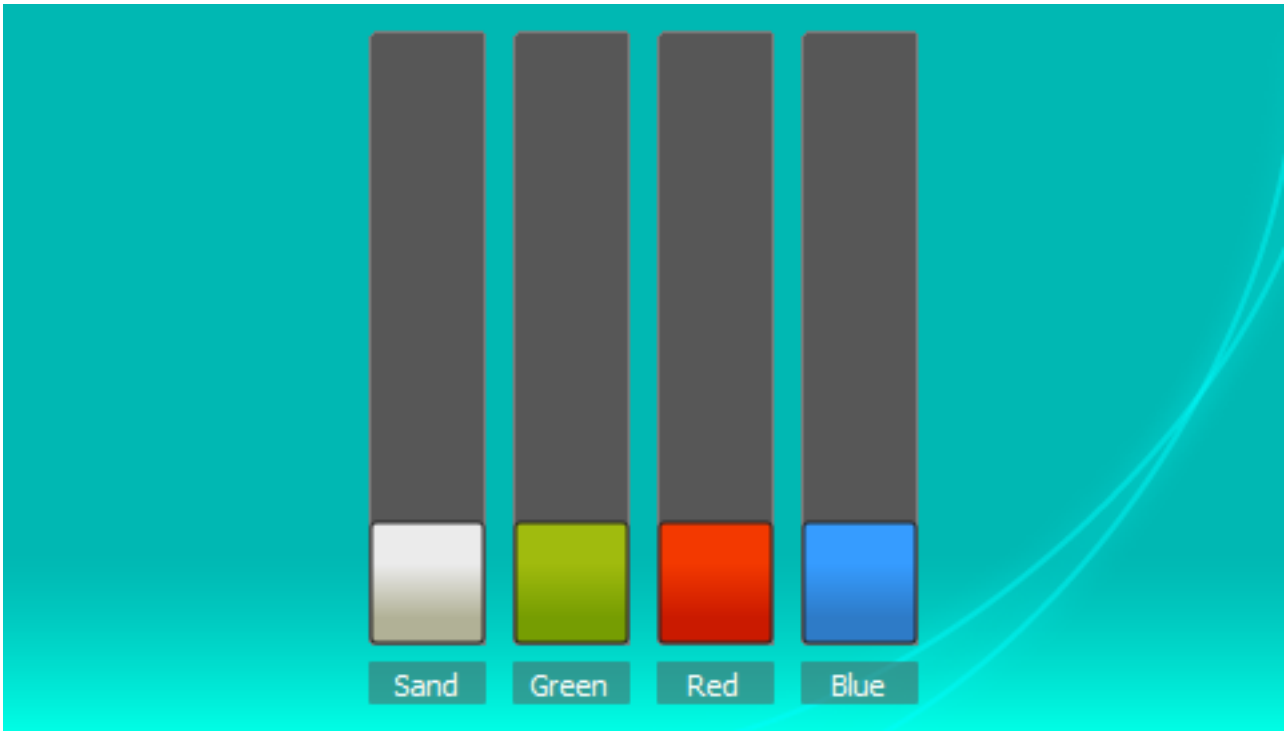
- Sand (default)
- Olive Green
- Red
- Blue



The following states are shown in the image above:

- Sand Dim
- Olive Green Dim
- Red Dim
- Blue Dim

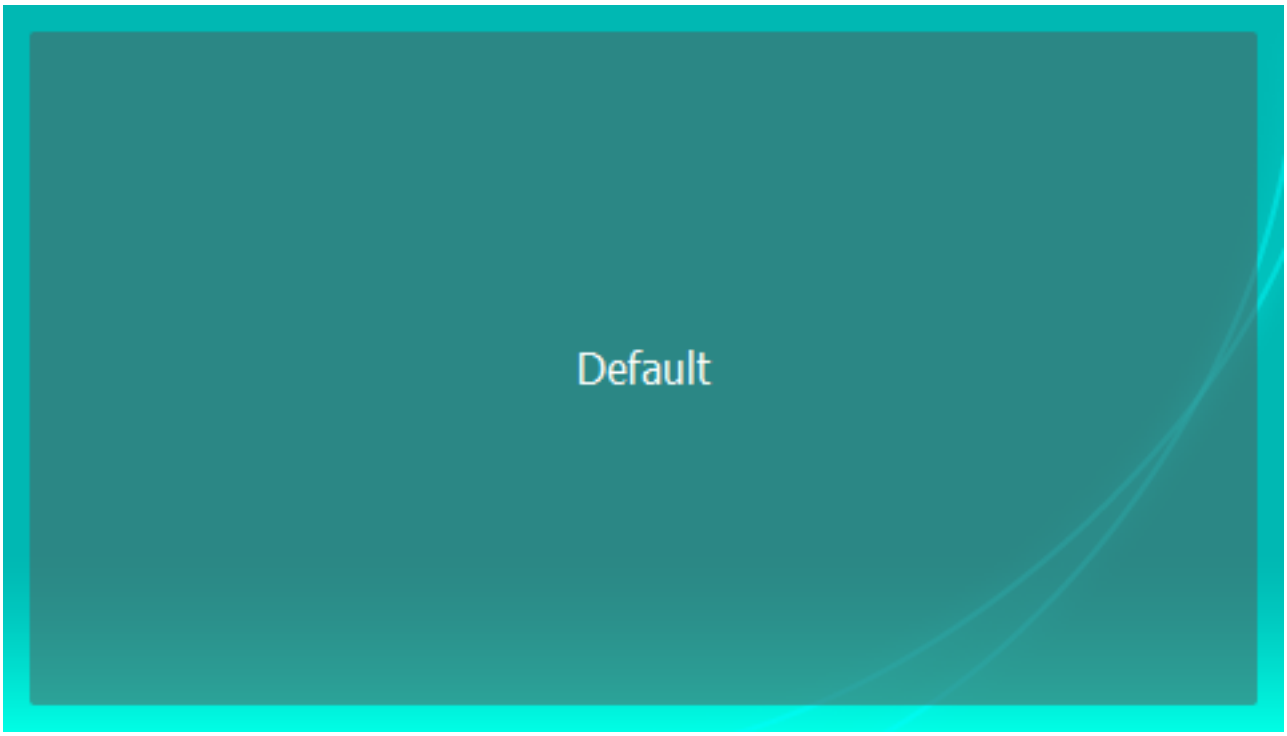
Slider States



The following states are shown in the image above:

- Sand (default)
- Olive Green
- Red
- Blue

Label States



The **Default** state is shown in the image above.

Theme Editor

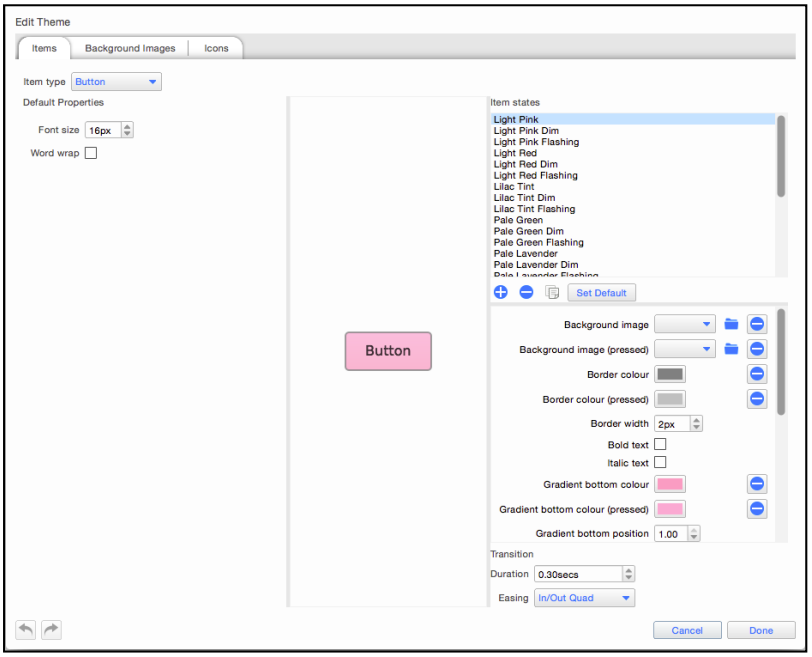
The theme editor facilitates the creation and editing of custom Touch Interface Themes. It allows you to add and edit background images, icons and item states.

Click Theme Editor on the toolbar to launch the theme editor.

Editing a project theme

The theme editor has three tabs for editing different aspects of a theme.

Editing item states



Default Item Properties - these set default values for certain properties that will be applied to an item when it's created for the first time. These properties can usually be edited in the main property editor of Interface view.

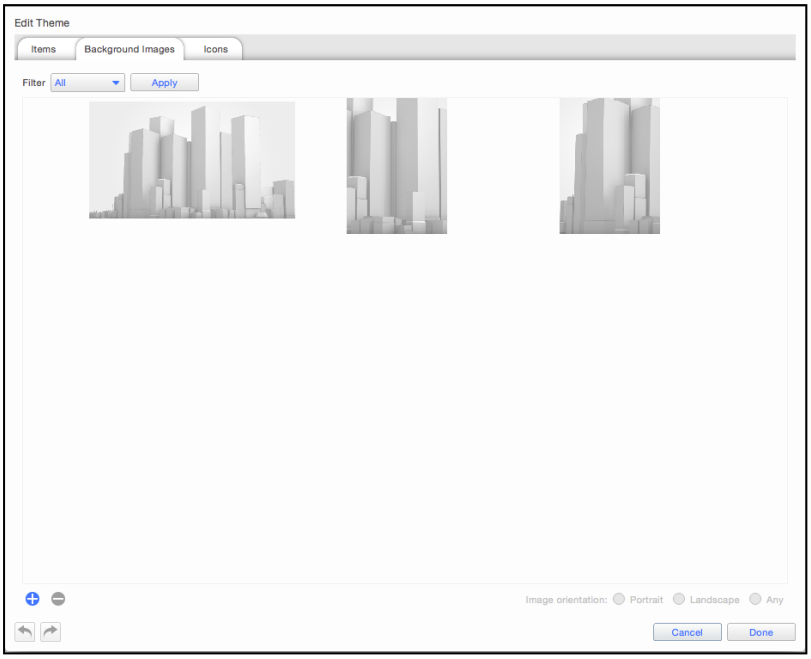
Item States - Select a state to edit its properties. The default state is shown with "(Default)" after its name.

Double-click a state (Windows) or press the Enter key (OS X) to rename the selected state. Click to add a new state. Click to delete a state (not possible for the default state). Click to duplicate a state. The state of an item can be changed using a Set Touch Control State action (see Actions for more details) for more information about Touch Device actions.

State Property Editor - Edit the properties of the selected state. Works in the same way as the main property editor.

State Transition Editor - Edit the transition that is applied to the item properties when the current state is applied to the item. Easing is the curve that property values will follow.

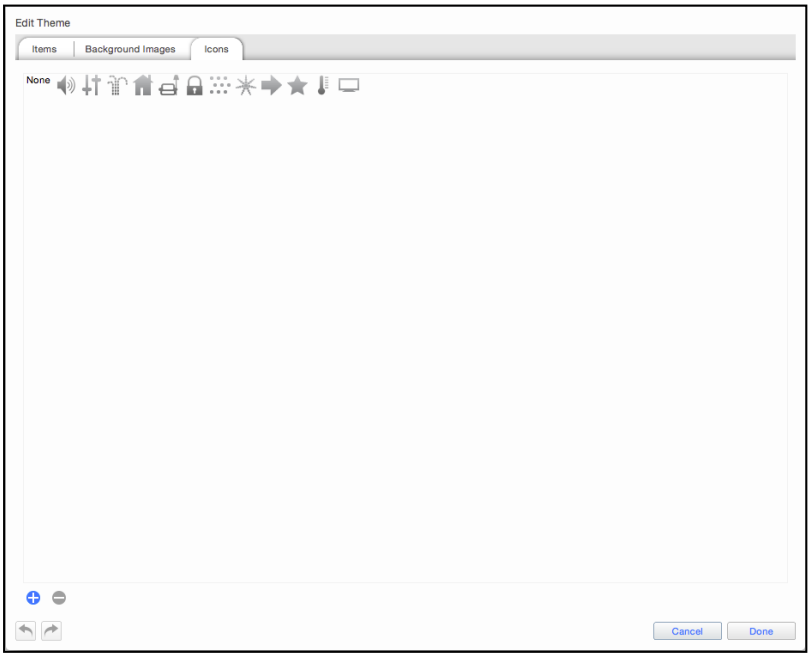
Editing background images



Click to add a new background image from a file. Click to remove the currently selected images from the theme. The image files will not be deleted.

You can set the orientation of images so that they are only offered as backgrounds for projects of the same orientation. If the image isn't specific to an orientation, for example if it's meant for tiling or centring on the screen, then set its orientation to 'Any'. You may filter which images are shown using the drop down near the top of the window.

Editing icons



Click to add a new icon from a file. Click to remove the currently selected icons from the theme. The image files will not be deleted.

NOTE: When using .svg files for images, ensure they use the SVG Tiny 1.2 profile. If in doubt, please contact support.

Export project theme

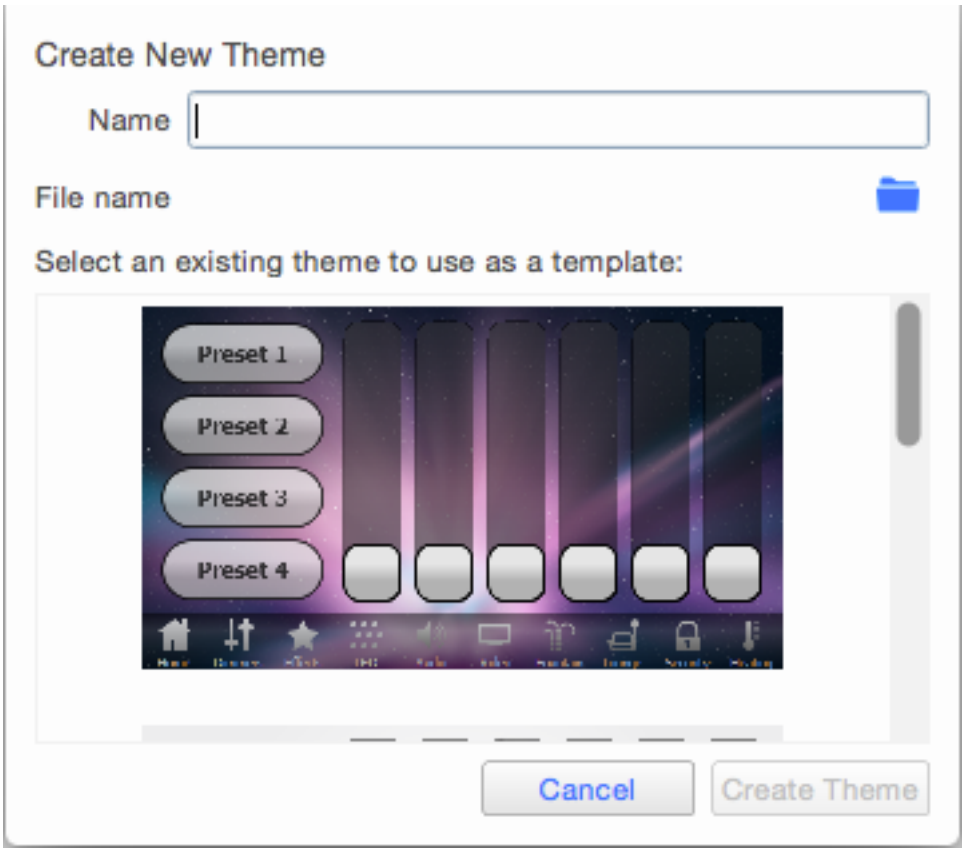
The Theme Editor launch button has an option to export the theme in the current project. This is useful for using the same theme on different projects.

You can access this option using the arrow to the right of the Theme Editor button

To export a theme you will have to provide a file name as well as a directory for the theme to be saved to.

Creating a new theme

To create a new theme you will need to create a new interface and go to the "Select the Theme" page. Click on the at the top of the window to create a new theme. You'll need to give the new theme a file name and choose a file path. You'll also need to choose a theme to use as a template.



You will now see the Theme Editor where you can edit item states in the new theme.

Saving changes to a theme

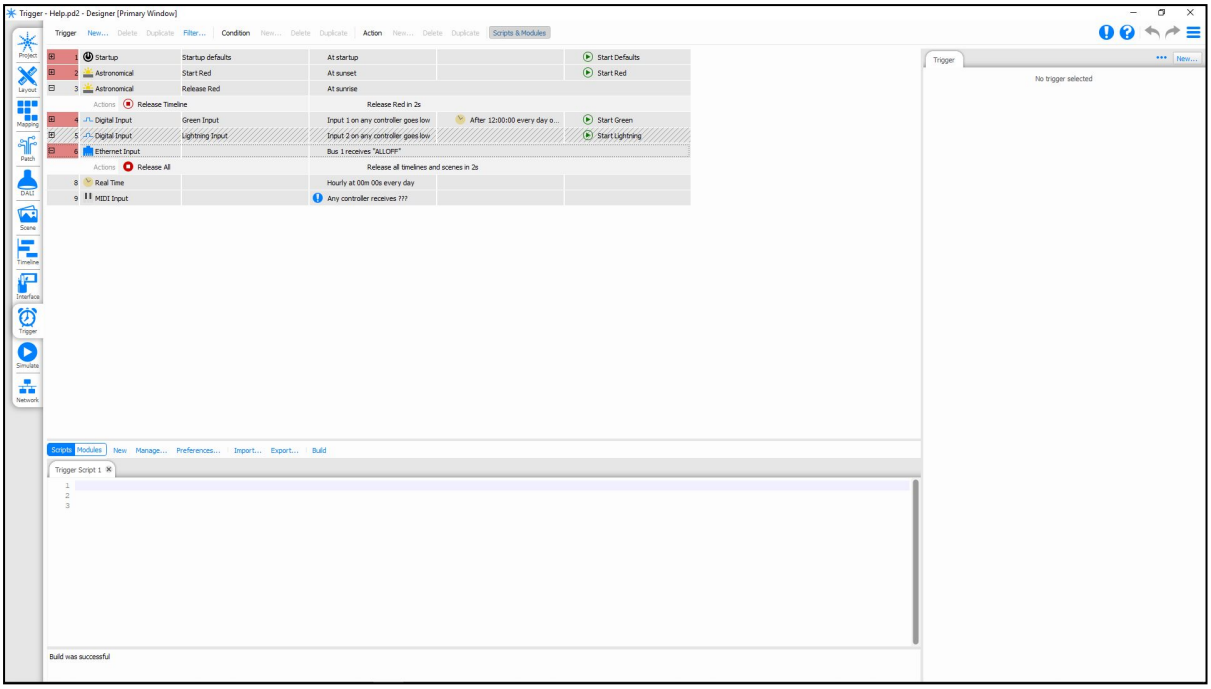
Click Done when you've finished editing the theme and your changes will be saved to the theme file. If you're creating a new theme, the theme will now be shown in the theme browser and offered when you create a new interface.

Trigger Overview

Keyboard Shortcuts

Ctrl+N	Create a new trigger of the last created type
Ctrl+left-click *on a trigger, condition or action*	Toggles its selection
Shift+left-click	Select a range of triggers, conditions or actions
Ctrl+A	When nothing is selected, select all triggers; when a condition or action is selected, selects all conditions/actions of the parent trigger
Hold Ctrl *while dropping a dragged trigger*	Create a copy of the trigger at the drop location
Hold Shift *while dropping a dragged condition or action*	Move the condition or action to the trigger it is dropped on
Delete/Backspace	Delete selected triggers, conditions or actions
Up/Down	Move current row indicator up and down, and select the row
Shift + Up/Down	Move current row indicator up and down, and add the row to the selection
Ctrl + Up/Down	Move current row indicator up and down, but don't change the selection
Left/Right	Collapse/Expand current trigger
Space	Select current row
Ctrl + Space	Add current row to the selection
Ctrl+B *in Script Editor*	Compile script

The Trigger window is used to “connect” your timeline programming to the outside world:



Controllers support a range of interfaces which can be used to Trigger the playback engine including an internal real time & astronomical clock. For example, digital input #1 (connected to a wall panel) could be set to start “Funky” timeline, “Advert” can be set to run on the hour every hour between sunrise and sunset and, at sunset, “Cleaning” would start.

This tab contains two sections. The main section in the middle is the Trigger management area

Creating a Trigger

To create a Trigger, click New Trigger in either the Trigger toolbar or the Configuration Pane. Select the required type from the searchable dropdown menu, see Triggers.

Configuring a Trigger

- Type - the Trigger type
- Number - the Trigger's unique number as used for Web Interface control purposes (can typically be left at the default value unless creating custom pages)
- Name - an identifier for the Trigger used in the Trigger Management Area, Simulate and Web Interface
- Description - a more detailed description for use in Reports.
- Group - the group that the Trigger belongs to (defined by colour) e.g. Red = outside, Green = inside
- Controller - the controller that will process the Trigger, note that Real Time, Astronomical, Ethernet Input and IO Module Triggers are processed by the Network Primary. Note: This is only available if the feature is activated (using advanced feature option •••)
- Test conditions on - Certain circumstances will require a condition to be tested on a device separate to the device receiving the trigger (e.g. Touch control trigger needing to test timeline status on LPC)
- Absorbed - uncheck to prevent the Trigger absorbing the match, see below
- Included - uncheck to hide this trigger from the controller's Web Interface and Director
- Enabled - uncheck to prevent the Trigger from running
- Parameters - the data required for each Trigger type, varies by type so refer to the appropriate Trigger descriptions. Note that if a parameter only has one option (e.g. only one timeline in the project), then it will be selected by default.

Inhibiting a Trigger

For testing purposes it is sometimes useful to inhibit one or more Triggers to examine more clearly the operation of others. A Trigger can be inhibited by unchecking the Enabled box in the Trigger configuration, the row details will be displayed in a grey .

Hiding a Trigger

By Default Triggers are available to be viewed and fired from various locations, including the Default Web Interface, and Director. To prevent triggers from being visible in these locations, they can be hidden using the Included checkbox within the Trigger properties.

Trigger Numbering

Under most circumstances the trigger numbers have no bearing on the operation of the project file. Users should be careful when changing trigger numbers if the controller API is utilised in a project.

It can be desirable to change the numbering of Triggers such that they are continuous through the project once all triggers are created and put in order (see below). This can be achieved by manually changing the Trigger numbers in the trigger properties or:

- Selecting multiple triggers
- Right-clicking on a selected trigger
- Choosing Renumber triggers...

In the popup window, the Starting number can be set, e.g. if a set of triggers relating to a section of the project should be together in the 10xx range.

The selected triggers will be numbered incrementally from the starting number in the order that they are present in the trigger list.

Trigger order & matching

The order in which Triggers are displayed in the Trigger Management Area is the order in which they are tested by the system. Once a Trigger is successfully matched then, if "Absorb on match"is checked, no further Triggers are tested for that event; the event is absorbed. Thus this Trigger order is important, particularly when using Conditions.

If you had two identical Triggers in your show then, assuming they had no Conditions, only the first one encountered would ever be matched. However, if you add a Condition to the first Trigger then it will only match when the Condition is true, and when it is false the second Trigger will match instead.

The ability to have the same Trigger have different results based on a Condition is very powerful. For instance you might have a single digital input that starts one timeline during the day and another during the night.

Changing the Trigger order

You can select and drag the Trigger up or down within the management area to redefine the order

Absorb on match

In some cases it is useful for a matched Trigger not to absorb the event and thus allow Triggers further down the list, so there exists the option to disable the default behaviour by unchecking the "Absorb on match" box for each Trigger as required.

Conditions

If you wish to constrain the Trigger with a Condition then use the New Condition button in either the Trigger Toolbar or the Condition Configuration area. Select the required type from the searchable dropdown menu, see Conditions.

Up to 32 Conditions can be applied to each Trigger in this way and you can select each one for configuration from the Trigger Management Area.

Configuring a Condition

- Type - the Condition type
- Negate - check to invert the operation of the Condition i.e. if the Condition does not match
- Parameters - the data required for each Condition type, varies by type so refer to the appropriate Condition descriptions. Note that if a parameter only has one option (e.g. only one timeline in the project), then it will be selected by default.

Changing the Condition order

To change the order in which Conditions are tested, select the Condition in the Trigger Management Area and drag it to the required location within the Trigger.

Actions

Every Trigger needs an Action, the thing to do, which you can add to a Trigger using the New Action button in either the Trigger Toolbar or the Action Configuration area. Select the required type from the searchable dropdown menu, see Actions.

Up to 32 Actions can be applied to each Trigger in this way and you can select each one for configuration from the Trigger Management Area.

Configuring an Action

- Type - the Action type
- Controller - the controller that will process the Action
- Parameters - the data required for each Action type, varies by type so refer to the appropriate Action descriptions. Note that if a parameter only has one option (e.g. only one timeline in the project), then it will be selected by default.

Changing the Action order

To change the order in which Actions are executed, select the Action in the Trigger Management Area and drag it to the required location within the Trigger.

Copying a Trigger, Condition or Action

Select the Trigger, Condition or Action to be copied in the management area and use the Copy option in the appropriate section of the Trigger Toolbar or right-click > Duplicate [Trigger, Condition or Action] to create a duplicate immediately below the current selection. When you duplicate a Trigger, the Conditions and Actions for that Trigger are duplicated as well.

To create a copy of an Action in a different Trigger, you can select the Action in the Trigger Management Area and drag it into the destination Trigger.

Deleting a Trigger, Condition or Action

Select the Trigger, Condition or Action to be deleted in the management area and use the Delete option in the appropriate section of the Trigger Toolbar or right-click > Delete [Trigger, Condition or Action] delete the current selection. When you delete a Trigger, the Conditions and Actions for that Trigger are deleted as well.

Trigger Filtering

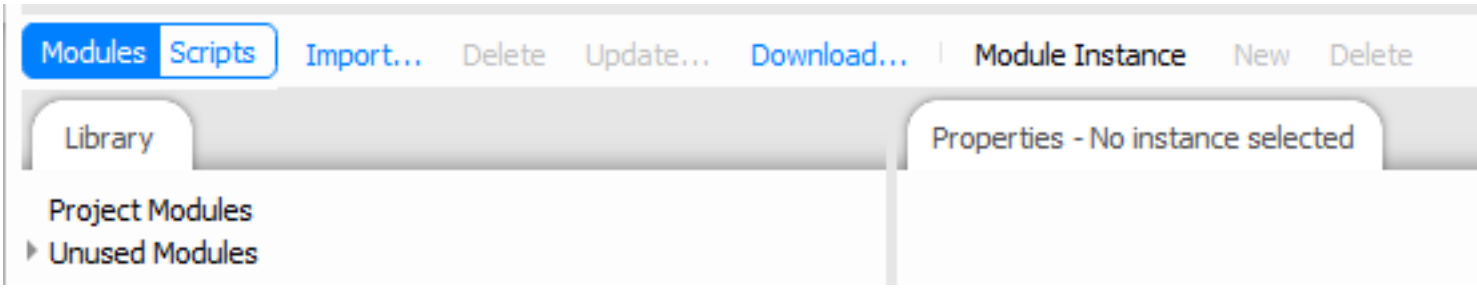
To filter Triggers by their group, you can use the Filter... option in the Trigger Toolbar. The Filter toolbar will be shown:

You can filter Trigger by their Type or by their Group.

You can disable filtering by Type and Group with the appropriate checkboxes.

IO Module Editor

The IO Module editor allows you to add IO Modules to your project. These IO Modules (IOMs) are small snippets of code that allow your Pharos Designer system to interact with 3rd party devices, or add additional functionality into your project file that was only possible via scripting before.



Import... allows you to import an IOM file from your hard-drive. This is only really necessary when you have a custom IOM or have been sent one from Support.

Delete allows you to delete an IO Module from your computer.

NOTE: Deleting an IOM will remove it from your IOM file in Designer's default folder system, so will no longer be visible in further projects. If this was done accidentally, the IOM can either be re-downloaded or re-imported.

Update... allows you to update an existing IOM from a source file on your hard-drive. This is only necessary if you are using an imported, rather than downloaded, IOM.

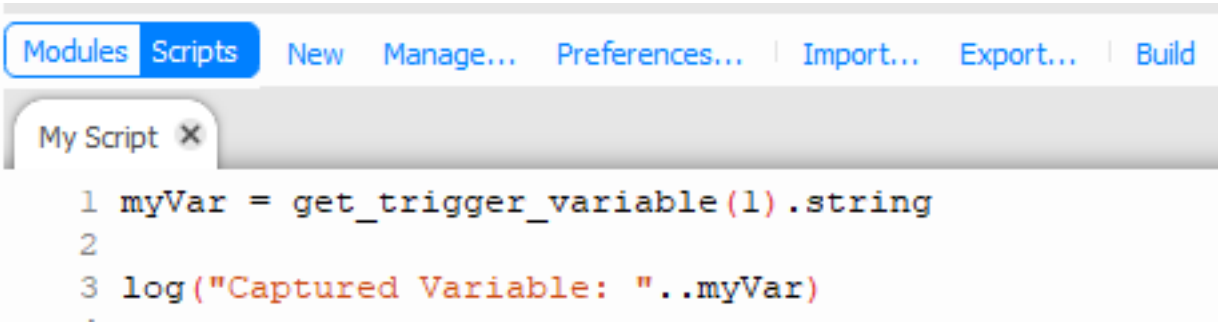
Download... will open the Online IO Module Library, which povides access to all released IO Modules and released betas.

Module Instances are the IOMs that are currently active within your project. Each IOM can be run in multiple instances, which ringfences them all off from each other. For instance, the timer of one will not trigger another instance.

Instances can be created or deleted from within the toolbar.

Lua script editor

The Lua Script Editor allows you to edit scripts from Triggers, Conditions and Actions within Designer. The Script Editor is launched by pressing the Scripts & Modules button on the Trigger Toolbar, and selecting Scripts in the bottom pane:



The main area of the editor is the code editor where you enter the source code of the script. The code editor will colour the Lua syntax to aid readability. Standard clipboard shortcuts and undo/redo are supported.

To create a new script for use in Conditions or Actions click New Script.

Scripts can be opened using the Open option and closed with the on the Script Tab.

To import a Lua script from an external file, use Import.

To save a Lua script to a file, use Export.

To compile the script and check for syntax errors, use Build. If there are errors in the script, they will be displayed at the bottom of the window.

Changes to scripts are saved automatically.

Find

Pressing Ctrl(Cmd) + F will open the find bar in the script editor.

This allows you to search for text within your script.

Aa	If selected, the case must match
|Abc|	If selected, the whole word must match
.*	If selected, Regular Expressions can be used in the search box

Triggers

A Trigger is an event that the controller receives which can then be used to tell the controller to do something. It is the IF part of an IF THEN statement.

Example:


```
IF (Real Time is 10:00:00) THEN (Start Timeline 1)
```

The Real Time Trigger will fire whenever the built in Real Time clock tells the controller that it is 10:00:00, and the controller will then Start Timeline 1.

There are many different trigger types available, each linking to a different internal or external triggering situation.

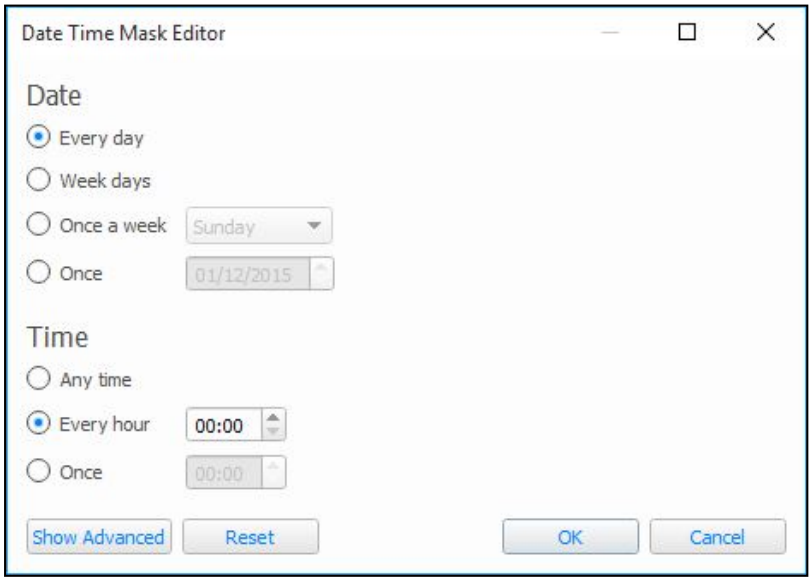
Clock/Calendar Triggers

Clock/calendar triggers use the controller's real time clock to fire triggers based on the current time, or astronomical or lunar events such as Sunset or the Full Moon. These triggers are often used when a schedule is required for the project. This could be as simple as starting a light show at sunset and stopping it at sunrise.

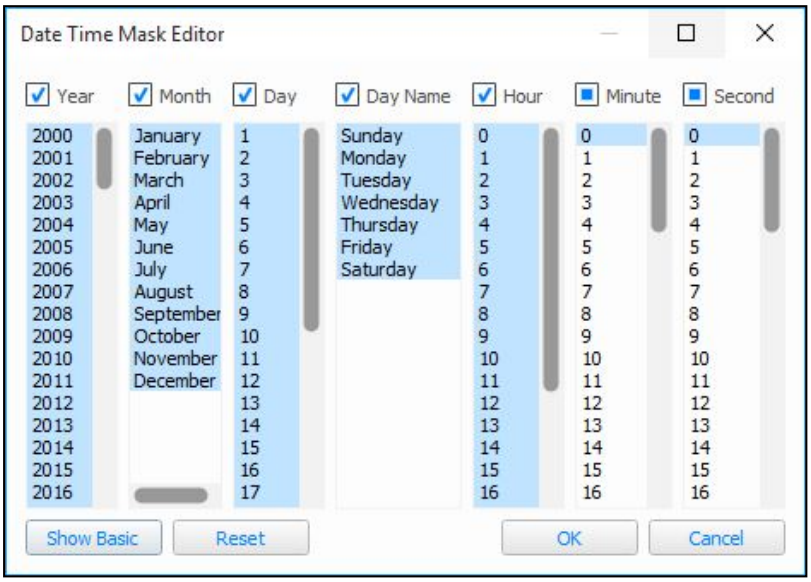
Real Time

The Controller has an internal, battery-backed real time clock. In a project with multiple Controllers only one Controller is set as the Network Primary (see Controller association), use the configuration settings to determine what sort of real time event will be matched, for example 5 minutes past every hour or at noon on a specific date.

The standard dialog allows you to deal with the most common cases, including one-off events or events that recur hourly, daily or weekly. Note that the maximum resolution of real time events is 1 second, so an "Any Time" trigger will fire every second during the specified date range:



There is also an advanced dialog that allows you to specify a precise mask of when the trigger should fire, using a combination of year, month, day of the month, day of the week, hour, minute or second. Highlighted values are included in the mask and make sure all values are highlighted in any column you don't care about. The trigger will fire at all times that match the specified mask in all columns - so no column should be blank or the trigger will never match:



Further information about the use of the real time clock can be found in the conditions chapter.

In the Network view, a Controller properties option exists to "execute real time triggers on startup". This will ensure that all real time triggers are executed from a user-specified time to the current time to reinstate the correct playback state in case the Controller is restarted for some reason (e.g. power loss, watchdog or remote reset).

NOTE: Real Time triggers are only tested by the Network Primary and then shared over the network so any conditions are tested on the Network Primary only.

Astronomical

The Controller is also equipped with astronomical clock algorithms which automatically generate the correct sunrise, sunset, dawn and dusk times for the location of the installation (see project properties). Use the configuration pane to select between sunrise, sunset, dawn or dusk and to specify an offset, negative or positive, in minutes. A negative offset will be the specified number of minutes earlier, and a positive offset will be later.

Two versions of dawn and dusk are offered, using the two definitions of twilight: *civil* and *nautical*. Please see Wikipedia for an explanation of these terms.

A Controller properties option exists to ensure that all astronomical triggers are executed from a user-specified time to the current time to reinstate the correct playback state in case the Controller is restarted for some reason (e.g. power loss, watchdog or remote reset).

NOTE: Astronomical triggers are only tested by the Network Primary and then shared over the network so any conditions are tested on the Network Primary only.

Lunar

As well as astronomical triggers the Controller uses lunar clock algorithms to calculate the lunar phases based on the location of the Controller (see project properties).

The lunar events are new moon, first quarter, full moon and third quarter. Use the configuration pane on the right to select the phase.

NOTE: Lunar triggers are only tested by the Network Primary and then shared over the network so any conditions are tested on the Network Primary only.

Playback Triggers

A playback trigger is fired by an event involving a timeline or scene, or by the controller booting. These could be used to start a particular timeline when the controller boots, or to always start a timeline when another has finished.

Startup

The startup trigger determines what the Controller should do after power up or reset. There are no configuration options.

Timeline Started

A timeline starting (generally as a result of a trigger or the timeline looping) can be used as a trigger. Use the configuration pane to select which timeline, or select 'Any' and choose a playback group to trigger when any timeline in that group is started.

Timeline Ended

A timeline reaching the end of its programming can be used as a trigger. Use the configuration pane to select which timeline, or select 'Any' and choose a playback group to trigger when any timeline in that group ends. For a looping timeline, this trigger will fire every time the timeline loops.

Timeline Released

A timeline releasing can be used as a trigger. Use the configuration pane to select which timeline, or select 'Any' and choose a playback group to trigger when any timeline in that group is released.

Timeline Flag

Any timeline can have one or more flags placed on the time bar (see working with timelines) to act as triggers. Use the configuration pane to select which timeline and the flag within that timeline, or select 'Any' and choose a playback group to trigger on any timeline flag on any timeline in that group.

Timeline can be set to any to match any flag in the project. The timeline number will be captured as a variable.

Flag can be set to any (with a specified timeline) to match any flag in that timeline. The time of the flag (in milliseconds) will be captured as a variable.

Scene Started

A scene starting (generally as a result of a trigger or a timeline preset being used) can be used as a trigger, use the configuration pane to select which scene, or select 'Any' and choose a playback group to trigger when any scene in that group is started.

Scene Released

A scene releasing (generally as a result of a trigger or a timeline preset being used) can be used as a trigger, use the configuration pane to select which scene, or select 'Any' and choose a playback group to trigger when any scene in that group is released.

Interactive Triggers

Interactive Triggers are triggers which respond to an input from a human (generally). These include push buttons on the Pharos BPS and Touch Device and other Touch Device controls. You would use these triggers if you have a Touch Device in your project and need to act upon interaction with the controls on the interface.

Soft Trigger

This trigger type is provided for triggering from the web interface, there are no configuration options.

Digital Input

The LPC and TPC with EXT have 8 digital inputs which can be used as triggers, either to detect a voltage or a contact closure driving with an "active high" signal. More information on High and Low for Contact Closure can be found here. Use the configuration pane to select which Controller (Any or a particular LPC or TPC with EXT), which Input (1 through 8) and the polarity of the logic - select Low for contact closure or when driving with an "active low" signal, select High for driving with an "active high" signal.

The Input can also be set to Low Held or Low Repeat. These will use the Held Timeout and Repeat Interval settings to fire the trigger once the Input has been low for the Held Timeout or every Repeat Interval after the Input goes Low.

The Clicked option can be used to fire the trigger when it changes to Low and back to High. The Clicked event will only occur if the input is toggled before the Held timer occurs, which can be set independently for each Remote Device in the Network view. More information can be found on the Held timer here.

To receive a digital input from a Pharos Remote Device, change the Device to RIO 80 or RIO 44 and set the RIO number, or leave as *Any*. The RIO 80 has 8 inputs and the RIO 44 has 4 inputs.

The inputs on the LPC and TPC with EXT hardware and the RIOs can also be configured as analog inputs in the Network Mode.

Analog Input

The revised LPC and TPC with EXT hardware has 8 inputs that can be configured as digital or analog inputs in the Interfaces tab of the Network view. The RIO 80 and RIO 44 have inputs that can be configured as digital or analog inputs in the Remote Devices tab of the Network view.

Use the Controller and Input settings to specify which Controller's analog input should be considered the input source. Alternatively, leave the Input set to *Any* to match any of the inputs of the Controller and to

capture the input as a variable. To use a RIO's input as the input source, change the Device from Local and select the RIO number, or leave this as *Any*.

Now you should specify the range of voltage to trigger on. You can choose whether to trigger every time the voltage changes within the specified range ("Changes in range"), or to only trigger when the voltage enters the specified range ("Enters range"). "Enters range" is generally more useful when you are using analog inputs to trigger timelines, but "Changes in range" would be required if you were using an analog input as a variable passed to a Set Intensity action to control the intensity for a group.

The voltage range of a Controller's or RIO's analog input can be configured in the Network view. The smallest measurable voltage change is 0.25V.

BPS Button Event

The BPS has eight buttons which can be used as triggers.

Use the configuration pane to select which Controller should process the trigger. Select the BPS, button number (or leave as *Any* - see variables) and the type of button event (Press, Held, Repeat, Release, Clicked). Setting the button number to *Any* will capture the pressed button as a variable.

Touch Device Triggers

Touch device triggers will respond to any touch device in the project. To respond only to a particular touch device, use the 'Touch Device' field to set the type and number of the touch device you're interested in. To specify a particular TPC, enable the 'Trigger Controller Edit' project feature, set the trigger's controller to your TPC and set 'Touch Device' to 'Local'.

Touch Button Event

Whenever a button in a Touch Device user interface is touched, triggers of this type will be checked for a match.

The Button field should be set to the Control Key of the button you're interested in - this is a property of buttons that is set in Interface. Either pick a control key from the list, or type it in.

The Event defaults to 'Click', which is a complete press and release touch action. Other options are Press, Release, Held and Repeat, like the BPS Button trigger.

Touch Slider Move

Whenever a slider in a Touch Device user interface is moved, triggers of this type will be checked for a match.

The Slider field should be set to the Control Key of the slider you're interested in - this is a property of sliders that is set in Interface. Either pick a control key from the list, or type it in.

Touch Colour Change

Whenever a colour picker in a Touch Device user interface is touched, triggers of this type will be checked for a match.

The Picker field should be set to the Control Key of the colour picker you're interested in - this is a property of colour pickers that is set in Interface. Either pick a control key from the list, or type it in.

Touch Page Change

Whenever the current page of a touch device user interface is changed, triggers of this type will be checked for a match.

The 'Page' field should be set to the name of the page you're interested in - this is a property of pages that is set in Interface. Either pick a page name from the list, or type it in. You can also specify whether you want the trigger to fire when entering or leaving that page.

Touch Keypad Code

When the Enter key on a keypad is pressed, triggers of this type will be checked for a match.

The Keypad field should be set to the Control Key of the keypad you're interested in - this is a property of keypads that is set in Interface. Either pick a control key from the list, or type it in.

Touch Inactivity

Whenever the sleep/awake state of a touch device screen is updated, triggers of this type will be checked for a match.

Choose whether to trigger after a period of inactivity or when the Touch Device becomes active (is touched) again.

The timing for when the touch device is set to inactive can be adjusted in the Device Properties area of the Network tab.

Protocol Triggers

Protocol Triggers are generally triggers which include communication with another device using a command protocol such as Serial (RS232/485), Lighting control data (DMX/eDMX/DALI) and Ethernet (TCP/UDP). These would be used when another device is used which can communicate with one of these protocols, and they could be used to start a timeline when a particular string is received from another device, or to set the intensity of a group of fixtures based on an Audio input.

Commands

Protocol Command triggers can be used when a specific message is being send to a controller by another control system or device. These could be an ASCII string sent over Serial which should cause a timeline to start or a MIDI message from a Show control system to turn down the house lights in a performance space.

Serial Input

RS232 remains a very popular protocol for interfacing equipment and the RS232 port of a Controller or Remote Device can be configured to support most common data formats. RS485 is a more robust alternative to RS232 (better noise immunity, longer cable lengths and faster data rates) and is a widely supported protocol. A Controller or Remote Device can be configured to receive RS232 full-duplex or RS485 half-duplex in the Network view, see Controller interfaces and Remote Devices. A TPC with EXT can receive RS232 full-duplex.

To receive serial from a Controller's serial port, leave Device as Local and use the Controller setting to specify which Controller's serial port should be considered the input source.

For the old LPC Xs with 2 serial ports, the Port setting selects which of the two RS232 ports should be the input source.

Alternatively, set the Device to a RIO and select the RIO number.

Now define the string of input characters to be matched as the trigger. There are three formats in which serial strings can be entered:

Hex	A series of hexadecimal characters (0-9, a-f, A-F) where pairs of values are interpreted as a byte.
Decimal	A series of decimal characters (0-255) separated by "." characters.
ASCII	A series of ASCII characters. The special characters '\n' for new line, '\r' for carriage return, and '\t' for tab are supported.

Additionally, each byte can be replaced with a wildcard to match a range of input characters and these wildcards can even be captured as variables to determine the trigger's action.

Ethernet Input

Use the Controller setting to specify which Controller should process the Ethernet input. Select the Ethernet Source (see Controller interfaces) and press Edit to define the string of input characters to be matched as the trigger in much the same way as RS232 (see above).

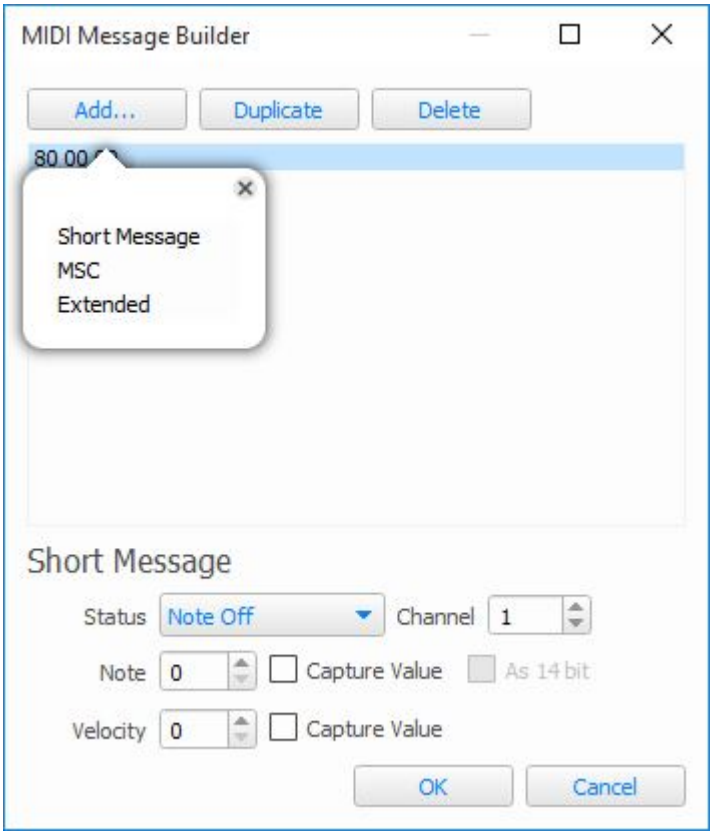
NOTE: The maximum input string size is 1.5kB. Any input larger than this will result in a second trigger being fired.

MIDI Input

MIDI is another very popular protocol for interfacing equipment and the MIDI input trigger allows you to define, via a convenient MIDI Message Builder, the type (Short message, MIDI Show Control or Extended) and command string that is to be matched as the trigger. Variables can be captured to determine the trigger's action.

Use the Controller setting to specify which LPC's MIDI port should be considered the input source. To use the MIDI port on a RIO A, set the Device to RIO A and specify the RIO A number, or leave this as *Any*. In this case, the RIO number will be captured as a variable.

Press Edit to open the Message Builder:



Press Add, select one of the three message types and then the specific command and variables.

Press Delete to delete a command string.

The resulting hexadecimal string will be constructed automatically and displayed in the window for reference with question marks ("??") indicating undefined characters in MIDI Show Control (since we do not know in advance how many characters will be captured) or <c>, <d> and <x> as appropriate for Short and Extended messages.

Press Ok to finish.

A comprehensive guide to MIDI is beyond the scope of this document, see the MIDI Manufacturers Association for more details, and the manual for the equipment to be interfaced will also certainly be an invaluable reference.

Controller Online

Use this trigger if you wish to act upon the detection of a Controller.

Use the configuration pane to select the Controller's identification number (or leave as *Any*).

NOTE: This trigger is intended for use in multi-controller projects and will not be fired when the network primary controller itself receives power. Projects requiring this functionality should instead use the Startup trigger.

Controller Offline

Use this trigger if you wish to act upon the loss of a Controller.

Use the configuration pane to select the Controller's identification number (or leave as *Any*).

Remote Device Online

Use this trigger, not the Startup trigger (which will fire before the Remote Devices can be detected), if you wish to act upon the detection of a Remote Device, for example to configure it with settings other than its defaults.

Use the configuration pane to select which Controller should process the trigger and select the Remote Device's type and identification number (or leave as *Any* - see variables).

Remote Device Offline

Use this trigger if you wish to act upon the loss of a Remote Device, for example to enter a fail safe state and issue a warning.

Use the configuration pane to select which Controller should process the trigger and select the Remote Device's type and number (or leave as *Any* - see variables).

Live Video Signal

Use this trigger to act upon the presence or absence of Live video.

Use the configuration pane to select which Controller should process the trigger and select the Event (Signal lost/Signal found).

Cloud Connection State

Use this trigger if you wish to act upon the connection or disconnection of a controllers Cloud Site. The current status of a controllers connection to a Cloud Site is shown on the home page of the web interface.

DALI Triggers

DALI Triggers are specifically used to trigger based upon messages travelling on a specified DALI bus. This allows integration of a Pharos Controller with another DALI controller, so that a timeline for some DMX fixtures can be started at the same time as a DALI command is sent.

DALI Input

RIO D, RIO D4 or TPC with EXT required.

Use the trigger's *Interface* property to select which DALI interface to use for triggering.

The RIO D and TPC with EXT snoop the DALI bus and so the trigger can be set up to respond to any DALI commands:

- Command - select Direct Level (0-254), Recall Scene or Relative Level
- Address - select All, Group (0-15) or Ballast (1-64)
- Min/Max - select the level to match for Direct Level triggering
- Scene - select the scene (1-16) for Scene matching
- Type - select the type of Relative Level command to match

The RIO D, RIO D4 and EXT both recognise DALI input from Light Sensors and Occupancy Sensors that utilise Tridonic eDALI commands. When triggering from an Occupancy Sensor select which state is to be matched. When using a Light Sensor, specify what range of light level (0-254) is to be matched. See the table below for light levels:

Light Sensor Level	Lux Range
0 - 31	0.00 - 7.75
32 - 63	8.00 - 15.75
64 - 95	16.00 - 31.75
96 - 127	32.00 - 63.75
128 - 159	64.00 - 127.75
160 - 191	128.00 - 255.75
192 - 223	256.00 - 511.75
224 - 254	512.00 - 1008.00

NOTE: Light Sensor and Occupency sensor commands require the Tridonic custom DALI commands Project feature to be enabled

DALI Bus Power

RIO D, RIO D4 or TPC with EXT required.

Use this trigger if you want to act upon a change of the electrical state of a DALI bus associated with a specific DALI interface. Buses can be in one of three states: Correct Power, Incorrect Power and No Power.

NOTE: RIO D4 cannot detect incorrect bus power and selecting this option on an interface associated with a RIO D4 will present a project issue.

DALI Ballast Error

RIO D, RIO D4 or TPC with EXT required.

Use this to trigger from a DALI ballast reporting an error. Specify the interface then use All to match if any ballast on that interface reports an error. Alternatively select a single address to match to. Next select the error type to match to.

DALI Button Event

RIO D4 required.

Use this trigger if you want to act when a button event is received by a DALI interface.

Select the interface on which to listen for the event and the event source. The event source can be:

- Instance
- Device
- Device/Instance
- Device Group
- Instance Group

The trigger can act on Any event, or choose from:

- Button Pressed
- Button Released
- Short Press
- Double Press
- Long Press Start
- Long Press Stop
- Button Free
- Button Stuck

NOTE: Not all button stations send all events, and some stations may need configuration to send certain events. Consult the manufacturer's documentation for more information.

The trigger captures five variables:

1. Short address
2. Device group
3. Instance number
4. Instance group
5. Event type

DALI Illuminance

RIO D4 required.

Use this trigger if you want to act when an illuminance report is received by a DALI interface.

Select the interface on which to listen for the event and the event source. The event source can be:

- Instance
- Device
- Device/Instance
- Device Group
- Instance Group

Select the minimum and maximum Level within which the trigger should act.

The trigger captures five variables:

1. Short address
2. Device group
3. Instance number
4. Instance group
5. Illuminance value (0-1023)

DALI Occupancy

RIO D4 required.

Use this trigger if you want to act when an occupancy event is received by a DALI interface.

Select the interface on which to listen for the event and the event source. The event source can be:

- Instance
- Device
- Device/Instance
- Device Group
- Instance Group

Select the type of occupancy to look for: Any, Vacant or Occupied.

Select the type of movement to look for: Any, No Movement or Movement.

The trigger captures six variables:

1. Short address
2. Device group
3. Instance number
4. Instance group
5. Occupancy type
6. Movement type

Dynamic

Dynamic Triggers tend to receive a value which can be anywhere within a range (e.g. DMX 0-255). These inputs generally have the Changes in Range event and Enters range event, so that a trigger can be fired whenever the input changes or only when it crosses a threshold. This could be used to set the intensity of some fixtures whenever a DMX input changes, or start a timeline when a sensor connected as an analog input passes a threshold (e.g. wind speed)

DMX Input

LPC and LPC X rev 1 can receive DMX directly. TPC, LPC X rev 2 and LPC X S3 can only receive DMX-In via Art-Net and sACN.

Use the Controller setting to specify which controller should receive the DMX.

Now you should specify which DMX channel to look at and the range of values to trigger on. You can choose whether to trigger every time the value changes within the specified range ("Changes in range"), or to only trigger when the value enters the specified range ("Enters range"). "Enters range" is generally more useful when you are using DMX to trigger timelines, but "Changes in range" would be required if you were using a DMX channel as a variable passed to a Set Intensity action to control the intensity for a group.

DMX Input State

LPC and LPC X rev 1 can receive DMX directly. TPC, LPC X rev 2 and LPC X S3 can only receive DMX-In via Art-Net and sACN.

Use the Controller setting to specify which controller should receive the DMX.

The Event state defines when the trigger should be fired.

The Input Lost event will be fired when the controller detects that it is no longer receiving DMX on the configured input.

The Input Detected event will be fired when the controller detects that it is receiving DMX data after not receiving it previously.

Audio Input

The RIO A has a stereo balanced line level audio input that can be used as a trigger.

To trigger from a RIO A, select the number of the RIO A, or leave this set to *Any* to cause the trigger to attempt to match against audio input from any RIO A. In this case, the RIO number will be captured as a variable.

Use the Channel setting to specify whether the trigger should match against the left or right audio channel, or the combination of the two. Now select which frequency band to use, or leave this set to the overall volume of the channel. Each RIO A can analyse incoming audio as up to 30 frequency bands - see Remote Devices.

The Peak checkbox tells the trigger to match on the decaying level of the last peak in the audio frequency band.

Finally, specify the range of values to trigger on. You can choose whether to trigger every time the value changes within the specified range ("Changes in range"), or to only trigger when the value enters the specified range ("Enters range"). "Enters range" is generally more useful when you are using audio to trigger timelines, but "Changes in range" would be required if you were using an audio band as a variable passed to a Set Intensity action to control the intensity for a group.

Temperature

NOTE: This trigger will only fire on a TPC or a TPS. The TPS 5 or TPS 8 do not support this feature.

The TPC and TPS have a temperature sensor, which can be used in triggers.

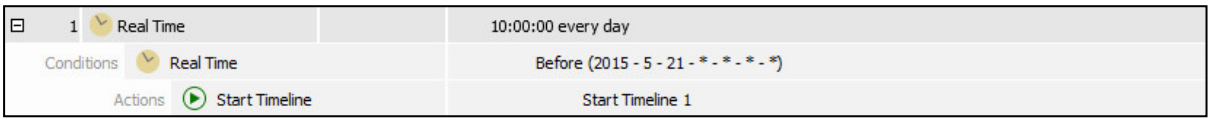
Use the Touch Device option to specify which Touch Device should be considered the input source. To specify a particular TPC, enable the 'Trigger Controller Edit' project feature, set the trigger's controller to your TPC and set 'Touch Device' to 'Local'. Select the units as Celsius or Fahrenheit, then choose how to respond to changes. You can choose whether to trigger every time the temperature changes within a specified range ("Changes in range"), or to only trigger when the temperature enters a specified range ("Enters range"). "Enters range" is generally more useful when you are using temperature changes to trigger timelines, but "Changes in range" would be required if you were using the temperature reading as a variable passed to a Set Intensity action to control the intensity for a group.

Conditions

A Condition is used to specify when a Trigger should run. In an IF Then statement, this is an AND within the IF (IF AND THEN).

For the trigger to fire, the Condition must also be met.

Example:




```
IF (Real Time is 10:00:00) AND (Real Time is Before 21/05/2015) THEN (Start Timeline 1)
```

The Real Time Trigger will fire whenever the built in Real Time clock tells the controller that it is 10:00:00 AND that the date is before 21/05/2015, and the controller will then Start Timeline 1.

Any of the Conditions can be added to any of the Triggers to narrow down when they will be fired.

You can have several triggers of the same type with the same parameters, but different conditions and actions, so that the same input can have a different Action depending on another factor (e.g. time of day)

There are various different types of Condition:

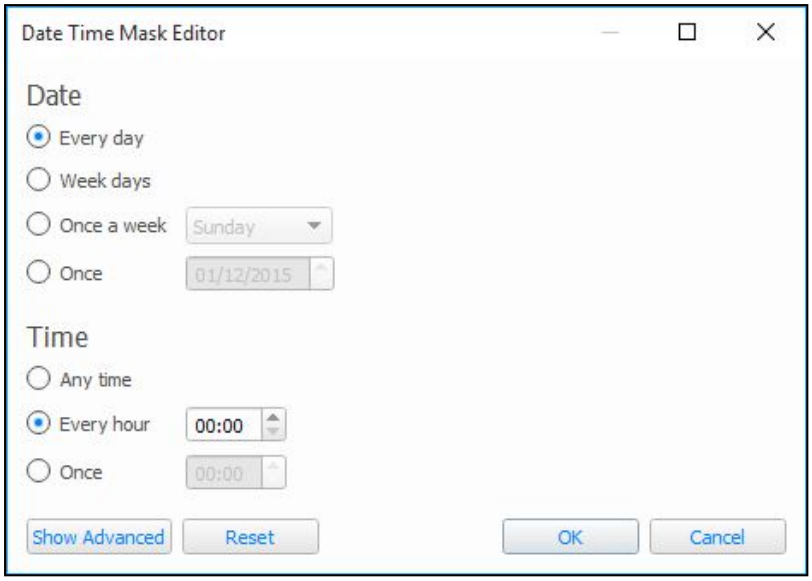
Clock/Calendar Conditions

Clock and Calendar conditions are used to specify a time/date that the trigger event must occur before or after.

Real Time

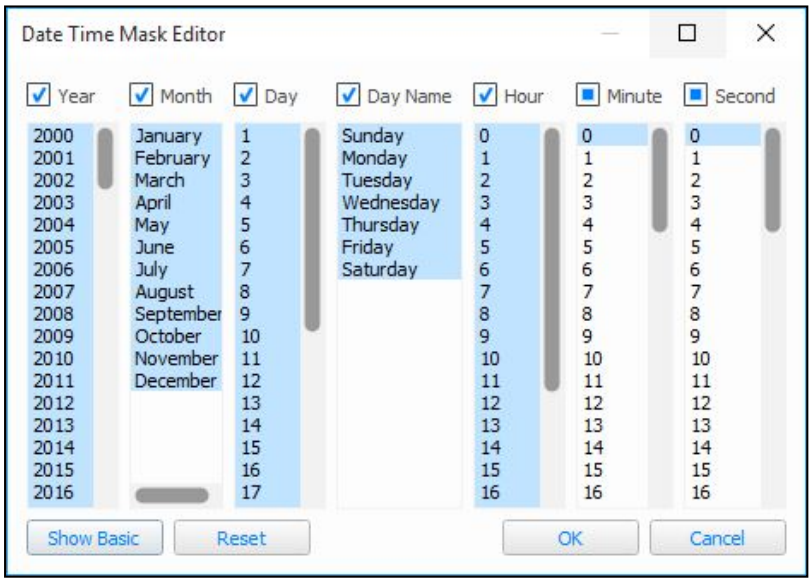
Real time conditions can be used to limit the operation of a trigger to certain times. A single condition can be set to match if the current time is before, after or equal to the time specified. Remember that the advanced dialog can be used to set a mask - this can be particularly useful with the "equal" setting for defining ranges, for example daily opening times. Where you want to specify a very specific range of times you can use two real time conditions on the same trigger, one specifying the time it must be after and the other the time it must be before, and both must match.

The conditions work by creating a mask of times, where each value of a component (year, month, day, day of week, hour, minute or second) can either be in the mask or not. When a trigger that has this condition on it is triggered, the current time will have a single value for each component. If the operator is Equals, the mask must contain those values for the condition to be satisfied:



Choosing "Every day" means all years, months and days are in the mask, so they will all satisfy the condition. Similarly, "Any Time" means all hours, minutes and seconds are in the mask, so any time will satisfy the condition. "Once a week" means only one day of the week is in the mask, so the condition is only satisfied when tested on that day of the week. Choosing a particular date or time means that only that date or time is set in the mask, so no other will satisfy the condition.

Using the Advanced mode, you can create more versatile masks:



For example, for the condition to be satisfied between 10pm and 4am, you would highlight all years, months, days, days of the week, minutes and seconds in the mask, but only set 22, 23, 0, 1, 2 and 3 in the hour mask. Thus, the condition will only be satisfied when the current hour is between 10pm and 4am.

Another example might be if you wanted the condition to be satisfied in every tenth minute on Sundays. Here, you would highlight all years, months, days, hours, and seconds in the mask, highlight only Sunday in the Day Name and highlight 0, 10, 20, 30, 40 and 50 in the minutes.

If the operator is set to Before (or After), the condition is satisfied if the current time is before (or after) the time set in the mask. If the mask contains a unique time (a single value for year, month, day, hour, minute and second), this should be easy to understand.

If the whole mask is set for a component of the date, that component is always satisfied as being Before (or After) the current time.

If the mask contains multiple (but not all) values in a component of the mask, only the first set value is taken. For example, if the day of the week component has Monday and Tuesday set, this is interpreted as being Before or After Monday.

When the operator is Before or After, the day of week is only considered if every value of the day component is set (so it will be satisfied on any day of the month).

NOTE: Conditions are always tested on the Controller that handles the trigger. Real time triggers are always handled on the Controller designated as Network Primary. But if you use real time conditions in situations where they will be tested on Controllers that are not the Network Primary then it is up to the user to make sure the time and date are set correctly on all the Controllers and not just on the Network Primary although they should synchronize automatically.

Astronomical

The Controllers are also equipped with astronomical clock algorithms which automatically generate the correct sunrise, sunset, dawn and dusk times for the location of the installation (see project properties).

Astronomical conditions can be used to limit the operation of a trigger to daytime or night time by selecting between dawn & dusk, sunrise & sunset, sunset & sunrise, etc. You can also specify offsets, negative or positive, in minutes as required.

There is also the option to select real time instead of sunset, sunrise or twilight and enter a time to create a hybrid condition such as "Between dusk and 01:00" or to create real time "between" conditioning which spans midnight.

Two versions of dawn and dusk are offered, using the two definitions of twilight: *civil* and *nautical*. Please see Wikipedia for an explanation of these terms.

Lunar

As well as astronomical triggers the Controller uses lunar clock algorithms to calculate the lunar phases based on the location of the Controller (see project properties).

Lunar conditions can be used to limit the operation of a trigger to specified lunar phases by selecting between new moon & full moon, first quarter & third quarter, etc.

Playback Conditions

Playback conditions are used to check whether a playback object (timeline or scene) is currently in use, or more complicated conditioning based within the Lua scripting environment

Timeline running

Use this condition to determine if a timeline is currently running. Running is defined as being between the start and the end of the timeline - so a timeline holding at end is not running. This condition can be useful if you want to only start a timeline if it is not already running. Sometimes timelines are used as timers and this condition is used to determine if the timer has expired. Use the configuration pane to select which timeline, or select 'Any' and choose a playback group to trigger when any timeline in that group is running.

Timeline onstage

Use this condition to determine if a timeline is currently affecting the output of the Controller. It will be true as long as one fixture patched to this Controller is being controlled by the timeline. It does not matter whether the timeline is running or holding at end. Use the configuration pane to select which timeline, or select 'Any' and choose a playback group to trigger when any timeline in that group is onstage.

Note that, unlike the timeline running condition, its result may vary between different Controllers in a network system because it depends on whether fixtures in the timeline are locally patched.

Scene onstage

Use this condition to determine if a scene is currently affecting the output of the controller. It will be true as long as one fixture patched to this controller is being controlled by the scene. Use the configuration pane to select which scene, or select 'Any' and choose a playback group to trigger when any scene in that group is onstage.

Script

Use this to run a Lua script where the returned value determines whether the condition is true or not. The script selection drop-down defaults to Quick Script. Using Quick Script, you can provide a single Lua statement which can be negated by checking the box labelled 'Negate'. To use a longer script, use the script selection drop-down to select an existing script from the project, or click 'New Script' to open the script editing dialog.

The Pharos Controllers support a scripting language that can be used for handling complicated conditional triggering or other advanced control requirements. The user can write scripts and set them to run in response to any trigger event. From within a script you can do all the things that you can do with a trigger in the triggers screen – access passed-in variables, test conditions and perform actions - but you can also define more complicated conditional statements and perform mathematical operations.

Example Scripts are available in this help file.

WARNING: Scripts are an advanced feature intended to solve problems that cannot be addressed in any other way. They are not as user-friendly as the normal triggers interface and incorrectly written scripts will not work as intended and could cause other problems with the operation of your Controller. For help with writing scripts, please see the Trigger Script Programming Guide, or please contact support to discuss requirements for a particular project.

Interactive Conditions

Interactive conditions are used to require that a certain interactive element is in a certain state for the trigger to fire e.g. a digitial input being high, or a BPS button being pressed.

Digital input

You can specify a condition based on the current state of an LPC's, TPC with EXT's or Remote Device's digital input. Leave the Device as Local to check an LPC's or TPC with EXT's input, or choose a RIO. Select the input number and whether it is active high or low (select low for contact closure).

Note that if you have more than one RIO of the same type with the same address then the condition will check against the most recent event received.

Digital inputs on Controllers or Remote Devices can also be used to detect contact closures.

Digital word

This condition allows you to test multiple of the digital inputs as a single condition. By clicking repeatedly on the numbers representing each input you can specify whether it has to be low, high or either (the default) to match.

As a side-effect the condition will also capture as a variable the state of all inputs set to match as a binary number. This can be useful if you want to pass a lot of information (such as a timeline number) using a set of digital inputs. When building the binary number low (or contact closed) is treated as a one and high (or contact open) is treated as a zero and input 1 is the least significant bit (LSB) and input 8 is the most significant bit (MSB).

BPS Button

You can specify a condition based on the current state of a button. Select the BPS and button number and whether it is pressed (Down) or not (Up).

Touch Device Conditions

Touch device conditions will respond to any touch device in the project. To respond only to a particular touch device, use the 'Touch Device' field to set the type and number of the touch device you're interested in. To specify a particular TPC, enable the 'Trigger Controller Edit' project feature, set the parent trigger's 'Test conditions on' field to your TPC and set 'Touch Device' to 'Local'. You can leave 'Test conditions on' set to 'Local' if your trigger's controller is a TPC.

Touch Button Pressed

You can specify a condition based on the current state of a maintained touch button. The Button field should be set to the Control Key of the button you're interested in. The condition checks whether the button is pressed and can optionally be negated to check the inverse.

Protocol Conditions

Protocol conditions are generally based around lighting or Pharos protocols, but also include analog inputs. These could be use to specify that eDMX Pass-Through can only be enabled if there is an eDMX signal to pass-through, or to only attempt to send DALI commands if the correct power is applied to the DALI Bus.

Status

Protocol Status conditions are specifically based on the current status of a protocol (Pharos or Lighting).

Controller Online

Use this condition to determine if a Controller is online (or offline with "Negate" checked). Select the Controller's identification number from the configuration pane.

Remote Device Online

Use this condition to determine if a Remote Device is online (or offline with "NOT" checked), select the type (RIO or BPS) and its number.

eDMX Pass-Through Detected

Use this to test if a valid eDMX source is detected on the specified port. See patch for more information.

Live Video Signal Detected

Use this to test whether a valid Live Video Signal is being received.

Output Enabled

Use this to test whether a specified output protocol is enabled.

Cloud Site is Disconnected

Use this to test if the controller is disconnected from a Cloud Site or not. The current status of a controllers connection to a Cloud Site is shown on the home page of the web interface.

DALI Status

DALI Bus Power

Use this condition to determine the electrical state of a specific DALI bus.

DALI Ballast Errors

Use this condition to determine if any or a single ballast(s) have reported a fixture error.

Dynamic

Dynamic Protocol conditions are specifically based on the current level of an input.

Analog Input

The revised LPC hardware and TPC with EXT have 8 inputs that can be configured as digital or analog inputs in the Interfaces tab of the Network view. The RIO 80 and RIO 44 also have configurable inputs, see Remote Devices.

You can specify a condition based on the current state of an analog input. Set Device to Local to use a Controller's input or choose a RIO. Then select the input number and the percentage range of input voltage. The voltage range of an Controller's or RIO's analog input can be configured in the Network view.

DMX Input

This condition will test whether the last received value for a particular DMX channel is within the specified range.

DMX Input Detected

This condition will test whether the controller is currently receiving DMX Input.

Actions

An Action tells the controller what to do when it receives a Trigger. It is the THEN part of an IF THEN statement.

Example:


```
IF (Real Time is 10:00:00) THEN (Start Timeline 1)
```

The Real Time Trigger will fire whenever the built in Real Time clock tells the controller that it is 10:00:00, and the controller will then Start Timeline 1.

There are many different Actions that can be linked to a Trigger which can affect the output of the controller, feedback on a Pharos BPS or Touch Device or send a message to another control system, among other things.

Where included, the Controller setting specifies which controller in a Multi-controller setup should run the action

Playback Actions

Playback actions are used to directly or indirectly affect the lighting output of the controller.

Timeline Actions

Timeline Actions are used to control Timelines and their output to fixtures.

Start Timeline

Starts a timeline, use the configuration pane to select which timeline.

Release Timeline

Releases a timeline, use the configuration pane to select which timeline and an optional release time.

Toggle Timeline

Starts a timeline if it's not running, or releases the timeline if it is, use the configuration pane to select which timeline and an optional release time.

Release All

Choose which playback objects to release, with an optional release time.

- All - Release all timelines and scenes
- Timelines - Release all active timelines
- Scenes - Release all active scenes

- Off - Ignore groups
- Group - Release objects in the chosen group
- Except Group - Release objects not in the chosen group

NOTE: If any overrides are in effect, these will only be released with a Release All - All action. To clear overrides with any of the other options, use the Clear RGB action at the same time.

Pause Timeline

Pauses a timeline at its current position - effects and media will also freeze, use the configuration pane to select which timeline.

Resume Timeline

Resumes playback of a paused timeline from its current position, use the configuration pane to select which timeline.

Pause All

Pauses all timelines at their current position - effects and media will also freeze.

Resume All

Resumes playback of all paused timelines from their current positions.

Set Timeline Rate

The set timeline rate action allows the playback speed of a particular timeline to be modified on the fly. You can select which timeline you want to control or get the timeline number from a variable. The rate is specified as a percentage, where 100% is the programmed rate, 200% would be double speed, and 50% would be half speed. The rate can also be driven by a variable.

Set Timeline Position

The set timeline position action allows the playback position of a particular timeline to be modified on the fly, typically the timeline would be paused to prevent it running on by itself. You can select which timeline you want to control or get the timeline number from a variable.

The reference for the desired position can be specified as a relative position, absolute position, or flag position.

Relative position requires a percentage with 0% being the start and 100% the end of the timeline. The position can also be driven by a variable.

Absolute position requires a time in seconds. The playback position will be set as if the timeline has been running for the number of seconds specified. Where a timeline is set to loop, this means that playback positions beyond the length of the timeline are converted to a position in the timeline. For example, a 10s looping timeline set to 24s will have its playback position set to 4s.

Flag position allows the playback position of a timeline to be set to a timeline flag. Flag position cannot be used when the Timeline is selected using a variable.

Scene Actions

Scene Actions are used to control the output of Scenes.

 Start Scene

Starts a scene, use the configuration pane to select which scene.

 Release Scene

Releases a scene, use the configuration pane to select which scene and an optional release time.

 Toggle Scene

Starts a scene if it's not running, or releases the scene if it is, use the configuration pane to select which scene and an optional release time.

Other Playback Actions

These Playback actions can be used to affect the playback in other ways.

Master Intensity

Sets the intensity of a group of fixtures, use the configuration pane to select the controller, group, intensity level, fade and delay times. Because the LPCs are genuine lighting controllers as opposed to DMX framestore devices, realtime control of intensity is available at all times as it would be on a sophisticated lighting console. You can control the intensity of one or more groups of fixtures regardless of what timeline(s) they may be running.

If you select a VLC/VLC+ the intensity of the whole of the selected Content Target in the specified composition will be mastered instead of a fixture group.

You can think of each group as having its own intensity fader, which this action allows you to move between 100% (default) and 0%. You can specify which group to affect and the new position for the fader. It is sometimes useful to set the fader position (as a percentage) from a variable - this permits direct intensity mastering via an input such as serial, MIDI or DMX. The fade and delay times can also be set from variables.

The fader modifies the programmed intensity for all fixtures within the group. On startup all groups have their faders at 100%. Where multiple groups containing the same fixtures have their intensity reduced then the decrease is cumulative.

Note that if you decrease intensity for one group you can only increase it again by acting on the same group. Applying an increase intensity action to a different group will have no effect even if that group contains the same fixtures - you would be trying to move a different fader.

 Increase &  Decrease Intensity

Increases or decreases the intensity of a group of fixtures, use the configuration pane to select the, group, step size (in percent), fade and delay times. Because the LPCs are genuine lighting controllers as opposed to DMX framestore devices, realtime control of intensity is available at all times as it would be on a sophisticated lighting console. You can control the intensity of one or more groups of fixtures regardless of what timeline(s) they may be running.

If you select a VLC/VLC+ the intensity of the whole of the selected Content Target in the specified composition will be mastered instead of a fixture group.

You can think of each group as having its own intensity fader, which these actions allow you to move between 100% (default) and 0%. You can specify which group to affect and the increment by which to change the fader position. It is sometimes useful to set the step size, fade and delay times from variables.

The fader modifies the programmed intensity for all fixtures within the group. On startup all groups have their faders at 100%. Where multiple groups containing the same fixtures have their intensity reduced then the decrease is cumulative.

Note that if you decrease intensity for one group you can only increase it again by acting on the same group. Applying an increase intensity action to a different group will have no effect even if that group contains the same fixtures - you would be trying to move a different fader.

Set RGB

Use this to set a fixture or group's Intensity, Red, Green, Blue and Colour Temperature levels selectively either to a fixed value or to track a variable. The latter makes for some very interesting realtime effects when used in conjunction with 2D and Media presets. Set a fade time to introduce the change. You can also choose a fade path for the change.

To choose which levels are controlled, use the checkboxes for each parameter, e.g. if you only select Intensity, then only the Intensity level will be set. This allows you to have a single slider for each colour on the TPC (for example).

If you select a VLC/VLC+ colour and intensity will be mastered for the whole of the selected content target.

Clear RGB

Use this to clear one or all fixture IRGB overrides (see above), set a fade time to release the change (s).

If you select a VLC/VLC+ the intensity will be mastered for the whole of the selected content target.

Set Text Slot

The Set Text Slot trigger action allows you to change the value of a text slot from a trigger, see the Text Preset.

You select the slot either by picking from the Text Slot list or by specifying a variable. If you use a variable, the variable must have captured a string in the trigger, and that string must be the name of an existing text slot.

The value to put in the text slot is then selected with the second variable.

Set Timecode

Use this to set the timecode position for one of the six Time Sources (set the appropriate timecode format).

Set Volume

Use this Action to set the output volume of the controller (used for Timeline Audio Output)

Transition Target

Use this action to change the position or rotation of a Target (Primary, Secondary or Additional Content Targets or Adjustment Targets).

Select the Controller, Target Type, Composition/Adjustment, Content Target Type (Primary, Secondary, Target 3-8) and Property (Rotation, X Position or Y Position) and set the relevant values.

Count specifies how many times to make the change , Period determines how long the change should take and Delay determines the pause before making the change.

Set Content Target Blur

Use this action to set a blur on one or more content targets within the project.

Use the blur radius to affect the level of blurring applied to the target, all content output to the target will then be blurred with this setting.

You can cause the blur to fade in using the fade and delay settings.

Interactive Actions

These Interactive actions are used to cause feedback to be displayed on an Interface, such as the Touch Device or the BPS

Set BPS Button LED

The BPS has eight buttons each with an integral white LED.

Use the configuration pane to select the BPS, button number (which can be driven by a variable) and the desired LED behaviour. Enabling "Set all other LEDs to default" will set the other LEDs to their default values as specified in BPS properties.

Touch Device Actions

Touch device actions will apply to all touch devices in the project. To apply only to a particular touch device, use the 'Touch Device' field to set the type and number of the touch device you're interested in. When used in conjunction with a Touch Device Trigger, selecting 'Incoming Device' will apply the action to the device the parent Trigger originated from. To specify a particular TPC, enable the 'Trigger Controller Edit' project feature, set the action's controller to your TPC and set 'Touch Device' to 'Local'.

Set Touch Control Value

Use this action to show feedback on Touch Device controls by changing their current value(s). Currently the Slider and Colour Picker controls support this.

Set the Control field to the target control key, or use the variable injection syntax to make this action work for several controls with similar control keys - the syntax is the same as for the Serial and Ethernet Output action.

Set the Index field to the index of the value that should be changed. For a Slider, this should always be 1, but for a Colour Picker it could be 1, 2 or 3 to set red, green or blue. The index can alternatively be set from a variable. Finally choose the value to set, or elect to set this from a variable.

Set Touch Control State

Use this action to show feedback on Touch Device controls by changing their appearance. The theme applied to an Interface contains various 'states' for each control type. This action lets you change the active state for a control.

Set the Control field to the target control key, or use the variable injection syntax to make this action work for several controls with similar control keys - the syntax is the same as for the Serial and Ethernet Output action. It is also possible to use the wildcard character to change multiple controls at once. For example, using "button*" would set all controls with a key that begins with "button" to the specified state.

Select the state from the drop down list or choose to set the Control to its default state.

Set Touch Control Caption

Use this action to change the caption of Touch Device controls, including Labels.

Set the Control field to the target control key, or use the variable injection syntax to make this action work for several controls with similar control keys - the syntax is the same as for the Serial and Ethernet Output action.

Finally enter the text to set as the new caption. Variables can be used in this text, using the same syntax as for the Serial and Ethernet Output action.

Set Touch Page

Use this action to change the current page shown on a Touch Device .

Set the Page field to the name of the target page, or use the variable injection syntax to make this action work for several pages with similar names - the syntax is the same as for the Serial and Ethernet Output action.

Disable Touch Device

The entire user interface of a Touch Device can be enabled or disabled. Select the Touch Device you're interested in, then choose whether to enable or disable the user interface.

Lock Touch Device

If security has been set up in the Interface then this action can be used to show the lock screen on the target Touch Device. The user must enter the correct code on the keypad in order to unlock the Touch Device.

Select the Touch Device you're interested in, then choose whether to lock or unlock the user interface.

Set Screen Brightness

The brightness of the backlight of a Touch Device may be set using this action. Select the Touch Device you're interested in, then set the value as a percentage, or elect to set this from a variable. Use the Fade Time option to specify a time, in seconds, over which the brightness should fade. The maximum time is 60s.

NOTE: The brightness of the Touch Device screen can be set automatically in response to changes in ambient light - see Controller Properties.

Set Touch Button Pressed

You can set the state of a maintained touch button. The Button field should be set to the Control Key of the button you're adjusting. Set the checkbox according to whether the button should be pressed or not.

NOTE: This action will not activate triggers related to changes in the maintained button's state.

Protocol Actions

Protocol Actions are used to communicate with other control systems or third party devices or to affect third party signals going into the controller.

Start RDM Poll

Initiates the RDM fixture polling process, refreshing the controller's record of connected RDM fixtures.

This action is only available on LPC/LPC X and TPC family controllers.

Output Serial

RS232 remains a very popular protocol for interfacing equipment and the RS232 port of a Controller or Remote Device can be configured to support most common data formats. RS485 is a more robust alternative to RS232 (better noise immunity, longer cable lengths and faster data rates) and is a widely supported protocol. A Controller or Remote Device can be configured to send RS232 full-duplex or RS485 half-duplex in the Network view, see Controller interfaces and Remote Devices. A TPC with EXT can send RS232 full-duplex.

To send serial from a Controller's serial port, use the Controller setting to specify the Controller number, leave the Device as Local and choose a port number. For the revised LPC hardware and TPC with EXT this should be set to 1. For the LPC X this should be set to 1 or 2, depending which RS232 port is being used.

Alternatively, set the Device to a RIO and select the RIO number.

Now define the string of output characters. There are three formats in which serial strings can be entered:

Hex	A series of hexadecimal characters (0-9, a-f, A-F) where pairs of values are interpreted as a byte.
Decimal	A series of decimal characters (0-255) separated by "." characters.
ASCII	A series of ASCII characters. The special characters '\n' for new line, '\r' for carriage return, and '\t' for tab are supported.

Output Ethernet

Use the Controller setting to specify which Controller should generate the Ethernet output. Define the recipient's IP address and Port, select the messaging protocol (UDP, TCP) and define the string of output characters to be transmitted. The recipients IP address and port number can be passed with variables.

When sending UDP messages it is possible to specify a bus to send from. This will force the UDP packet to be sent from the port number specified in that bus.

When sending TCP messages it is possible to specify a bus to send from. If the selected bus is of type TCP Client, the connection to the third-party device will be created before the message is sent. If the bus is of type TCP, the message will be sent using an existing connection to the third-party device, if one is available, otherwise the data will be sent from a different port to that which is specified in the bus settings.

Output MIDI

MIDI is another very popular protocol for interfacing equipment and the MIDI input trigger allows you to define, via a convenient MIDI Message Builder, the type (Short message, MIDI Show Control or Extended) and command string that is to be output.

Use the Controller setting to specify which LPC's MIDI port should be used as the output. To use the MIDI port on a RIO A, set the Device to RIO A and specify the RIO A number.

Press Edit to open the Message Builder:

Press Add, select one of the three message types and then the specific command and variables.

Press Delete to delete a command string.

The resulting hexadecimal string will be constructed automatically and displayed in the window for reference with question marks ("??") indicating undefined characters in MIDI Show Control (since we do not know in advance how many characters will be captured) or <c>, <d> and <x> as appropriate for Short and Extended messages.

Press Ok to finish.

A comprehensive guide to MIDI is beyond the scope of this document, see the MIDI Manufacturers Association for more details, and the manual for the equipment to be interfaced will also certainly be an invaluable reference.

Output Digital

The RIO 08 has eight relay outputs, the RIO 44 has four and the RIO 80 none.

Use the configuration pane to select the RIO, output and the state of the relay.

It is also possible to Toggle the output (turning it on if it is off and off when it is on)

Toggle Audio

Use this to stop a RIO A from processing audio. This can aid troubleshooting as audio activity tends to fill the log. Select RIO A and specify the RIO A number.

Toggle eDMX Pass-Through

Use this to enable or disable eDMX Pass-Through on an LPC's DMX ports. Choose which port you want to enable or disable by choosing from the port selection box. See patch for more information.

Disable Output

Use this to stop output of a selected protocol from the controller. This will prevent the controller from outputting the selected protocol, allowing another devices to take over control of the lighting, or disabling an output for test purposes.

DALI Actions

Set DALI Output

Use this to set the output on DALI fixtures patched to a specified Interface. You can specify whether you want all of the fixtures on that Interface, fixtures in a certain group or a single fixture to be affected by the change. You can also specify a fade time or choose to reuse the last fade time stored on the ballasts.

This action allows the following to be set:

- Level - The intensity of the fixture
- XY - The intensity and R,G,B values
- TC - The intensity and colour temperature
- RGBWA - The intensity and direct colour values

The Level property has two curve options available. These are 'DALI Native' and 'Linear'.

Selecting DALI Native will send the specified level directly to the ballast which then converts this to the DALI dimming curve (which is logarithmic) and outputs the level.

If Linear is selected, the controller will modify the value so that once the ballast applies the normal DALI dimming curve maths, the resultant output will be the originally supplied level.

Recall DALI Scene

Use this to recall a DALI scene on DALI fixtures patched to a specified Interface. You can specify whether you want all of the fixtures on that Interface, fixtures in a certain group or a single fixture to be affected by the scene change. You can also specify a fade time or choose to reuse the last fade time stored on the ballasts.

Send DALI Command

Use this to send a DALI command to DALI fixtures patched to a specified Interface. You can specify whether you want all of the fixtures on that Interface, fixtures in a certain group or a single fixture to be affected by the command. DALI commands include off, fade up/down, step up/down, step to min/max, step down and off and step up and on. Where fading is involved, you can specify a fade rate or choose to reuse the last fade rate stored on the ballasts.

When the Tridonic custom DALI commands Project feature is enabled, Light Sensor and Motion Sensor commands can be output to emulate these devices.

NOTE: Light Sensor and Occupancy sensor commands require the Tridonic custom DALI commands Project feature to be enabled

Start DALI Emergency Test

Use this to start a Duration or Function test on all or a single DALI address(es) on the specified interface.

Stop DALI Emergency Test

Use this to stop a Duration or Function test on all or a single DALI address(es) on the specified interface.

Mark DALI Ballast Fixed

Use this to mark all or a single emergency ballast as fixed on the specified interface.

Set DALI Feedback State

RIO D4 required.

Use this action to send either a DALI Activate Feedback or DALI Stop Feedback command on the specified interface.

DALI Feedback consists of something a device can do to provide user indication, for example lighting an LED indicator.

Select the Interface, Address and Instance on which to send the command.

Use the Command property to specify whether the action should start or stop feedback.

Remote Management Actions

Remote Management actions are used to communicate with the Remote Management service that is powered by SixEye.

Remote Site Notification Action

Use this action to send a Remote Site Notification to subscribed Remote Site users. Choose between a Warning or Error notification and provide a message. Notifications will be sent to Users containing the Notification type, message, Device and Site the Notification was generated from, a timestamp generated by the Remote Site and a link to the device in the Remote Site. The current user must be logged into the Remote Site in Designer during upload.

Run Remote Site Task Action

Use this action to run a Task in the controllers current Remote Site. To select a Task ensure the current user is logged into the Remote Site in Designer and has the correct Permissions to interact with Tasks. The specified controller must also be part of the Remote Site. Once setup the device itself is responsible for running the Task, so user permissions are irrelevant.

System Actions

System actions are actions that functionally affect the controller or the project, such as running scripts.

Log Message Action

Use this action to easily log a message into the log of the Web Interface. The Level option will determine at which “log level” the message can be seen at, and all levels below. For example, a Terse level will show

when the log level is set to Terse, and every level below, but not Critical. However, a Debug message will only show in Debug.

Enqueue Trigger

Use this to fire one Trigger from within another Trigger. The second trigger will be fired once the actions in the first trigger have run. You can specify which trigger you want to fire or which variable you want to use to get the trigger number from. You can also choose whether you want test the conditions of that trigger or not.

Run Script

Use this to run a Lua script, press Launch Editor to open the script editing dialog. If you can not achieve what you want with the triggers and actions provided it is almost certain that a script can be defined to solve your problem.

The script selection drop-down defaults to Quick Script, where you can provide a short Lua script. To use a longer script, use the script selection drop-down to select an existing script from the project, or click 'New Script' to open the script editing dialog.

Pharos Controllers support a scripting language that can be used for handling complicated conditional triggering or other advanced control requirements. The user can write scripts and set them to run in response to any trigger event. From within a script you can do all the things that you can do with a trigger in the triggers screen – access passed-in variables, test conditions and perform actions - but you can also define more complicated conditional statements and perform mathematical operations.

Example Scripts are available in this help file.

WARNING: Scripts are an advanced feature intended to solve problems that cannot be addressed in any other way. They are not as user-friendly as the normal triggers interface and incorrectly written scripts will not work as intended and could cause other problems with the operation of your Controller. For help with writing scripts, please see the Trigger Script Programming Guide, or please contact support to discuss requirements for a particular project.

Hardware Reset

Use this to force the Controller(s) to perform a hard reset which is equivalent to a power cycle. Note that unlike PC based solutions there is no particular advantage or maintenance requirement to periodically reset a Controller, this action is offered purely as a method of resetting the system to a defined, start-up state.

Variables

Variables are a way of collecting numbers from inputs and using them in actions. Some examples would be:

- Receiving a MIDI note on message and using the note value as a timeline number to start.
- Using a DMX input channel to master the intensity of a group of fixtures.
- Receiving a serial command on one Controller and outputting a related serial command on another.

Unless they have been enabled through the Project Features page, the variables will not be displayed. You can enable them using the ••• Advanced Feature button.

Triggers that capture variables

Timeline Started, Timeline Ended and Timeline Released

The Timeline Started, Timeline Ended and Timeline Released triggers capture the timeline number as variable 1 if the Timeline parameter is set to *Any*.

Scene Started and Scene Released

The Scene Started and Scene Released triggers capture the timeline number as variable 1 if the Scene parameter is set to *Any*.

Timeline Flag

If the timeline and flag has been set to Any, the timeline number, flag and name are captured into variables 1, 2 and 3 respectively.

If the timeline number has been set and flag has been set to Any, the flag and name are captured into variables 1 and 2 respectively.

Digital Input

The Digital Input trigger will capture the input number if the Input parameter of the trigger is set to *Any*.

If triggering from a RIO's digital input, the trigger will capture the input number if the Input parameter of the trigger is set to *Any*. The RIO number will be captured as variable 1 and the input number as variable 2 if both these parameters are set to *Any*. If only one of these parameters is set to *Any* then the captured number will be stored as variable 1.

Analog Input

Captures the analog input as a percentage in variable 1. For example, if the input range of the Controller's analog input is set to 0-10V and the input is 4V then variable 1 will be 40%.

If triggering from a RIO's analog input, the analog input value is captured as a percentage in variable 1, then the RIO number (if set to *Any*) and the input number (if set to *Any*) in subsequent variables. If the RIO number and the input number are set to *Any* then variable 2 will be the RIO number and variable 3 will be the input number. If the RIO number is specified then variable 2 will be the input number.

Serial and Ethernet Input

Serial and Ethernet trigger data is entered as a string of data bytes, represented in either ASCII, hex or decimal form. Any single byte or group of consecutive bytes can be matched by specifying a wildcard, and the value stored as a variable. Multiple wildcards can be used and each will store into the next available variable. There are three types of wildcards supported:

NOTE: If you need to capture a < or > symbol, please enter them as follows: \< or \>. For instance, to capture 1<2, you would enter 1\<2. This would then store 1<2 as a string.

<c> or <C>	Will match any single character (or byte) and store its raw value (0-255) as the next variable. You can add a length to the wildcard to match multiple characters and treat them as a single number - so <4c> would match a 32 bit number. Maximum length = 4.
<d> or <D>	Will match a decimal character (ASCII, 0-9) and store its numeric value (0-9) as the next variable. You can add a length to the wildcard to match multiple decimal characters and treat them as a single number - so <4d> would match four decimal characters and treat them as a number from 0-9999. Maximum length = 10. Optionally, an upper limit can be applied (<3d:255>) to set the maximum value for the incoming number. This way any range can be converted to a percentage for use with actions, e.g. Set RGB.
<x> or <X>	Will match a hexadecimal character (ASCII, 0-f) and store its numeric value (0-15) as the next variable. You can add a length to the wildcard to match multiple hexadecimal characters and treat them as a single number - so <2x> would match two hexadecimal characters and treat them as a number from 0-255. Maximum length = 8.
<s> or <S>	Will capture a string of arbitrary length. To determine where the string ends, you must either: • Specify a terminator yourself. For example, the trigger <s>\n would capture everything up to (but not including) the first \n character received. A terminator cannot be another variable, it must be a literal character, so <s><d> is not a valid trigger. • Send a NULL character (0x00) to the Controller to indicate the end of the string. This NULL character is assumed and is not shown in the Designer interface. You can also say that you want to capture a string with a predetermined number of characters. For example, <4s> will capture 4 bytes and store it as a string. There is no need for a terminator in this case.

Note that if the input data does not match the wildcard type then the trigger does not match. So if you have specified the wildcard <3d> and the input is ASCII "12y" then the trigger will not match because the 3 characters were not all of the required decimal type.

When using Ethernet Inputs the last two variables in the trigger will be the IP address and the source port number of the device the message was received from.

If triggering from a RIO's serial input, the RIO number will be captured as the first variable if set to *Any*.

To capture a float, this can be achieved using <s>. Once this has been captured, it needs to run through a script to turn it back into a float. For this, we could use the following:

```
Lua

   floatVal = tonumber(get_trigger_variable(1).string)
```

DMX Input

When a DMX Input trigger matches it will implicitly store the channel value as variable 1.

MIDI Input

In short MIDI messages, you can capture data 1 and/or data 2 into a variable by checking the 'Capture' checkbox. If both are checked, data 1 is variable 1 and data 2 is variable 2. For some short messages, i.e. Pitch Wheel, the two data bits are treated as a single 14 bit value. To capture this 14 bit value, check 'Capture' for data 1 and check the '14 bit variable' checkbox.

In MSC messages, if the 'Cue number' and 'List number' are left blank, the received values will be captured in variables. Cue number is captured into variable 1 and list number into variable 2.

Extended messages support the same wildcard format as serial triggers. The only difference is that <2c> captures a 16-bit value in serial triggers and it captures a 14-bit value in MIDI triggers.

If triggering from a RIO A's MIDI input, the RIO A number will be captured as the first variable if set to *Any*.

Audio Input

When an Audio input trigger matches it will implicitly store the level for the band as variable 1.

If triggering from a RIO A, the RIO A number will be captured if set to *Any*.

DALI Input

If the trigger is set to match to 'Any' for Group or Ballast then the Group or Ballast number will be stored as variable 1.

If the trigger is using a Min to Max range then the matching number will be stored as variable 2.

DALI Ballast Error

If All is selected instead of a specific address then the address of the ballast reporting the error will be stored as variable 1.

BPS Button

If the button number is set to *Any*, the trigger captures the pressed button as variable 1.

Alternatively, if the BPS station number is set to *Any*, then the station number is captured as variable 1 and the button number as variable 2.

Touch Button Event

You can use one trigger to respond to multiple buttons by using variables - the syntax is the same as for Serial and Ethernet Input triggers, e.g. button<3d> will match a button with the control key button001 or button002, etc. and capture the number as a variable. The name of the page that the button is on will also be captured as the final variable.

Touch Slider Move

You can use one trigger to respond to multiple sliders by using variables - the syntax is the same as for Serial and Ethernet Input triggers, e.g. slider<3d> will match a slider with the control key slider001 or slider002, etc. and capture the number as a variable.

The value of the slider will be captured as a variable.

The order of captured variables will be:

1. Any captures from Key (with additional variables where required)
2. Slider Position (0-255)
3. Name of the interface page containing the slider

Touch Colour Change

You can use one trigger to respond to multiple colour pickers by using variables - the syntax is the same as for Serial and Ethernet Input triggers, e.g. colour<3d> will match a colour picker with the control key colour001 or colour002, etc. and capture the number as a variable.

The RGB values will always be captured as 3 variables.

The order of captured variables will be:

1. Any captures from Key (with additional variables where required)
2. Red level (0-255)
3. Green level (0-255)
4. Blue level (0-255)
5. Name of the interface page containing the colour picker

Touch Page Change

You can use one trigger to respond to multiple pages by using variables - the syntax is the same as for Serial and Ethernet Input triggers.

Touch Keypad Code

You can use one trigger to respond to multiple keypads by using variables - the syntax is the same as for Serial and Ethernet Input triggers, e.g. keypad<3d> will match a keypad with the control key keypad001 or keypad002, etc. and capture the number as a variable.

The code entered into the keypad will be captured as a variable.

The order of the captured variables will be:

1. Any captures from Key (with additional variables where required)
2. The entered code (as a string)
3. Name of the interface page containing the keypad

Conditions that capture variables

Digital Word

This condition will capture a variable from the inputs set to match either value. This variable will be added on the end of any variables captured by the trigger.

Conditions that use variables

Run Script

Variables can be accessed from Lua scripts.

DALI Ballast Errors

The DALI ballast address on a specific interface can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Actions that use variables

Captured variables can then be used in actions by specifying the variable index (corresponding to the order in which the variables were captured). If you have multiple actions associated with a trigger then each action can use the variables independently.

Start, Release, Toggle, Pause and Resume Timeline

Rather than selecting a timeline as a property of the action you can specify the timeline number in a variable. This is a very powerful feature when you want an external system to be able to call up any one of a large number of timelines because you do not need to define separate triggers for each timeline.

Start, Release and Toggle Scene

Rather than selecting a Scene as a property of the action you can specify the Scene number in a variable. This is a very powerful feature when you want an external system to be able to call up any one of a large number of Scenes because you do not need to define separate triggers for each Scene.

Set Timeline Rate

You can select the timeline to modify with a variable (as for Start Timeline). You can also choose to pass in the rate percentage using a variable.

Set Timeline Position

You can select the timeline to modify with a variable (as for Start Timeline). You can also choose to pass in the position percentage using a variable.

Enqueue Trigger

You can select the trigger to Enqueue with a variable.

Run Script

Variables can be accessed from Lua scripts.

Master, Increase and Decrease Intensity

The intensity level or increment can be passed in by a variable. Select the "Variable" option and then choose the variable index.

Set RGB

The target for the Set RGB can be set from a variable. First select the override type (Fixture or Group) and then set the selector to Variable and then choose the variable index.

The Red, Green and Blue values for colour can be passed in as variables. Select the "Variable" option for the colour you want to adjust and then choose the variable index. The fade time for the action can also be passed in as a variable. Select the "Variable" option and then choose the variable index.

Clear RGB

The target for the Set RGB can be set from a variable. First select the override type (Fixture or Group) and then set the selector to Variable and then choose the variable index.

The Red, Green and Blue values for a group of fixtures can be passed in as variables. Select the "Variable" option for the colour you want to adjust and then choose the variable index. The fade time for the action can also be passed in as a variable. Select the "Variable" option and then choose the variable index.

Set Text Slot

The slot that is to be set an be selected using a Variable. Set the selector to Variable and then choose the variable index. This variable should be a string that matches the identifier of the text slot.

The text to set the Text sot to can also be set from a Variable. Set the selector to Variable and then choose the variable index.

Set Volume

The level can be set from a variable.

Serial and Ethernet Output

In the same way that you can use wildcards to match data in a serial or Ethernet trigger, you can insert the value of captured variables into your serial output messages. The same wildcard types are supported to define how to output a variable:

<c>	Will output the value of a variable as a raw byte (0-255).
<d>	Will output the value of a variable as a decimal number (ASCII, 0-9).
<x>	Will output the value of a variable as a hexadecimal number (ASCII, 0-f). Any letters will be

	lowercase.
<X>	Will output the value of a variable as a hexadecimal number (ASCII, 0-f). Any letters will be uppercase.
<s>	Will output a captured string. <s> will output the entire captured string. <4s> would output the first 4 characters of the captured string.

As with input you can specify a length if you want to output the variable as a longer decimal or hexadecimal number. So a variable value of 175 output with <4d> would add ASCII "0175" to the serial output. Note that it is padded with leading zeros to fill the specified length. If the value was too large to express in the specified length it would be truncated from the left, so <2d> would output the number 123 as ASCII "23".

Output strings are allowed to begin with a wildcard. By default each wildcard takes the next variable in the order they were captured. If you want to output the variables in a different order then you can add a variable index to the wildcard in the form <3,2d> where 3 is the variable index. If you specify an output wildcard where there is no corresponding capture variable then it will have value of zero and output accordingly.

Output Digital

The Device number (not type), Output number and State can all be set from a variable.

The Output number will be in the range 1-4 or 1-8 depending on the number of outputs on the RIO.

The State will be an integer where 0 is off and any other integer is on.

MIDI Output

Short messages can output a captured value for data 1 and data 2. Pick a variable using the 'Variable Index' controls. If data 1 is outputting a captured value, you can optionally send it as a 14-bit value, with the lower 7 bits in data 1 and the upper 7 bits in data 2, by checking '14 bit variable'.

Outputting a MSC message allows you to set the cue number and/or list number by choosing a variable with the 'Variable Index' controls.

Extended messages allow you to output captured variables using the same syntax as serial actions.

DALI Set Level

The DALI group or ballast number on a specific interface can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

The level of a DALI ballast, group or all DALI ballasts can be passed in by a variable. Select the "Variable" option and then choose the variable index.

DALI Recall Scene

The DALI group or ballast number on a specific interface can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Recall a DALI scene on a specific interfaces single ballast, group or all of the ballasts. Select the "Variable" option and then choose the variable index.

DALI Command

The DALI group or ballast number on a specific interface can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Start DALI Emergency Test

The DALI ballast address on a specific interface can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Stop DALI Emergency Test

The DALI ballast address on a specific interface can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Mark DALI Ballast Fixed

The DALI ballast address on a specific interface can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Set BPS Button LED

The button number and intensity level can be passed in by variables.

Set Touch Button Pressed

The touch device number can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Set Touch Control Value

You can use the variable injection syntax to make this action work for several controls with similar control keys - the syntax is the same as for the Serial and Ethernet Output action.

You can also use the * wildcard to match any string e.g. button* would match any key starting with button.

Variables can also be used to set the index of the action and the value to set the control to.

The touch device number can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Set Touch Control State

The touch device number can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Set Touch Control Caption

You can use the variable injection syntax to make this action work for several controls with similar control keys - the syntax is the same as for the Serial and Ethernet Output action.

You can also use the * wildcard to match any string e.g. button* would match any key starting with button.

Variables can also be used in to pass the Text. The variable should contain a sting.

The touch device number can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Set Touch Page

The target page number can be passed using a variable. This should be a number.

The touch device number can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Lock Touch Device

The touch device number can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Disable Touch Device

The touch device number can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Set Screen Brightness

The level of the Screen brightness can be set using a variable. This variable should be a percentage value.

The touch device number can be passed in by variable. Select the relevant "Variable" option and then choose the variable index.

Transition Content Target

The Composition number, Property parameters, fade and delay can be set using a variable.

Set Content Target Blur

The Composition number, Blur radius, fade and delay can be set using a variable.

Notes

- For hex strings, if a wildcard is inserted after an odd number of digits, the odd digit is treated as the lower 4 bits of the byte. For example, ff1<d> will be interpreted as ff01<d>.
- If you want to match a '<' character, you must precede it with a backslash. In general, a backslash followed by any character will match that character (ignoring the backslash).
- The * wildcard will only work when used in actions, when capturing on an input (for example Touch Button Event) you should use another wildcard such as <s> or leave it blank as it will still case match.

IO Modules

An IO Module is an add-on for Pharos Designer, which can be used to add to the functionality of the triggering in the project.

The IO Module is a collection of zero or more Triggers, Conditions and Actions. These Triggers, Conditions and Actions can be used in the same way as the default, built in Triggers, Conditions and Actions.

Adding IO Modules to the project

Designer has a series of IO Module as part of the installation.

These are automatically brought into the project, but aren't added to the project. They are listed in the IO Module Library as Unused Modules.

To "Use" a module

- Select the required module
- Choose New under Instance Properties
- Set an instance name and any properties that the module requires

To Import an IO Module

Enable the IO Module Creator within the Project Features view.

Open the IO Module window within the Trigger mode.

Select Import and browse for the .iom IO Module file.

The IO Module will be imported and added to the project.

To Download an IO Module

In addition to the IO Modules packaged with Designer, there is an Online Library of IO Modules that can be downloaded and included in your project.

Choosing the Download option in the IO Module window within the Trigger Mode will open the online library.

Selecting any modules here will download them and import them into your project for use.

To Create an IO Module

Enable the IO Module Creator within the Project Features, view.

Open the IO Module window within the Trigger mode.

Select Create to create a new IO Module from the source files.

Select the package.json file and the IO Module will be created.

Writing an IO Module requires a good understanding of Lua Scripting, and the IO Module API. This API can be found here.

IO Module Instances

Each IO Module will have at least one instance.

Where configured in the module, multiple instances can be used to set up communications with multiple separate devices.

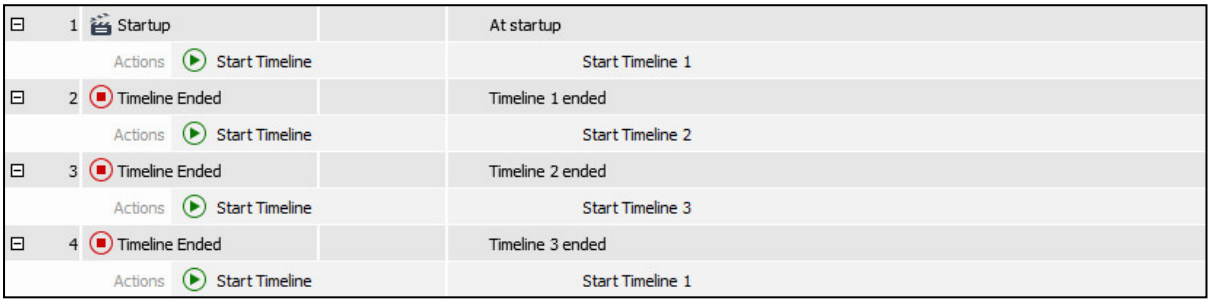
Triggers, Conditions and Actions added by the module will have a property to select the instance that should be used (utilising the instance properties).

Examples

Below are some examples of effects which can be achieved through Triggering

Timeline Looping

Through triggering, it is possible to cause a set of timelines to loop continuously.

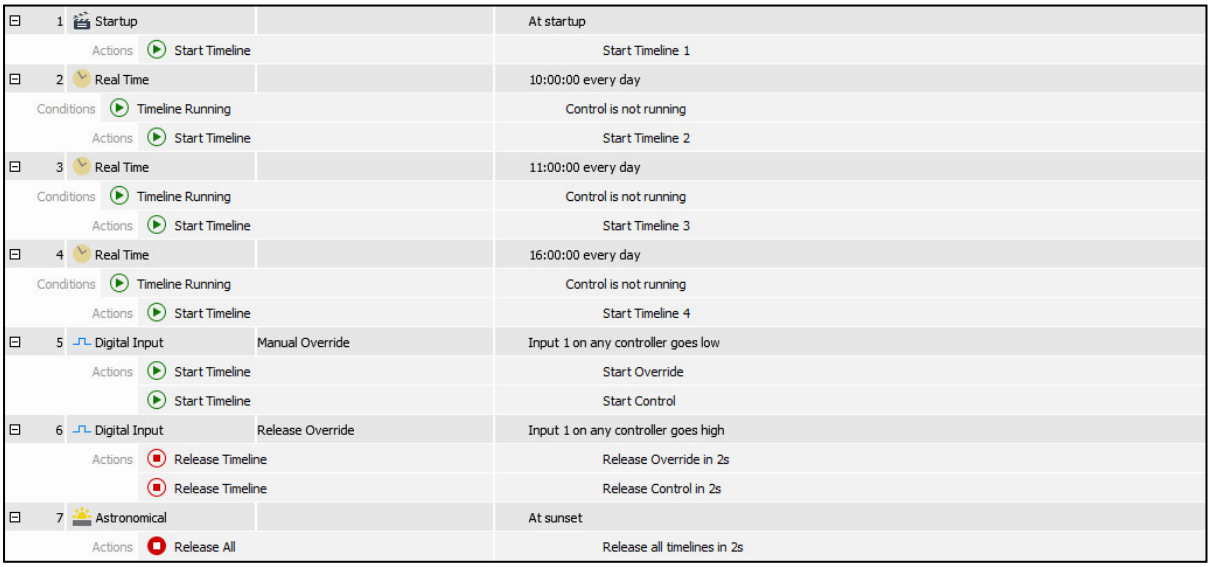


Trigger 1	At Startup Timeline 1 is started
Trigger 2	When Timeline 1 ends, start Timeline 2
Trigger 3	When Timeline 2 ends, Timeline 3 starts
Trigger 4	When Timeline 3 ends, Timeline 1 starts

This will continue and the 3 Timelines will play one after the other forever.

Lock Out Programming

Sometimes a manual override is required, e.g. using a key switch or other input. This can then also prevent the normal triggering from running.



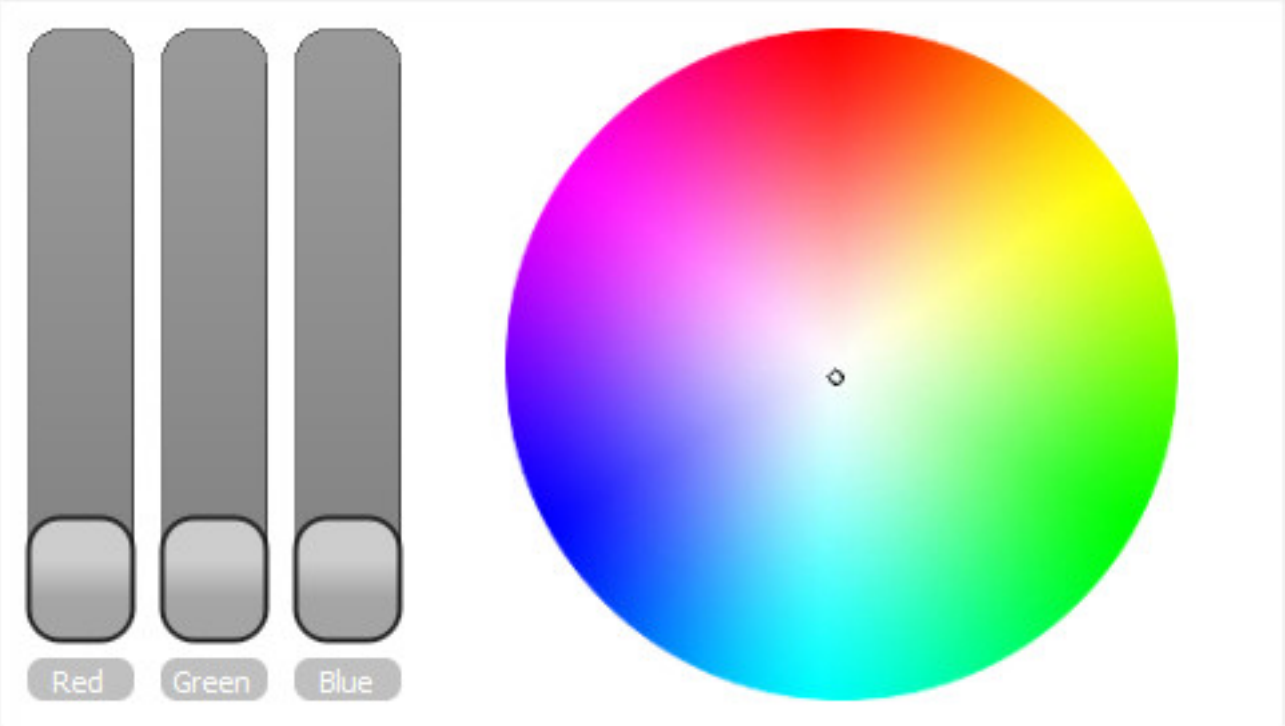
Trigger 1	A startup trigger to start a timeline
Trigger 2-4	Realtime scheduling for starting timelines, with the condition that Timeline Control is not running
Trigger 5	Manual override, starts the override timeline and the Control Timeline
Trigger 6	Manual override release, releases the override timeline and Control Timeline.
Trigger 7	Sunset trigger to release all timelines

When you start the override, you also start a control timeline (this is a timeline with a flag and a loop). The normal scheduling triggers use a timeline running condition so they can only run if the Control timeline is not running. The trigger to release the override also releases the control timeline so the normal programming can resume.

Touch Device Colour Picking and Sliders

The controls on a Touch Device can be used to override the RGB values of colour mixing fixtures. These overrides will set the levels for the fixture/group set in the action and will override other effects (depending on the Playback Override Priority)

1	Startup		At startup
	Actions	Set RGB	Set IRGBT(255,255,255,255,-) on fixture 1 in 0s using Default path
2	Touch Slider Move		Slider "slider001" is moved
	Actions	Set RGB	Set IRGBT(-,variable 1,-,-,-) on fixture 1 in 0s using Default path
		Set Touch Control Value	Set control "colour001" value 1 from variable 1
3	Touch Slider Move		Slider "slider002" is moved
	Actions	Set RGB	Set IRGBT(-,-,variable 1,-,-) on fixture 1 in 0s using Default path
		Set Touch Control Value	Set control "colour001" value 2 from variable 1
4	Touch Slider Move		Slider "slider003" is moved
	Actions	Set RGB	Set IRGBT(-,-,-,variable 1,-) on fixture 1 in 0s using Default path
		Set Touch Control Value	Set control "colour001" value 3 from variable 1
5	Touch Colour Change		Colour Picker "colour001" is changed
	Actions	Set RGB	Set IRGBT(255,variable 1,variable 2,variable 3,-) on fixture 1 in 0s using Default path
		Set Touch Control Value	Set control "slider001" value 1 from variable 1
		Set Touch Control Value	Set control "slider002" value 1 from variable 2
		Set Touch Control Value	Set control "slider003" value 1 from variable 3



Trigger 1	At Startup: • Set the fixtures to full intensity (255) and white (25,255,255) • Set the Slider positions to the top (full)
Trigger 2	When slider001 (Red) is moved:

- Set the Red property of the RGB override of Fixture 1 to the same value as the slider (variable 1)
- Set Index 1 of the colour picker (Red) to the same value as the slider (variable 1)

When slider002 (Green) is moved:

Trigger 3

- Set the Green property of the RGB override of Fixture 1 to the same value as the slider (variable 1)
- Set Index 2 of the colour picker (Green) to the same value as the slider (variable 1)

When slider003 (Blue) is moved:

Trigger 4

- Set the Blue property of the RGB override of Fixture 1 to the same value as the slider (variable 1)
- Set Index 3 of the colour picker (Blue) to the same value as the slider (variable 1)

Whenever the Colour Picker is moved:

Trigger 5

- Set the RGB override of Fixture 1 to Red = variable 1, Green = variable 2 and Blue = variable 3 (these are the RGB values coming from the colour picker)
- Set Index 1 of the three sliders to the value of the relevant component of the Colour Picker (Red = variable 1, Green = variable 2 and Blue = variable 3)

Setting Intensity from Variable input

You may want to set the intensity of a fixture or group from a dynamic input such as Analog input or Touch Slider move. Below are a series of triggers exhibiting the required structure.

1	Analog Input		Input 1 on any controller changes when in range [0%, 100%]
	Actions	Master Intensity	Set All Fixtures to variable 1
2	DMX Input		Channel 1 on any controller changes when in range [0, 255]
	Actions	Master Intensity	Set All Fixtures to variable 1
3	Audio Input		Combined Volume on RIO A 1 changes when in range [0, 255]
	Actions	Master Intensity	Set All Fixtures to variable 1
4	Touch Slider Move		Slider "slider001" is moved
	Actions	Master Intensity	Set All Fixtures to variable 1
5	Analog Input		Input 1 on any controller changes when in range [0%, 100%]
	Actions	Set RGB	Set IRGBT(variable 1,-,-,-,-) on All Fixtures in 0s using Default path
6	DMX Input		Channel 1 on any controller changes when in range [0, 255]
	Actions	Set RGB	Set IRGBT(variable 1,-,-,-,-) on All Fixtures in 0s using Default path
7	Audio Input		Combined Volume on any RIO A changes when in range [0, 255]
	Actions	Set RGB	Set IRGBT(variable 1,-,-,-,-) on All Fixtures in 0s using Default path
8	Touch Slider Move		Slider "slider001" is moved
	Actions	Set RGB	Set IRGBT(variable 1,-,-,-,-) on All Fixtures in 0s using Default path

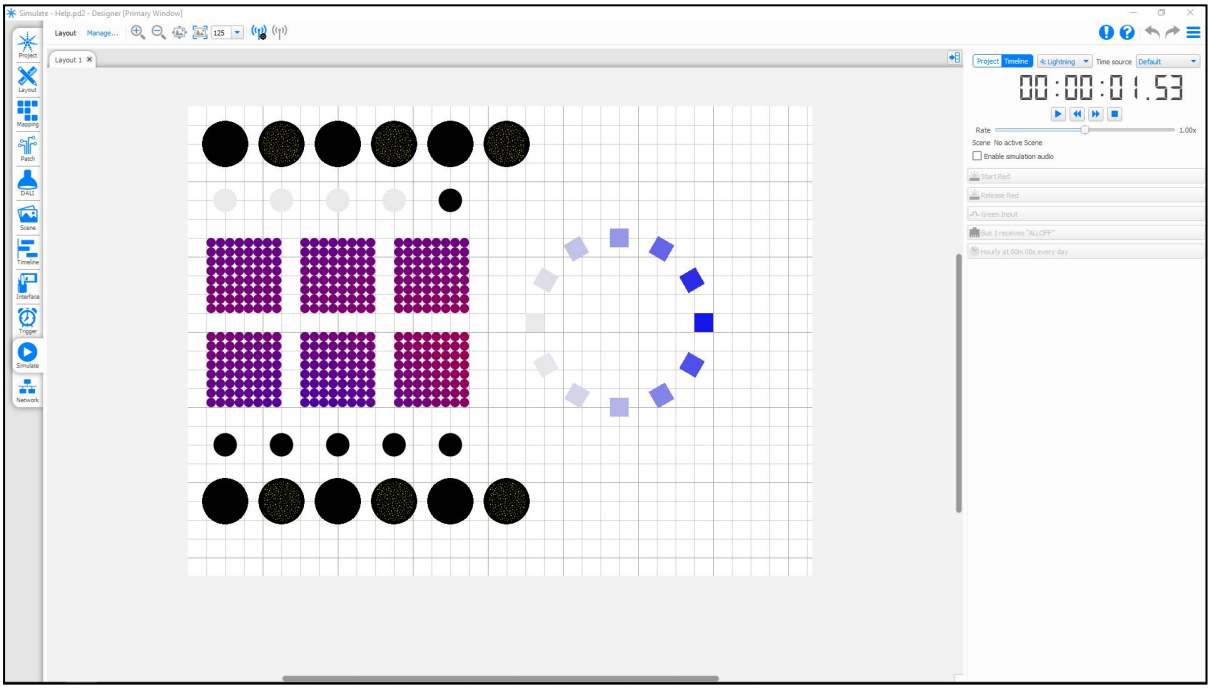
All these triggers take in a Dynamic input (variable level/volume) and this level will be captured as a variable. This can be passed to the Set RGB action. The Set RGB action has the option to set the Intensity of the fixture/group. This will override other programming, if you want to master the intensity of the group without changing the output colours/effects, you can use the Master Intensity action.

Simulate

Keyboard Shortcuts

Space	Start/Pause Simulation
Esc	Stop Simulation
Ctrl+0	Reset the zoom
Ctrl+F	Zoom to fit the window
Ctrl++	Zoom in
Ctrl+-	Zoom out
Ctrl+ mouse wheel	Zoom in and out
Middle-click + drag	Zoom into the drawn rectangle
Alt+ mouse wheel (Shift+ mouse wheel)	Scroll Horizontally

The simulator allows you to preview your programming on the Layout:



The window comprises 3 sections: The main portion of the window is your Layout. Top right are the simulator controls with time counter, and below them any programmed triggers. Note that the time counter shows simulated time not real time.

NOTE: The simulator cannot be used with VLC layouts.

Simulator modes

Timeline mode

Use this mode to simulate a single timeline, typically the one you are editing. Select the timeline and press Start to simulate, press Reset to reset the timeline.

Project mode

Use this mode to simulate the whole project and verify your trigger programming and timeline interaction. Any triggers created will appear on the right hand side and can be activated by clicking on them. Activating a trigger will automatically start the simulator clock, press Reset to reset all timelines and triggers.

Simulator controls

The Simulator Play controls are replicated on the Timeline Toolbar, and either can be used to control the simulated timeline.

Start/ Pause

Toggles between Start and Pause accordingly, the keyboard spacebar can also be used.

Skip backwards & forwards

When the simulator is running or paused these buttons skip backwards or forwards in 10 second increments.

Reset

Resets the simulator, the simulated triggers (Project mode only) and releases playback so, if Output Live is enabled, the venue will go to black (fixture defaults).

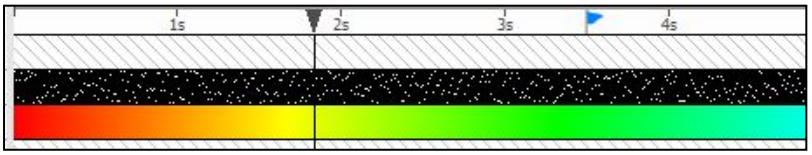
Rate Slider

Adjusts the simulator's playback speed (not the timeline's programming) from 0x (paused) > 1x (normal, default position) > 60x (fastest).

NOTE: Reset does not reset this setting so remember to set it back to 1x (normal) when you've finished.

Timeline Mode play head (Timeline mode only)

In Timeline Mode, when simulating a timeline in Timeline mode, the simulator's current playback position is marked by the play head; a black vertical line and arrow on the timeline ruler:



You can grab the play head for manual positioning by clicking on and dragging the arrow along the ruler, akin to "scratching" since you are now driving the simulator by hand at whatever speed and in whatever direction you choose, very useful for examining transitions in detail for example - easiest with the simulator paused.

You can also make the simulator jump (when running or paused) by just clicking on the ruler at the required target time, very useful for getting and keeping the simulator in an area of interest, particularly with long timelines.

Testing trigger variables

Any triggers which capture variables have a widget that allows values to be tested. These variables take the form of a comma-separated list and can either be numbers or text strings. Text strings should be bounded with double quotes.

For example, entering: 100, 50, "ABC", "100" would inject variable indices 1>4 with the values 100 (number), 50 (number), ABC (text) and 100 (text).

Testing trigger conditions

Conditions are not tested by the simulator.

Simulating timecode

By default, for any timeline that has been set to use a timecode source (see timeline properties) the simulator will emulate the timecode. If, however, a Controller is connected and receiving real timecode then the simulator can instead be made to track this real timecode by selecting Time Source from the Timecode drop-down menu.

Output Live

Output Live is not possible without associated and patched Controllers but this allows you to view the programming on the installation itself without having to upload repeatedly after every iteration. The Output Live button can also be found in the Scene view to facilitate the programming of the Scenes, particularly setting the position parameters.

Output Live completely overrides the LPC's playback engine with all calculations instead being performed within Designer.

NOTE: If your controller/s is/are password protected, you will need to login to the controller/s from the Network Mode

Output Live Mask

Using an Output Live Mask, you can choose which fixtures are output to when you use Output Live. The Output Live Mask can be used to filter which Groups, Layouts, Universes, Controllers or DALI Interfaces can have the Output Live data applied to.

Multiple Layouts

If your project includes multiple Layouts, then tabs will be available across the top to choose between them. This will include Normal Layouts and VLC Layouts.

VLC Layouts

To Simulate VLC output, your computer will need to include support for OpenCL 1.2. You can choose which computer hardware should be used for this support from the Simulate Preferences. If your computer doesn't support OpenCL 1.2 then you won't be able to simulate VLC output.

Simulation Audio

By selecting Enable simulation audio in Simulate, an additional row will be added to timelines. This allows an audio track to be dropped onto this row to be played back within Designer when the Timeline is Simulated.

To Import Audio

Audio is imported in the same way as video clips; from Mapping or the Media Preset library.

Network Overview

Pharos products are designed to operate on an Ethernet network for maximum scalability and the range can be split into two classes:

Controllers

The primary processing "brains" in a system, designed to operate as single, stand-alone units or co-operatively as a scalable system, automatically synchronised and managed over the network. All Pharos Controllers have an integrated web interface for remote management.

Controllers use TCP/IP for communication and, as a result, need to be correctly configured, see Controller connection.

Once the Controllers have been connected they are then uniquely associated with Controllers in the Designer project, see Controller association.

Touch Panel Controller (TPC)

The TPC is an advanced lighting controller with an integrated, customisable, capacitive touch screen. It outputs up to 512 channels of eDMX and is able to output multiple protocols simultaneously. See Controller properties.

TPC with EXT

When using a TPC that is connected to an EXT the TPC functions in largely the same way. The extra output and triggering options provided by the EXT will be seamlessly applied to the TPC when the Controller Type in the Network tab reads "TPC+EXT". This includes the ability to output a physical DMX universe.

Lighting Playback Controllers (LPC)

Three models are available for DMX & eDMX lighting control, the LPC 1 (512 control channels), the LPC 2 (1024 control channels) and the LPC 4 (2048 control channels). The LPC 4 can output up to 1024 channels as local DMX - the remaining channels must be output as eDMX. All 2048 channels may be output as eDMX.

The LPC X is available for eDMX and DVI control with capacities ranging from 10 universes (LPC 10; 5,120 control channels) to 100 universes (LPC 100; 51,200 control channels).

Unlike common DMX frame store devices they are extremely powerful and flexible and can be configured in numerous ways to suit the application, see Controller properties.

Video Lighting Controllers (VLC/VLC+)

The VLC is designed to handle large single canvases of DMX fixtures e.g. building facades, bridges or media screens. The VLC is available in capacities ranging from 50 DMX universes up to 1500 DMX universes from a single unit with further scaling over Ethernet.

The VLC+ is additionally available in a 3000 universe option.

Remote Devices

The Pharos range includes various Remote Devices that augment one or more Controllers with additional remote inputs, outputs and user interfaces.

Remote Devices can only be powered by the PoE network, use multicast UDP for communication and have varying configuration options, see Remote Devices.

Controller Connection

Before you can configure and upload to the Controllers they must be connected to the PC running the Designer software. Depending on the Controller, this connection can generally be made in one of two ways:

Ethernet

This is the most flexible method of connecting one or more Controllers and is well suited to a permanent installation. TCP/IP Ethernet itself needs configuration and management, in particular the setting of IP addresses.

If a controller is on an incompatible network, it will be displayed in the Network Mode with Grey text.

DHCP (default)

Pharos Controllers are factory set to receive an IP address from a DHCP server so one must be present on the network.

Link local (DHCP error)

Should the Controller fail to find a DHCP server it will assign itself a "link local" (169.254.x.x) IP address which can be used with Designer to establish a temporary connection. However, a DHCP server should be found or a Static IP address set to establish a permanent connection.

Static IP (optional)

It is sometimes desirable to set a Static IP address so that the IP address of the Controller is always known (DHCP served IP addresses can change). Refer to Controller configuration.

Multicast

Pharos Controllers and Remote Devices also use a block of Multicast addresses for "discovery" and Remote Device communication so these addresses must be available: 239.192.38.7 and 239.192.38.8

Default gateway

Must be consistently set to either nothing or a valid IP address.

Managed switches and firewalls

Managed Ethernet switches and your PC's Security Firewall can conspire to make life difficult - by blocking Multicast addresses for example. Pharos recommends the use of Unmanaged switches and disabling your PC's Firewall if you're experiencing connection problems.

eDMX considerations

While the LPC X has a second, dedicated Ethernet Protocol port with its own IP settings, the TPC and LPC 1, 2 & 4 must share their single Ethernet port. However, this single Ethernet port can be configured with two IP addresses, making it easier to manage routing of output protocol data. See Controller Protocols.

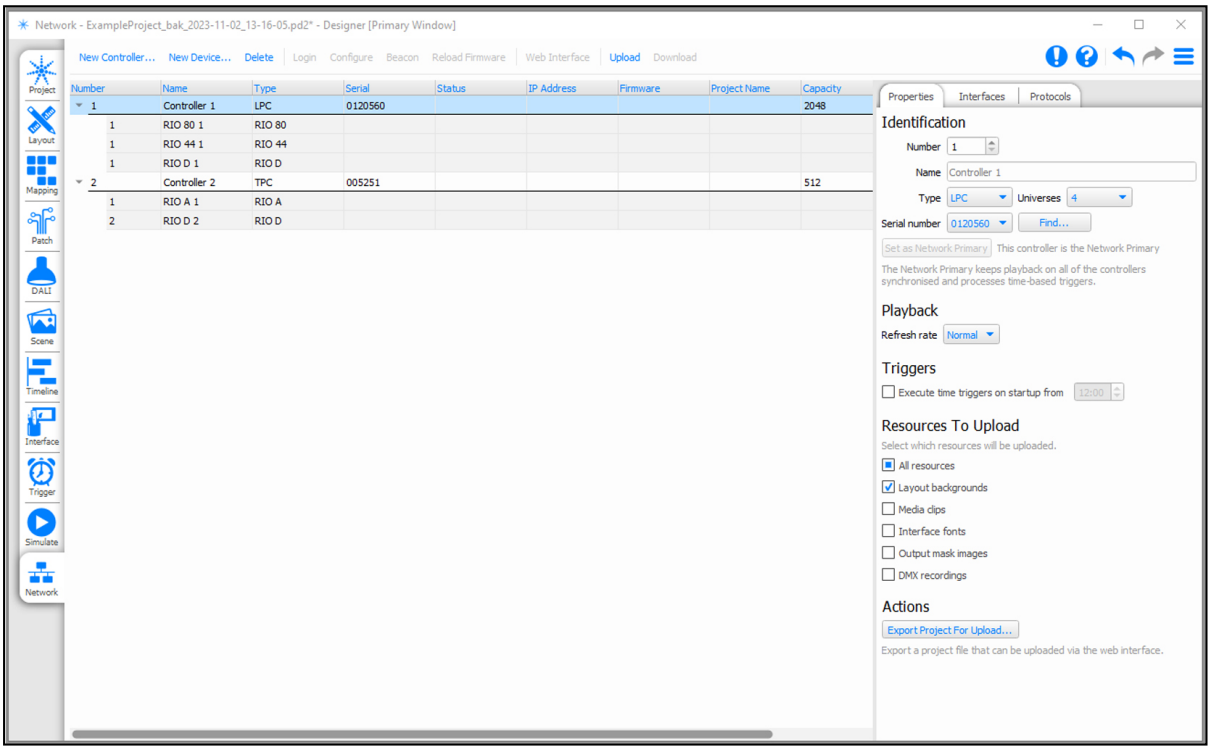
For further information about the Controller hardware and its input/output ports please refer to the Installation Guide supplied with the unit or available from the website.

Ethernet over USB

In addition to being able to connect to a controller using a traditional Ethernet connection, Designer is able to communicate with an LPC using an Ethernet over USB connection. The controller will appear in Designer as if the connection has been made using Ethernet (with the controller having an IP address) and all the same functionality is available, but the physical connection is made using a USB cable.

Network window

Once you have connected the Controller, select the Network tab to view your connected Controllers & Remote Devices:



If a controller is on a different network, and is not discovered on USB, it will appear as grey in the network spreadsheet. If a controller is on a different network (regardless of discovery on USB), a warning message is shown in the Controller Config tab.

Controller firmware

IMPORTANT: Controllers must be running the same version of firmware as the Designer software. Controllers with incompatible firmware will be highlighted in red.

NOTE: Controller's on a v1.x.x firmware version will not show up in Designer 2. See Conversion for more details.

To update a Controller's firmware:

1. Select the incompatible Controller, the row will be highlighted
2. Press Reload Firmware on the network toolbar
3. The firmware update will proceed - you must not disturb this process

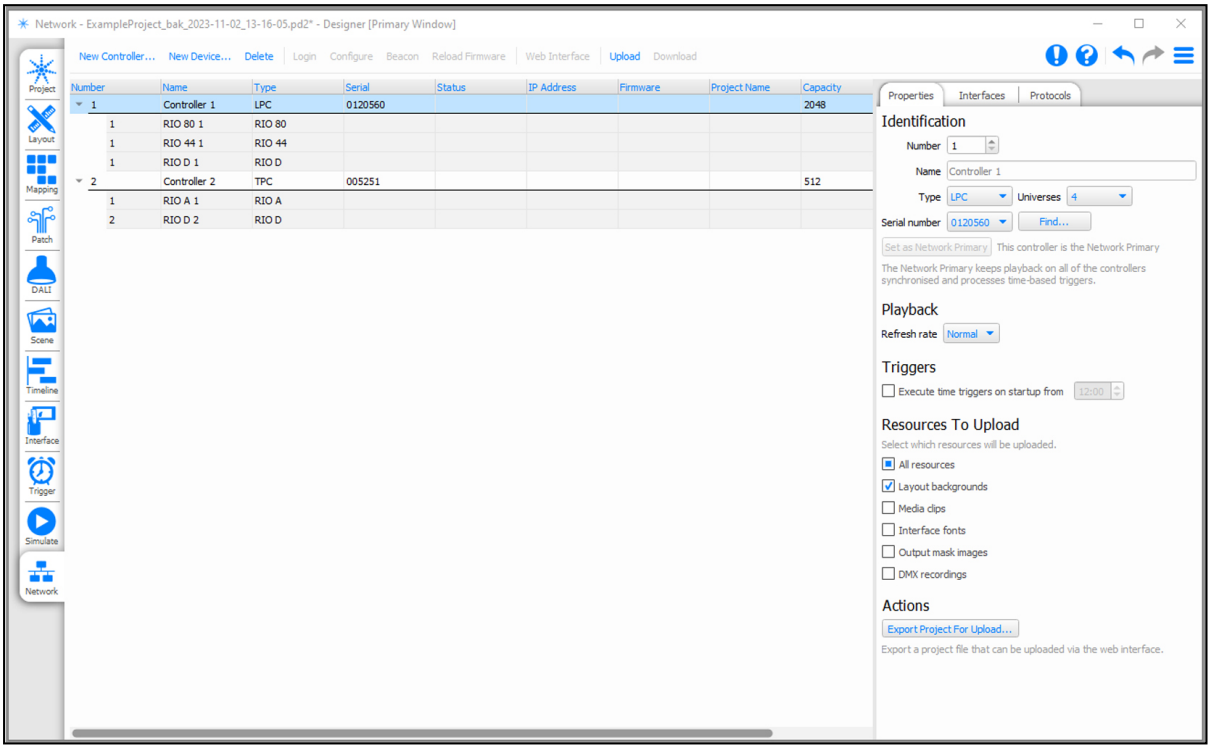
Alternatively a utility application is provided that also allows you to update the LPC X's bootloader and firmware, see LPC X Recovery Tool. There is a recovery procedure for the TPC and LPC 1/2/4.

The EXT's firmware is updated directly from the TPC. If the TPC detects that a connected EXT has the wrong firmware version then it will automatically update it - you must not disturb this process. The EXT's 'TPC Active' LED will illuminate continuously when this process has completed successfully.

Once all the connected Controllers have compatible firmware you can associate them with project Controllers and configure their hardware.

Device Association

It is here that you connect your project programming in Designer with real, networked Pharos devices:



Project vs Real Devices

The list of Devices is split into two sections: At the top is the list of project Devices which may or may not be associated with real Devices. Underneath is a list of all the unused real Devices found on the network that have not been associated with project Devices.

When you create a new project, it will have one or more Devices in it. These Project Devices are purely virtual and so must be associated with a real Controller on the network before you can Output Live or Upload.

IMPORTANT: You can only associate a Controller running the same firmware as Designer - Devices running incompatible firmware will be displayed in red. To update a connected Device's firmware see Controller firmware.

Managing project Devices

If you will use a different type of Controller for your project, such as an LPC X for a high DMX channel count, or more than one Controller if the installation is large or distributed, then you will want to modify the list of project Controllers.

To Add and Set the Type of a Project Device:

1. Press the New Controller or New Device button on the toolbar
2. Select the Device's Type from the dialog, if this is a Touch or Remote Device, select the Controller to attach it to.
3. Use the Properties pane to give the Device a useful name, perhaps describing where in the installation it is or what it controls
4. If appropriate, select the number of universes that the Controller supports

The Device has now been added to the project.

If the project has a TPC with an EXT then the EXT will be configured automatically by the TPC. You can add an EXT to a TPC in the project by checking the 'Configure EXT' check box in the TPC's Interfaces tab.

Alternatively a physical controller can be added to the project by Right-clicking and selecting Add to Project.

To Delete a Project Device:

1. Select the project controller, the row will highlight
2. Press Delete on the mode toolbar

IMPORTANT: Deleting a project controller that has been patched will result in the loss of this patch data.

Managing Replicated Projects

An Install Replication is a set of settings that allow you to simply upload the same project to multiple sets of devices.

To Create an Install Replication

- Enable Install Replications in Project Features.
- Select Create Replication in Network
- Associate the replicated controllers with the Real controllers (see below)

Associating project Controllers with real Controllers

Once you have added and configured your project Controllers (all that is required for programming and simulating) you must associate them with real Controllers on the network.

To Associate a Local Controller:

1. Select the project Controller, the row will highlight
2. In Controller Properties use the Serial Number pull-down to chose a real Controller of the same type to associate (the serial number can be found on the base of the LPC 1, 2 & 4, TPC and rear of the LPC X)
3. The real Controller will fuse with the project Controller so completing the row details

To Associate a Remote Controller:

If you know the IP address of a controller, you can use the Find button to input the IP Address of the controller directly.

This can be used to connect to a controller through a VPN connection or using port forwarding and a public connection.

Should this be the case, ensure that both Port 80 and Port 38008 are forwarded to the controllers local IP address.

To Identify a Device (Beacon):

1. Select an associated project Device or unused Device
2. Press the Beacon button on the mode toolbar, all the Device's status LEDs will flash. The screen back-light of a Touch Device will pulse.
3. Press Beacon again to return the Device to normal operation

Once all your project Devices have been associated with real Devices you can configure them, test your programming on the installation itself and finally upload to the Devices for stand-alone operation.

Network Primary

One Controller in your project must be allocated as the Network Primary, the first project Controller added is chosen by default. The Network Primary is responsible for network playback synchronisation and for issuing realtime and astronomical clock triggers. To set the correct date and time set, see Controller Configuration.

To change the Network Primary:

1. Select the project Controller which is to become the Network Primary
2. Press the Set as Network Primary button in the Properties pane

Web interface tools

To View a Controller's Web Interface:

1. Select the Controller
2. Press the Web Interface button on the Controller toolbar
3. Your computer's default browser will open the Controller's home page (or custom page if one has been created)

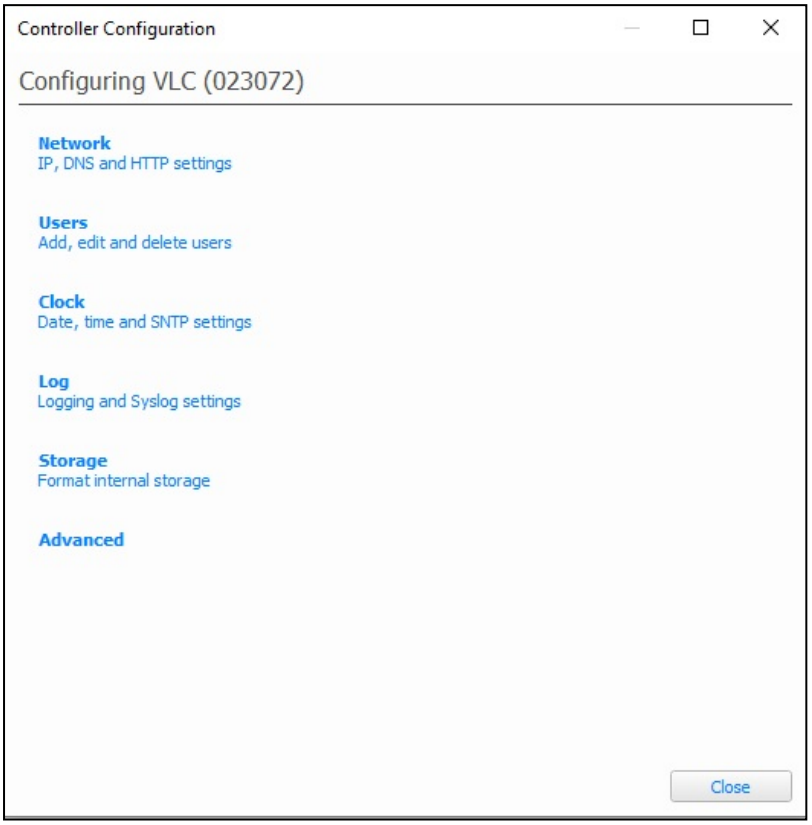
Device status

The fields in the device table provide status information:

Number	The unique identifier given to each Device in the project
Name	The user name given to each Device in the project, typically a name that identifies the Device's purpose or location
Type	The type of Device
Serial	The Device's serial number as found on the Device
Status	Indicates whether the Device is currently locked
IP Address	The Device's IP address which is either statically assigned or obtained from a DHCP server
Firmware	The Device's firmware version which must match that of Designer
Project Name	The name of the project that has been uploaded, as defined in the Project properties, or the project's file name.
Capacity	The number of available (unused) channels on a Controller
Used Channels	The number of used (patched) channels on a Controller

Device Configuration

With a Controller or Device selected, choose Configure on the Network Menu:



Configuration settings will remain after a reboot. However, these settings will be cleared after a factory reset of the Controller / Device. Uniquely, these settings are stored on the Controller / Device itself, not in the project or as part of the upload. The Controller / Device does not have to be associated with a project to have its settings configured from Designer.

Network

Network

Use these fields to set a static IP address for the Controller, by default the Controller is set to receive an IP address from a DHCP server.

You can also set the Subnet Mask for the controller using CIDR or dotted decimal notation, a Default Gateway and up to two Domain Name Servers (DNSs).

If the IP settings have been stored on the Controller's memory card as a "TPC.cfg" or "LPC.cfg" file then these fields will be greyed out.

Web Server

The ports opened by the Controller for access to the web server using HTTP or HTTPS can be manually configured. This can be useful if there are several Controllers in an installation and remote access is required via a router setup for port forwarding to each Controller.

By default the Controller uses port 80 for HTTP and port 443 for HTTPS for the web server.

Users

The Users page allows you to create users and assigned them to Access Groups.

Access to the controller and its web interface can be restricted based on one of several Access Groups.

To add a user, click the Add button. This will open a dialog which prompts for a username and password, and a Group Selection:

Access Group	Description
Admin	User has admin control of the device, allowing uploading and access to advanced Web Interface features (File Manager, Configuration)
Control	User can trigger actions on the Controller (Control)
Status	User can access the state of the controller (Home, Project Status, Log, Output, Input, Network)

Guest permissions can also be set here. This would determine how much of the Web Interface would not require a password. For example, setting Guest Permissions to "Status" would allow access to the Home, Project Status, etc, without needing a user to log in.

Clock

Date and Time

All Controllers have an internal realtime clock which is battery backed and so will operate even when the Controller is not powered. Whilst the internal realtime clock is accurate, the use of a Network Time server of some sort (NTP, DHCP) is recommended where possible (see above).

The Date and Time fields display the current settings of the selected Controller's realtime clock. Only the designated Network Primary needs be accurate as any other networked controllers will automatically synchronize their realtime clocks to the Network Primary.

Note that the project location settings include the correct GMT offset so, if using these location settings, you should set the time here to GMT not local time or the offset will be doubled.

To manually change the Network Primary's date and time:

1. Select the Network Primary
2. Enter the required settings into the Date and Time fields
3. Press Set

To synchronize the Network Primary's date and time to Designer:

1. Select the Network Primary
2. Ensure that the PC running Designer is set to the correct date and time
3. Press Sync to Designer

NTP Server

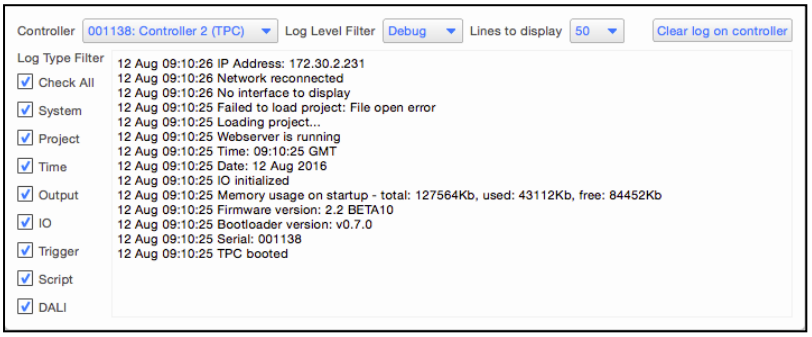
Check this and enter the IP address of the appropriate Network Time Protocol (NTP) server. You can also set the interval to Query the NTP server for the current time.

Note that Controllers with DHCP enabled will also synchronise with a suitably configured DHCP server.

Log

Log

Select the verbosity (detail) of the log that can be viewed either via the web interface or from within Designer using the Controller Log Viewer in the Main Menu and selecting a Controller (which can be connected via Ethernet or USB):



Syslog

Check this to enable logging to the specified IP address. Note that there is a performance penalty to pay for using Syslog so this should only be enabled for debugging.

Storage

The capacity of the Controller's memory card is displayed here with the option to format the card if required, press "Format Memory Card". Formatting the card will erase all project data and so an upload will be required to restore normal operation.

Advanced

Hardware Watchdog

Check this to enable the internal watchdog that will reset the Controller automatically in case of a software crash as a result of either a coding error (“bug”) or a random electromagnetic event such as a power brown-out or spike, nearby lightning strike or static discharge. A startup trigger will be required to determine what the Controller should do after such a reset, see triggers.

Watchdog Protection

Repeated reset protection will remove the controllers project file from the memory card if the controller reboots 5 times within 90 seconds. We strongly recommend leaving this setting on, especially if the controller is to be accessed remotely.

Storing configuration settings on the memory card (optional)

After a reset, LPCs look for a file called "lpc.cfg" on the memory card before using its current settings. TPCs look for a file called "tpc.cfg". You can use a text editor (e.g. Notepad) to create this file and copy it to the memory card to force the issue, useful for transferring the IP settings on the card with the project data. The format of the file needs to be:

```
ip 192.168.42.56 255.255.255.0 192.168.42.250
```

```
http 80
https 443
dns1 192.168.42.254
dns2 192.168.42.1
ntp 192.168.42.1
syslog 192.168.42.8
loglevel 3
watchdog off
```

ip	Required	Defines the IP address, Subnet Mask and Default Gateway for the controller
http	optional	Defines the HTTP port used by the controller (default 80)
https	optional	Defines the HTTPS port used by the controller (default 443)
dns1/dns2	optional	Defines a Domain Name Server (DNS) for the controller to use to resolve host names
ntp	optional	Defines the IP address of an ntp server that the controller should get its time from.
syslog	optional	Defines the IP Address of a Syslog server on the network
loglevel	optional	Defines the log level to be used by the controller, options
watchdog off	optional	Disables the controller's watchdog (not recommended), omit line to enable watchdog

loglevel Options:

0. Critical
1. Terse
2. Normal
3. Extended
4. Verbose
5. Debug

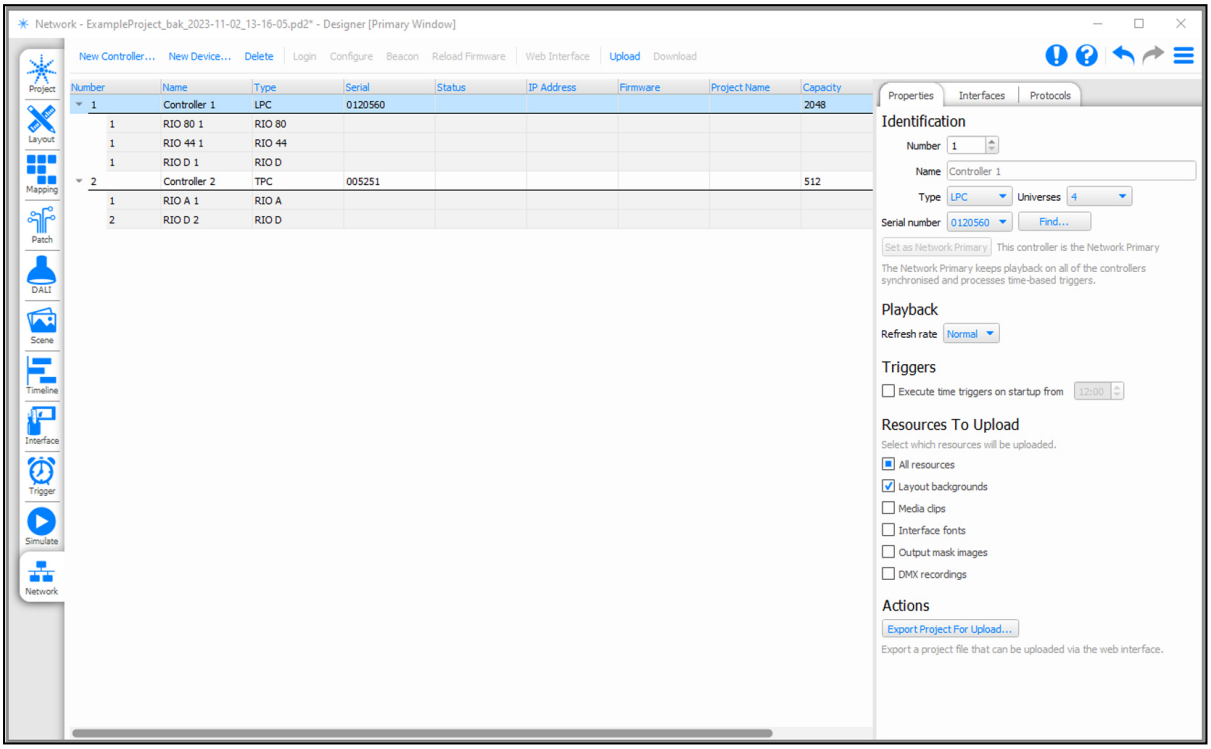
Using the tpc.cfg or lpc.cfg file to store the Controller's configuration on the memory card allows a Controller to be swapped, in case of failure for example, by just moving the memory card into another Controller.

Any settings within a *.cfg files will override settings set from within Designer.

NOTE: This file needs to be placed on the root of the memory card

Device Properties

With a Device selected, choose the Properties tab:



Identification

Use these fields to identify a project device with a name and type, then associate it with a real device and set the Network Primary, see Device Association.

Screen

Use these fields to manage the properties of a touch device and its display.

Interfaces for the TPC, TPS, TPS 5 and TPS 8 are created on the Interface tab, which is enabled when a relevant Touch Device is added to the project. The Interface dropdown shows all interfaces relevant to the selected device, e.g. a TPS 5 interface won't be shown if a TPC is selected, and vice versa. A new interface can be created by selecting the New... button, or the currently selected Interface can be edited by selecting Edit.

Set the backlight brightness for normal operation and for when the touch device has been inactive for a period of time. The inactivity time is also set here, along with the time before the screen turns off completely.

Set whether the backlight brightness should automatically adjust for changes in the ambient light level, and whether the screen should turn on if the proximity sensor detects someone walking up to it.

NOTE: Proximity detection is not available on TPS 5 or TPS 8.

Playback Refresh Rate

Select between Normal (33Hz: default, recommended), High (44Hz) or Low (20Hz: useful for older fixtures with DMX compatibility issues).

60Hz is also available, although is higher than the specification baud rate of DMX, so DMX output channels will be limited to 340 per universe. This will not affect eDMX output, although the nodes may suffer from the same limitation.

60Hz can be enabled in Project Features.

Triggers

If your project uses realtime or astronomical triggers then, when the Controller starts up, you may well want to assert the correct show state for the current time. You can set each Controller to do this automatically by checking the “Execute realtime triggers on startup” option.

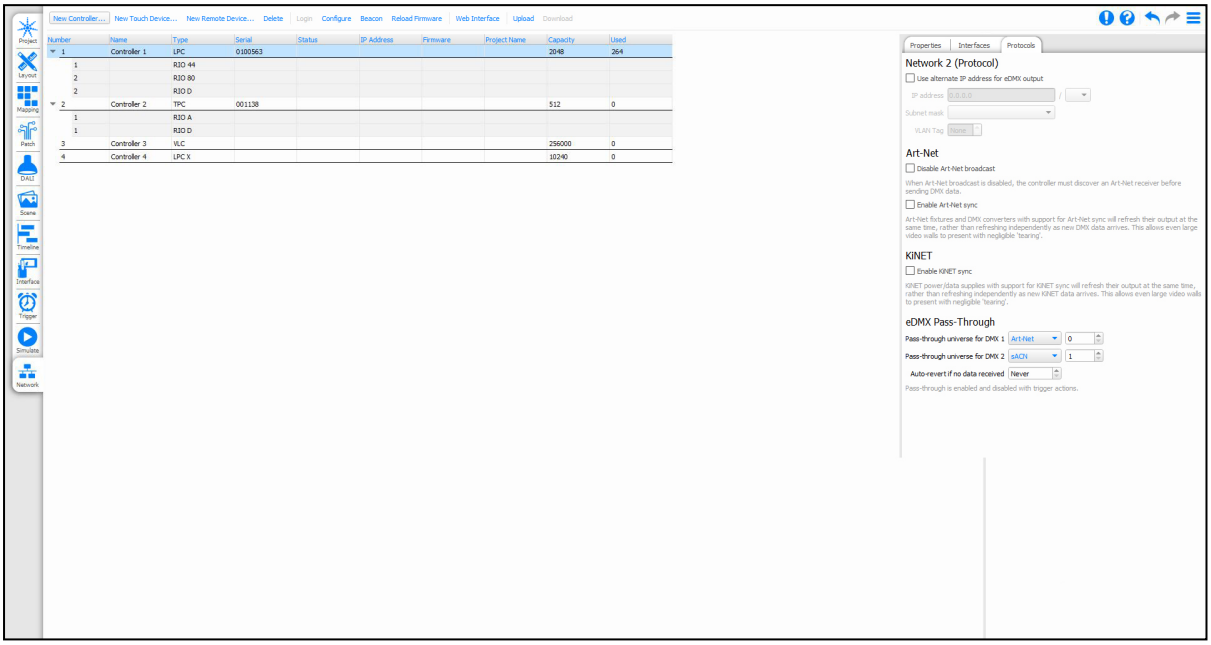
When this option is selected the Controller will execute all the realtime and astronomical triggers that would have fired between the user-specified time and the current time. You should select a time of day when your project is inactive or at its least active. Running all triggers from that point ensures that your project is in the correct state.

Any startup trigger in your project will run first and then any pending realtime or astronomical triggers. The Controller will attempt to preserve the timing so that timelines will be in the correct place as if it had been running normally.

IMPORTANT: Changes made to a Controller's properties will only take effect after an upload.

Controller Protocols

With a Controller selected, choose the Protocols tab which is also available from the Patch window:



Depending on the Controller type selected, the Protocols tab allows you to configure the available eDMX and other output protocols:

Network 2 (Protocol)

TPC, LPC 1, 2 & 4

Though the TPC and LPC 1, 2 & 4 only have a single Ethernet port, this port may be configured with two IP addresses - one for management data and the other for output protocol data. The default setting is for this dual IP mode to be disabled, so that the same IP settings are used for management and output protocol data. Dual IP mode is particularly useful for working with KiNET power supplies, which must use 10.xx.xx.xx addresses.

The Network 2 virtual port can be configured with a VLAN tag to route the data to the relevant VLAN when used managed networks and VLANs to separate the Management and Data networks within the network infrastructure.

LPC X, VLC and VLC+

The LPC X, VLC and VLC+ have a dedicated Ethernet Protocol port. The default setting is to obtain IP settings via DHCP but static IP settings can alternatively be set as required.

DMX Proxy (LPC 1 only)

If a Designer project has a TPC and an LPC 1, the TPC can output local DMX via the LPC's second DMX port. With the LPC 1 selected, choose the TPC from the drop down list.

Art-Net output customisation

By default, a controller with less than 30 universes of Art-Net patched will broadcast all data until a device requests unicast for a specific universe. Controllers with more than 30 Art-Net universes patched will only unicast data to devices requesting universe data and will not automatically broadcast.

There is the option to 'Disable Broadcast' for a controller. The controller will still unicast data to devices that request it. There is also the option to 'Always Broadcast' on a per universe basis. This will force the controller to always broadcast that universe's data. 'Always Broadcast' will override the 'Disable Broadcast' option.

Within the Protocol Properties, is the option to enable Art-Net Sync, which can be used to ensure multiple Art-Net receivers stay in sync (if they are capable of receiving this)

KiNET Output Customisation

Within the Protocol Properties, is the option to enable KiNET Sync, which can be used to ensure multiple KiNET receivers stay in sync (if they are capable of receiving this)

DVI/DisplayPort (LPC X only)

To output data using the DVI port you must first create a pixel matrix that matches the LED controller's pixel map. Once this has been done, use the Pixel Matrix pull-down on the protocol tab to select which matrix will be output via the port. Any programming for the fixtures in the pixel matrix will now output on the DVI port, not just programming applied directly to the pixel matrix.

The LPC X's port is set to a fixed 1024x768@60Hz resolution which is compatible with most LED controllers. The LED controller (or monitor) MUST be connected when the LPC X boots or resets for the port to become active.

The X and Y offset of the pixel matrix within the 1024x768 output can be set as required. The size of the pixel matrix can be scaled up using the multiplier setting - each pixel in the matrix will occupy an area equal to the square of the multiplier on the output.

eDMX Pass-Through

When using an LPC or TPC+EXT in a project it is possible to allow eDMX from another eDMX source to be passed through to the controller's DMX ports. With an LPC or TPC+EXT selected, the eDMX Pass-Through settings will be shown. Select which universe the DMX port will be transmitting. Note that with an LPC 2 you'll be able to choose a different universe for each DMX port on the controller. There is also the option to auto-revert to the project's output if eDMX isn't received for a specified amount of time.

This setting with setup the port to allow the eDMX pass-through to occur, but is enabled and disabled through Triggers

Note that only Art-Net and sACN are currently supported for eDMX Pass-Through.

IMPORTANT: Changes made to a Controller's protocols will only take effect after an upload.

eDMX Pass Through Merge

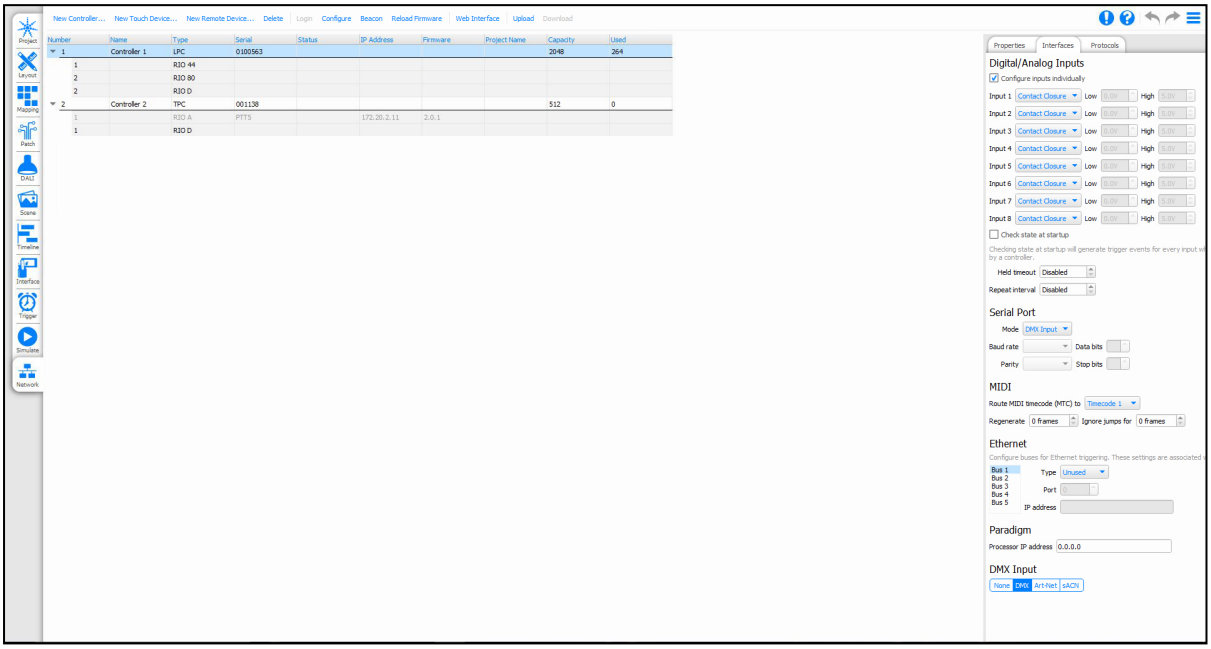
If sACN is being used for eDMX Pass Through and multiple sources are received, the controller will use the Source with the Highest priority.

If multiple streams are received with the same priority:

- If there are exactly 2 streams, the controller will do a HTP merge (per channel). Meaning the highest level for each channel from either source will be used.
- If more than 2 sources are received, then all streams are dropped.

Controller Interfaces

Choose the Interfaces tab to configure the input/output interfaces for a Controller:



IMPORTANT: Changes made to a Controller's interfaces will only take effect after an upload.

Configure EXT (TPC only)

Check the "Configure EXT" box if you want to see EXT interfacing options for a TPC.

Inputs (LPC 1, 2 & 4 and TPC with EXT only)

Inputs can be individually configured as either Contact Closure, Digital or Analog with the latter two modes allowing for the threshold or range to be selected. The maximum voltage range is 0-24V and the smallest measurable change is 0.25V.

Check the "Check State At Startup" box if you want the inputs to be read and acted upon by triggers at startup.

The Held Timeout is used to set a timeout for the Held event, and the Repeat interval is used to set the interval the Repeat.

Serial Port(s) (not Standalone TPC)

Use these fields to configure the Controller's integrated RS232 or RS232/485 serial port(s) specifying the baud rate, the number of data and stop bits as well as any parity bits used to match the settings of the connected device.

Note that the Controllers do not use a specific serial protocol but instead can generate or match any serial string by setting up the appropriate triggers. That being said, the serial port on the LPC 1/2/4 hardware can be configured here to receive the DMX protocol directly.

MIDI (LPC 1, 2 & 4 only)

The LPC's MIDI Input can read MIDI Timecode (MTC) allowing a project to be synchronised with audio-visual or show control equipment. The configuration options are:

- Route To - select one of the six Timecode Buses to which the MIDI timecode (MTC) should be routed.
- Regenerate for - select the number of frames that should automatically be generated in the case of loss of a valid signal.
- Ignore jumps for - select the number of frames that should be considered a valid jump in the timecode value.

MIDI messages other then MTC, for example MIDI Notes or MIDI Show Control (MSC), require no configuration and these protocols can be used simply by setting up the appropriate triggers.

Ethernet

The Controller's Ethernet port can send and receive Ethernet messages allowing a presentation to be synchronised with Building Management Systems (BMS) or show control equipment. The configuration options are:

- Route To - select one of the five Ethernet Buses to which the incoming Ethernet messages should be routed.
- Type - select the messaging protocol (UDP, TCP, TCP Client or Multicast - the latter will require an IP address).
- Port - enter the appropriate port.

When 'TCP' is selected the controller will only send messages after the client has already opened a TCP connection. With 'TCP Client' selected the controller will not receive messages on the bus until the first time the controller tries to send a message, at which point a connection will be made. Once this connection has been made messages sent to the controller will be accepted.

Note that the Controllers do not use a specific Ethernet protocol but instead can generate or match any Ethernet string by setting up the appropriate triggers.

Certain ports are disallowed due to them being used by other processes in the controller:

UDP	TCP
38007	38007
38008	38008
5568	HTTP: 80 (unless changed)
6454	HTTPS: 443 (unless changed)
42012	
49800	
55930	

DMX-In

The LPC supports DMX Input as raw DMX on the Serial Port, configured under Serial Port, or as eDMX (Art-Net or sACN). Select either DMX, Art-Net or sACN and a Universe number for DMX reception.

The TPC supports DMX Input via Art-Net and sACN. Select either Art-Net or sACN and a Universe number for eDMX reception.

The LPC X supports DMX Input via Art-Net or sACN. Select either DMX for the integrated port or Art-Net or sACN and a Universe number for DMX reception.

The VLC/VLC+ supports DMX Input via Art-Net or sACN. Select either DMX for the integrated port or Art-Net or sACN and a Universe number for DMX reception.

When using Art-Net as the Input source, the Universe must be specified as Net/Sub-net/Universe.

DMX Input Merge

If sACN is being used for DMX Input and multiple sources are received, the controller will use the Source with the Highest priority.

If multiple streams are received with the same priority:

- If there are exactly 2 streams, the controller will do a HTP merge (per channel). Meaning the highest level for each channel from either source will be used.
- If more than 2 sources are received, then all streams are dropped.

DALI (TPC+EXT only)

The EXT has a DALI bus interface. This can be used to control DALI ballasts via timeline programming and to receive DALI commands for use in DALI Input triggers.

Video Input (LPC X, VLC and VLC+ only)

The LPC X can receive video in to the controller over the DVI-D connection on the back.

The incoming video can be scaled up or down to an appropriate size for the pixel matrix.

Maximum size is 1920x1080 and minimum is 48x32.

By default, scaling is disabled.

NOTE: The VLC/VLC+ automatically scales the incoming video to the size of the Content Target

Audio Output (LPC X, VLC and VLC+ only)

The LPC X, VLC and VLC+ can be used to output timeline audio. The Default Volume for this can be set here. The current level can be set using the Set Volume Action.

Audio Input (LPC X S3 only)

The stereo balanced line level audio input of a LPC X S3 can be used for timecode input. Each channel can be used independently with the following configuration options:

- Route To - select the Timecode Bus to route the timecode to.
- Regenerate for - select the number of frames that will be generated by the LPC X's software flywheel in the event of a drop in timecode signal.
- Ignore jumps for - select the maximum size of jump in incoming frames that will be ignored.

Remote Devices

Please refer to the documentation supplied with the units for hardware details and installation instructions.

Connection

Other than the EDN, which requires mains power, Remote Devices can only be connected using a Power over Ethernet (PoE) connection. A suitable PoE repeater or switch must be provided.

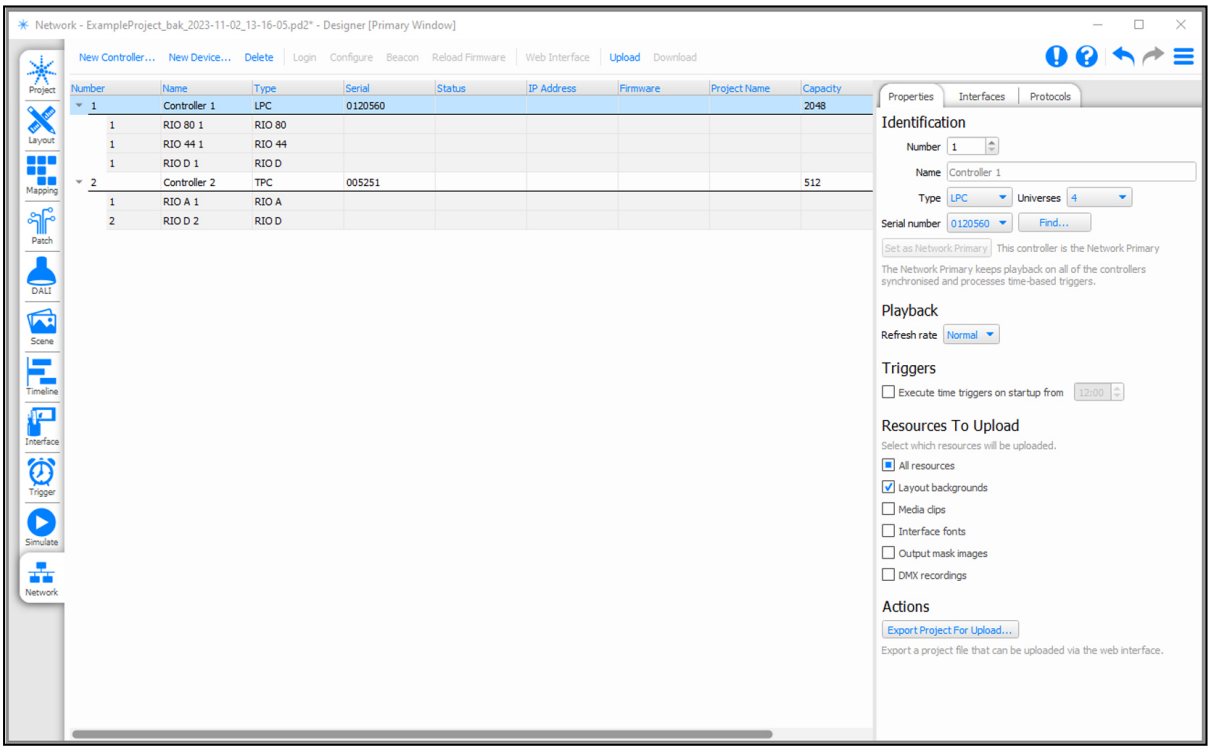
Multicast

Discovery of Remote Devices, from Designer or a controller, is achieved using multicast traffic with the 239.192.38.8 multicast group. For this reason, multicast must be available on your network.

TCP/IP

Each remote Device in the system must have an IP Address in the same range as the controllers in the project. This TCP connection is used for all communications with the controller, once the discovery process has been completed using multicast.

The Device table allows you to manage and configure any Remote Devices in the project and found on the network:



Project vs real Remote Devices

The list of Remote Devices is split into two sections: At the top is the list of project devices which may or may not be associated with real devices. Underneath is a list of all the unused real devices found on the network that have not been associated with project devices.

Managing Project Remote Devices

To add and set the type of a project Remote Device:

1. Press the New Device button in the toolbar.
2. In the New Device dialog, select the device type (RIO 80, RIO 44, RIO 08, BPS, BPI, RIO A, RIO D, RIO D4, RIO G4, EDN 10, EDN 20, TPS, TPS 5, TPS 8).
3. Choose the device's number (on some Remote Devices this is selected on the device itself, see Associating Remote Devices below).
4. Choose the device's parent controller.
5. Press Add, the Remote Device will be added to the project (and associated to a real device if one of the correct type and address is found on the network).

To delete a project Remote Device:

1. Select the project Remote Device by clicking its row, the row will be highlight.
2. Press the Delete button in the toolbar.
3. The Remote Device will be removed from the project and, if no longer associated at all, the real device will move to the bottom of the device table.

Remote Device firmware

IMPORTANT: Remote Device firmware may need to be updated if a new version of Designer software has been installed. Devices with incompatible firmware will be highlighted in red.

To update a Remote Device's firmware:

1. Select the incompatible device by pressing the left hand button, the row will be highlighted.
2. Press Reload Firmware on the Remote Device toolbar.
3. The firmware update will proceed - you must not disturb this process.

Associating Remote Devices

Unlike Controllers, which are uniquely associated with a project via their serial number, Remote Devices are associated by their address as selected on the unit itself. Automatic addresses 1-15 (1-4 for RIO G4) are provided with a manual option (M) for selecting more (up to 100). Remote devices may share the same address and thus identity, useful for repeating a user interface at both ends of a corridor for example.

To associate a project Remote Device with a real device (automatic addresses):

1. Select the project Remote Device by clicking the left hand button, the row will highlight.
2. Ensure that the device type and number matches a suitable unit, addressed at this number, found on the network.
3. The real device will move from the Unused list and fuse with the project device so completing the row details.

To associate a project Remote Device with a real device (manual addresses):

1. Ensure that the Remote Device is addressed to the "M" setting, you will need to note its serial number (label on back).
2. Select the project Remote Device by clicking the left hand button, the row will highlight.
3. Select the correct device type and the desired number.
4. Select the Remote Device's serial number from the pull-down menu of devices found on the network.
5. The real device will move from the Unused list and fuse with the project device so completing the row details.

Once all your project Remote Devices have been associated with real devices you can configure them, test your programming on the installation itself and finally upload to the Controllers for stand-alone operation.

Remote Input Output (RIO) device properties

Serial Port

The RIO 80, RIO 44 and RIO 08 have a multi-protocol serial port that can be configured to either RS232 full-duplex or RS485 half-duplex operation. The configuration options are:

- Type - select RS232 or RS485 as required.
- Baud rate - select the baud rate.
- Data bits - select the number of data bits (typically 8).
- Stop bits - select the number of stop bits.
- Parity - select the parity type.

I/O Configuration

The RIO 80, RIO 44 and RIO 08 differ by virtue of the number and type of I/O ports:

RIO 80	Eight inputs, no outputs & serial port
RIO 44	Four inputs, four outputs & serial port
RIO 08	No inputs, eight outputs & serial port

Inputs can be individually configured as either Contact Closure, Digital or Analog with the latter two modes allowing for the threshold or range to be selected. Outputs can be individually configured with a Startup state, whether the relay is on or off at startup.

Check the "Check State At Startup" box if you want the inputs to be read and acted upon by triggers at startup.

The "Maintain state on device loss" helps preserve the state of the RIO output relays should the RIO lose its connection to the controller. If this option is checked the RIO will go back to the same state it was in when it was last connected to the controller.

NOTE: The "Maintain state on device loss" feature will work if the RIO device restarts, or if the RIO loses its connection with its controller. However, if the controller restarts then the RIO settings will go back to the default values set within the Project file because the device memory is wiped upon restart.

The Held Timeout is used to set a timeout for the Held event, and the Repeat interval is used to set the interval the Repeat.

See triggers for usage.

Audio

The stereo balanced line level audio input of a RIO A can be used for Audio triggers. Select the Audio button to enable this mode and to see the following configuration options:

- Route To - select the Audio Bus to route the incoming audio to.
- Freq. Bands - select the number of frequency bands with which to analyse the incoming audio (max 30 per channel; the frequency bands sit along a logarithmic scale and have been chosen for an optimum response to music).
- Gain - turn Auto gain on or off, and set the manual gain level.
- Peak Decay Rate - set the rate at which peaks in each frequency band will decay (the peak level can be used in triggers).
- Initially Enabled - the audio feed from a RIO A can be turned on or off by triggers, and here you can set the initial state.

See triggers for usage.

Timecode

The stereo balanced line level audio input of a RIO A can be used for timecode input. Select the Timecode button to enable this mode and to see the following configuration options:

- Channel - the audio input of the RIO A that the timecode input will be connected to.
- Route To - select the Timecode Bus to route the timecode to.
- Regenerate for - select the number of frames that will be generated by the RIO A's software flywheel in the event of a drop in timecode signal.
- Ignore jumps for - select the maximum size of jump in incoming frames that will be ignored.

MIDI

The RIO A has a MIDI input and output interface. This can either be used in Remote Device MIDI triggers, or it can receive MIDI timecode. The configuration options here are for MIDI timecode only:

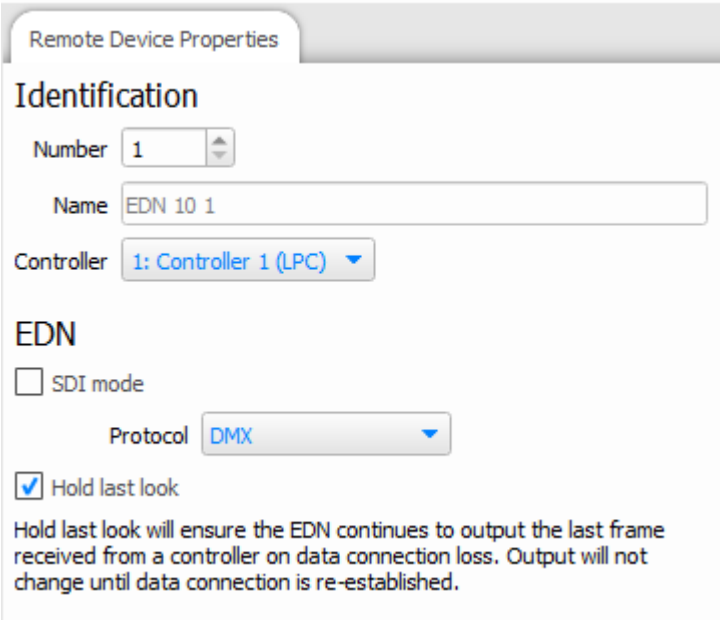
- Route To - select the Timecode Bus to route the MIDI timecode to.
- Regenerate for - select the number of frames that will be generated by the RIO A's software flywheel in the event of a drop in MIDI timecode signal.
- Ignore jumps for - select the maximum size of jump in incoming frames that will be ignored.

DALI

The RIO D and RIO D4 offer DALI bus interfaces. These interfaces can be used to control DALI ballasts via timeline programming or direct commands or to receive DALI commands for use in DALI Input triggers.

Ethernet Data Node (EDN) / RIO G4 and Serial Data Interface (SDI) device properties

Properties



NOTE: Hold last look when active will output at only 5Hz.

SDI

A node can be configured to output via connected SDIs by enabling the SDI checkbox. This will change the Protocols available below.

Protocol

This dropdown will display available protocols of the node. By default, it will offer DMX and UltraDMX, both of which can be connected directly to the node. For other protocols that require the SDI, these options will be displayed once the SDI checkbox has been checked.

Each node can only output one protocol at a time.

Touch Panel Station (TPS) properties

Use these fields to manage the properties of a touch device and its display.

Interfaces for the TPS, TPS 5 and TPS 8 are created on the Interface tab, which is enabled when a relevant Touch Device is added to the project. The Interface dropdown shows all interfaces relevant to the selected device, e.g. a TPS 5 interface won't be shown if a TPS 8 is selected, and vice versa. A new interface can be created by selecting the New... button, or the currently selected Interface can be edited by selecting Edit.

Set the backlight brightness for normal operation and for when the touch device has been inactive for a period of time. The inactivity time is also set here, along with the time before the screen turns off completely.

Set whether the backlight brightness should automatically adjust for changes in the ambient light level, and whether the screen should turn on if the proximity sensor detects someone walking up to it.

NOTE: Proximity detection is not available on TPS 5 or TPS 8.

Button Panel Station (BPS) device properties

Properties

The global properties for each BPS are set here:

- Minimum LED Intensity - set a percentage value as required, useful for ensuring that the buttons are always visible.
- Held Timeout - set the amount of time in milliseconds that a button must be pressed to be considered as being held.
- Repeat Interval - set the interval in milliseconds that a held button will transmit a repeat signal.

Button Configuration

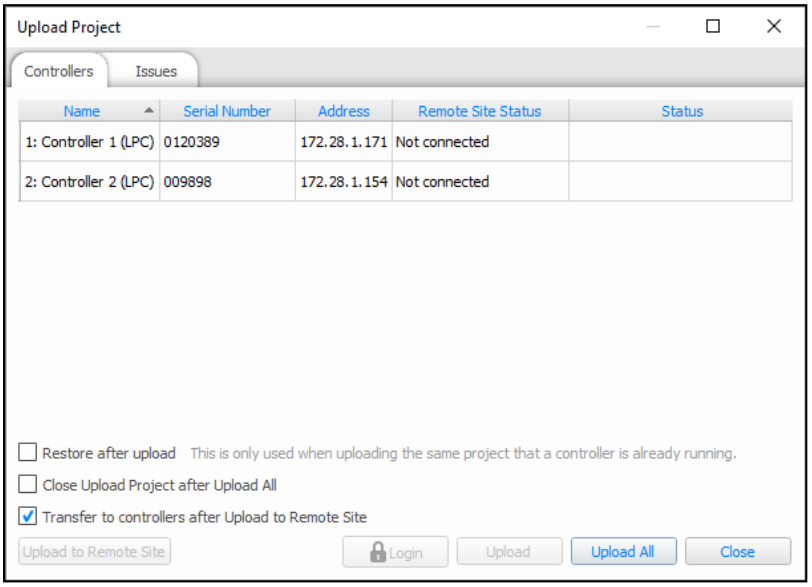
Each BPS has eight buttons with an integral white LED and the default setting for each button is set here:

- Effect - select the default LED effect (Off, Static, Slow Flash etc.).
- Intensity - set the default LED intensity (0% will equal the Minimum LED Intensity as set above).

See triggers for usage and BPS learning IR receiver for infra red operation.

Upload

Once you have confirmed that your programming is as you want you can upload to the Controllers by either pressing the Upload button on the Network window or via File > Upload (Ctrl + U):

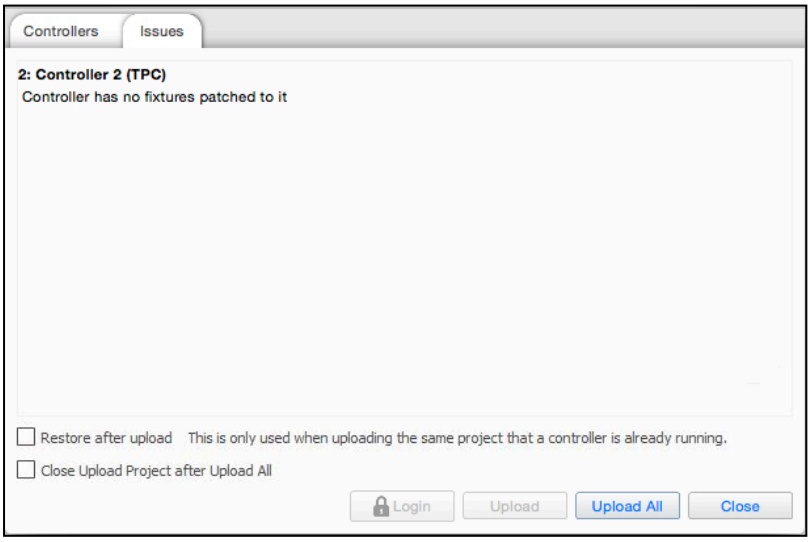


Direct upload (as opposed to remote, see the web interface), is not possible without connected Controllers (see the network section) but this allows you to upload your programming to one or more Controllers for stand alone operation.

You can either use the "Upload All" button to upload to all connected Controllers or select specific Controllers from list and press "Upload" to target them alone.

IMPORTANT: Changes made to the DALI configuration (including DALI groups and scenes) must be uploaded separately to the DALI ballasts, see DALI.

Issues



When you open the Upload dialog, there is a tab which lists potential errors with your project. Designer will check things like triggers and hardware configuration to make sure that there are no inconsistencies. If any issues are found, the Issues tab will be opened automatically and a description of each issue will be listed so

that you can take corrective action, see Issues. You can proceed with the Upload ignoring the errors should you wish.

Restore After Upload

Check the "Restore after upload" box if you want the Controllers to continue playback of active timelines and scenes from where they left off, useful for soft openings when programming is still being tweaked while the installation is open to the public. This is not the same as a startup event and will only restore active playbacks. Note however that changes to the fixture schedule or patch will force the Controllers to reset playback and so cause a momentary black out.

Close Upload Project After Upload All

Check the "Close Upload Project after Upload All" box to close the Upload dialog once the upload process has completed successfully. This option will persist across all future uploads.

Transfer to controllers after Upload to Remote Site

Check this checkbox to automatically transfer the uploaded project onto the controller when uploading to a controller connected to the SixEye Remote Management service via the Upload to Remote Site button.

Upload Options

The three upload buttons all serve slightly different purposes. Upload All will upload the project to all the local controllers in a project (which is usually what you want to do), Upload will only upload to the selected local controller (useful in cases where you are only making tweaks to a single controller but don't want the other controllers to reset), and Upload to Remote Site will upload to all controllers that are currently connected to the Remote Site linked to the loaded Designer Project.

Login

If your controller has a password set for it, you will need to log in to the controller before you can upload to it using the Login button.

Saving Compiled project

The Save button can be used to Save the compiled project file rather than uploading it to the controller. This compiled project can be used to upload over the controller's web interface, or by putting the file on the controller's memory.

The TPS hosts a cut down web interface that can be used to upload a compiled project file remotely.

What's Actually Uploaded?

A compressed version of the standard project file is uploaded to the controller. Typically (space allowing) all media and background images will be uploaded, along with all trigger, timeline and scene information. All controllers get all the data in the project (allowing the project to be downloaded from any controller).

Can the Project File be Retrieved from the Controller(s)?

The project file can be downloaded from the Network tab or the Controller's web interface.

Press the download button on the Network tab toolbar or the download project button on the Controller's Web Interface.

Cloud Association

Controllers within Designer can be added to a connected Cloud site.

Adding Devices

To add a device, you must first log into your Pharos Cloud account in Project Properties and select the relevant Site when prompted.

To Add the Controller to the Site

- Associate the controller with a controller in the project
- Select the project controller
- In the controller's Properties in the right hand pane, Click Add Controller to Cloud.

This will automatically start the process of adding the controller to the specified Cloud Site.

Once the process is complete, a Cloud Icon will appear in the controller's Status column.

Removing Devices

When connected to a Controller locally, the Disconnect Controller from Cloud option can be used to disassociate the controller with the Pharos Cloud Site

Errors

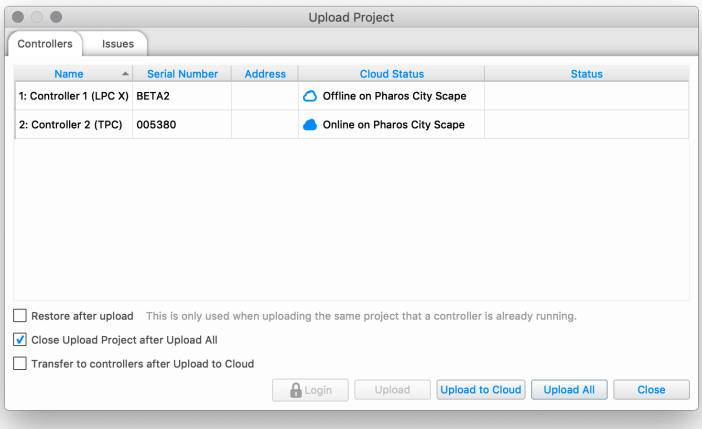
The Controller's Log will show the progress of the controller provisioning to Pharos Cloud.

Sometimes an error may be logged, e.g. if the controller does not have a Name Server correctly configured, or does not have internet access.

Uploading to a Cloud Site

Once you are connected to your Cloud Site, and your controllers are associated, you can upload your project to the Devices in the Cloud Site.

The Upload window will allow you to Upload to Cloud, and an option to choose whether to also Transfer the project to the controller.



If you don't automatically transfer the project, this can be done from the Files page within each controller in the Pharos Cloud Site.

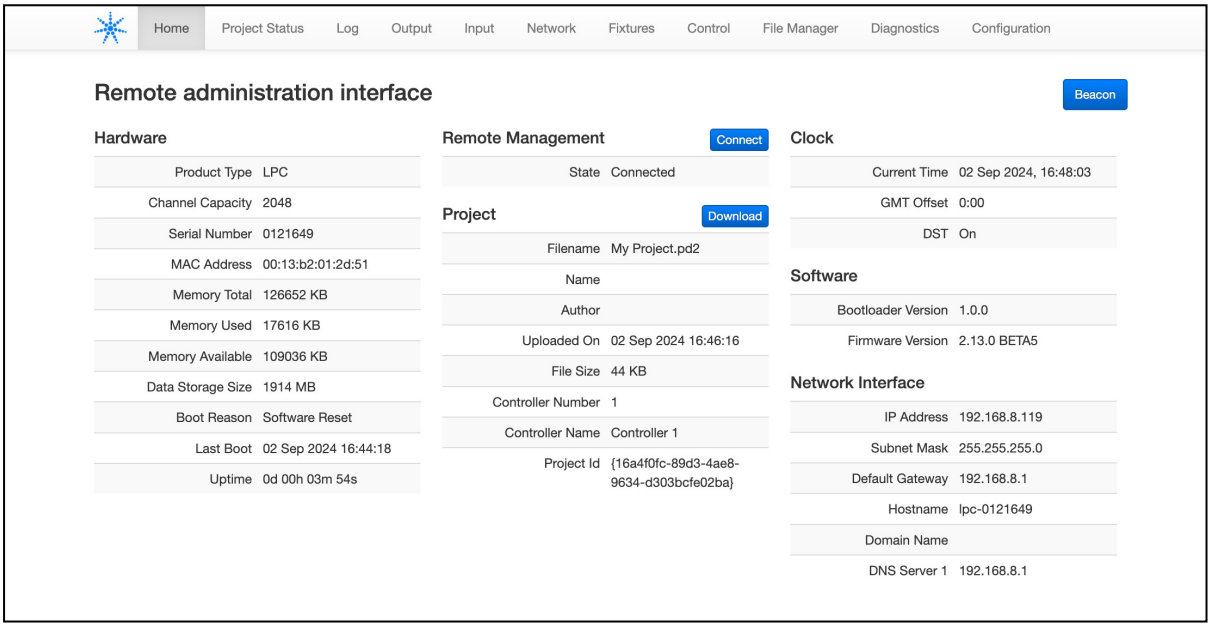
Default Web Interface

The Controller's internal web interface is a very powerful diagnostic and management tool. You can view a Controller's web interface from within Designer or, for remote access, browse to the index page at http://xxx.xxx.xxx.xxx/default/index.lsp, where xxx.xxx.xxx.xxx is the IP address of the Controller.

Some pages may not always be visible if they aren't relevant to the current project file (or if there isn't a project loaded) e.g. IO Modules.

Use the navigation buttons across the top to select these pages:

Home



The home page provides general information about the status of the Controller: Serial number, type, IP address, loaded project details and memory usage. The bootloader and firmware versions and MAC address are also given for reference.

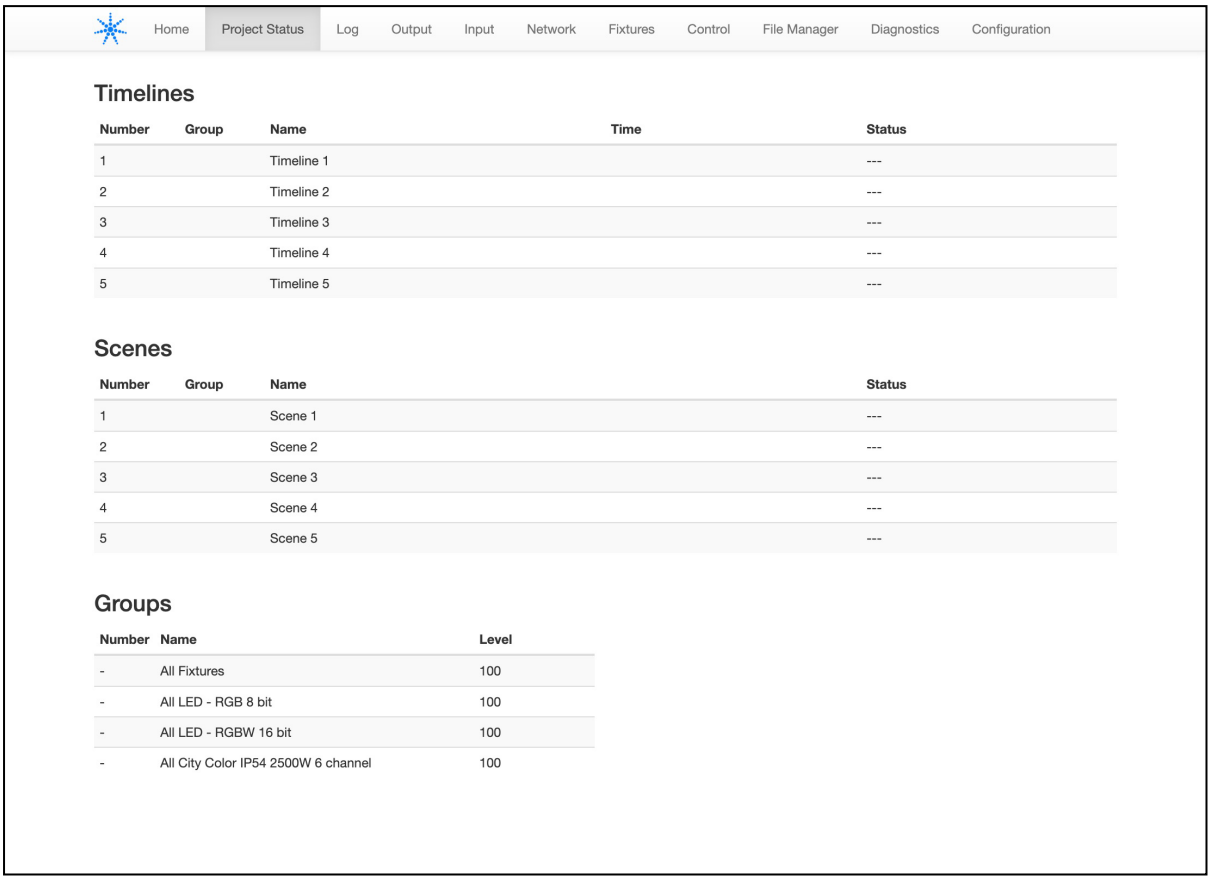
The current uptime of the controller and last Boot Reason are shown to aid troubleshooting.

NOTE: The reset reason is not available on LPC X Rev2 or VLC

From here, it is also possible to download the project file that is currently running on the controller.

For the LPC X, VLC and VLC+ particular attention should be paid to the temperature readings; insufficient ventilation may cause the ambient temperature to rise and thus system & CPU temperatures to reach excessive levels, degrading performance.

Project Status



The status page provides feedback on the current state of playback:

Timelines

All timelines are listed with their current state and running time:

---	Inactive, the timeline has not run since the last reset
Running	The timeline is running and contributing to the output (items "on stage")
Running (Inactive)	The timeline is running in the background and not contributing to the output, generally because it has been overridden
Halted	The timeline is halted and contributing to the output
Halted (Inactive)	The timeline is halted in the background and not contributing to the output
Holding at End	The timeline is holding at end and contributing to the output
Holding at End (Inactive)	The timeline is holding at end in the background and not contributing to the output
Released	Inactive, the timeline has run but has been explicitly released

In systems with more than one Controller it is important to understand that this timeline status is only pertinent to the Controller being accessed. For example, the accessed Controller may report that a timeline is Running (Inactive) because its fixtures are not contributing to the output while another Controller may well be Running (Active) because its fixtures are contributing to the output. In such systems the complete status can only be determined by interrogating all Controllers.

Scenes

All Scenes are listed with their current state:

Started	The Scene is being played back on at least one fixture
Started (inactive)	The Scene has been started, but is not affecting the output, generally because it has been overridden
Released	The Scene is not being output

In systems with more than one Controller it is important to understand that this scene status is only pertinent to the Controller being accessed. For example, the accessed Controller may report that a scene is Started (Inactive) because its fixtures are not contributing to the output while another Controller may well be Started because its fixtures are contributing to the output. In such systems the complete status can only be determined by interrogating all Controllers.

Groups

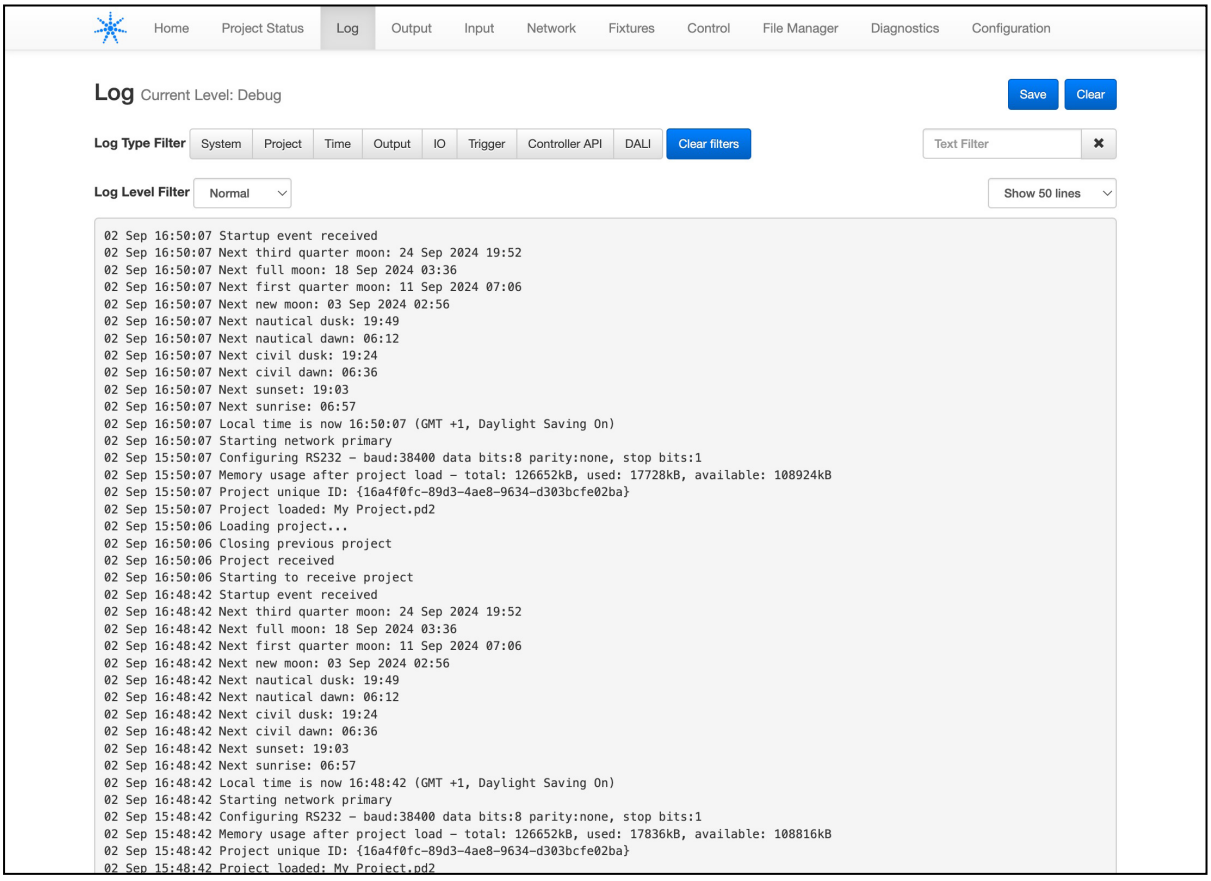
All groups are listed with their current intensity level.

Content Targets

On a VLC or VLC+, Scenes and Groups are replaced by the controller's content target/s. The current intensity master for each content target will be shown.

Use this page in conjunction with the Control and Log pages to interrogate and debug an installation.

Log



The log can be cleared and saved to file using the Clear and Save buttons. Two types of log are provided:

General log

A blow-by-blow account of all activity including input/output,RS232 serial strings for example, and trigger matching. Extremely useful in helping debug complex interfacing and triggering arrangements. Alternatively, the log can be viewed directly from within Designer over an Ethernet or USB connection using View > Controller Log.

System log

A less verbose log of the Controller's system activity, useful for examining the boot-up sequence to help debug problems.

Log Filtering

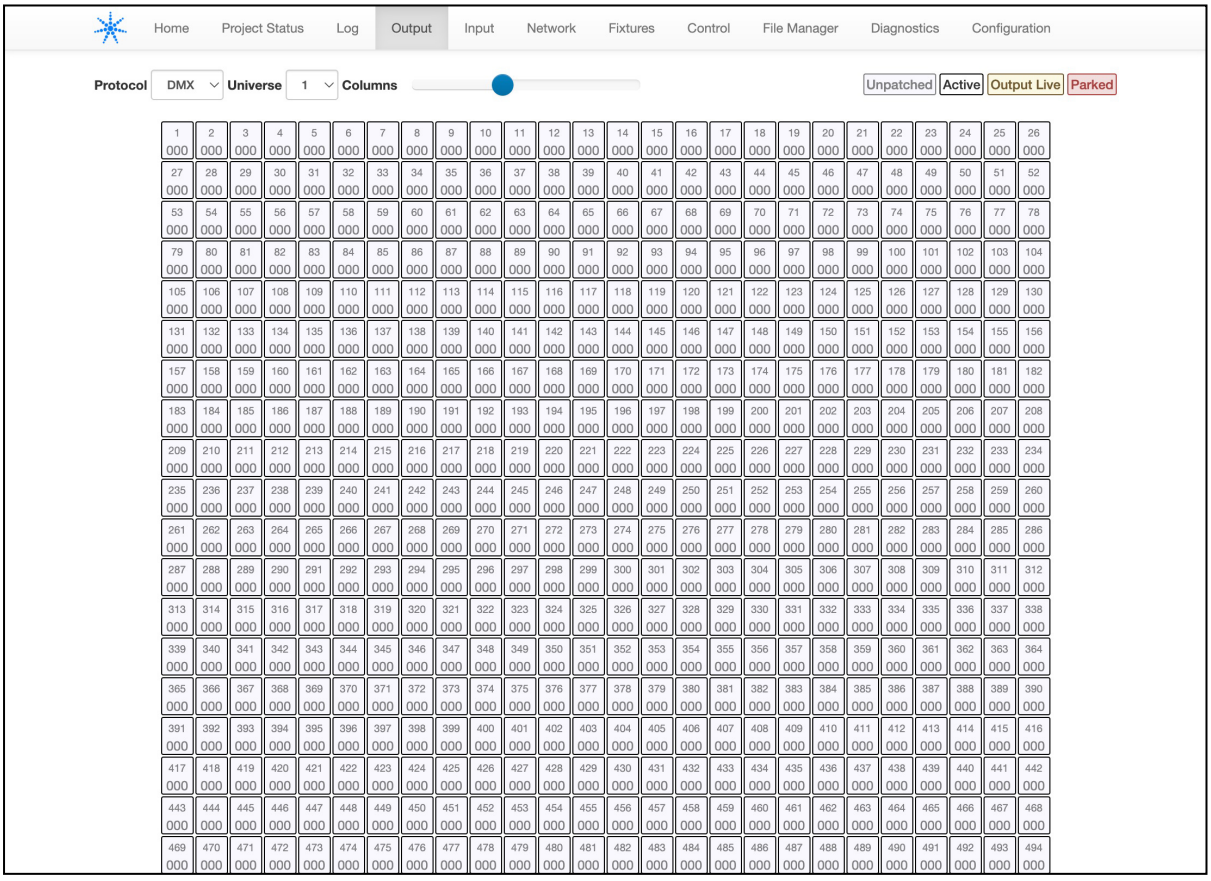
Filter down the log to messages about specific topics using the filter buttons across the top of the log view.

You can also filter the log by text using the Text Filter box on the right hand side.

Show Lines

The Show Lines control allows you to specify how many lines of the log to display. The default is the last 50 lines.

Output



View output

Select the Protocol/DMX Port to examine a numerical snapshot of the control data being output, refreshed every 5 seconds. Select DVI to examine a graphical snapshot of the pixel matrix output.

The channel blocks will change colour depending on how the channel is being controlled:

White	Playback
Grey	Unpatched
Blue Border	Output Live
Red border	Parked

Use in conjunction with Control and Status pages to debug an installation.

A message will also be displayed if the selected output has been disabled by a Disable Output Action.

Park and Unpark

If authentication is active on a controller enter the username and password for an account with Control and/or Admin.

Park allows you to lock the value of a particular channel without actually altering your programming. This can be useful to turn off a fixture that is misbehaving temporarily or to make sure a working light stays on while you are programming.

Park can be accessed from the output view of the web interface, simply enter the channel or range of channels and the value at which to park. Parked channels are shown in red within the output view. There is the option to Unpark from the same view.

Parked channels will remain parked when you upload shows or output live. However all parked channels will be cleared if the Controller is reset or the power is cycled.

Input

Inputs

Use to examine the status of the Controller's inputs.

NOTE: Contact closure return High when the input is open and Low when the input is closed.

DMX

Select the DMX input to examine a snapshot of the DMX values being input, useful for debugging DMX triggering and control.

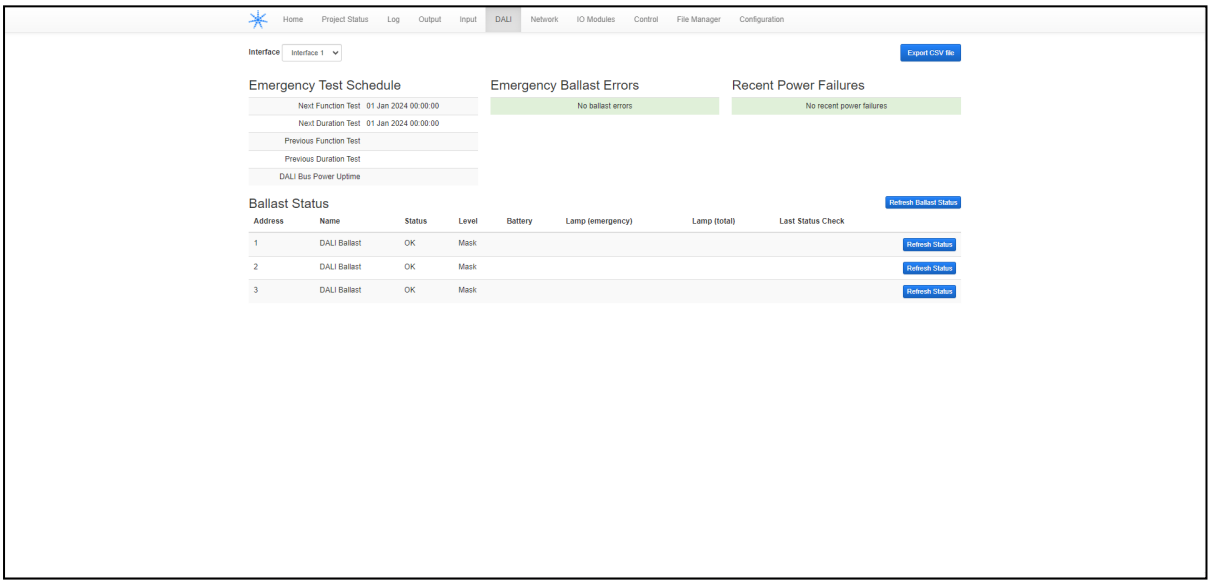
DVI

On LPC X or VLC/VLC+ the current Live Video input will be displayed on the Input page, with the ability to manually refresh the image.

DALI

CSV Export

This information can be viewed in a comma separated values file by clicking the Save button. Copy, paste and save the information you require in a separate document.



Emergency Test Schedule

Only populated if the current interface has emergency ballasts present. View information about when Emergency Ballast tests are due to take place as well as the time and date of previous tests. Also view the uptime of the DALI bus.

Emergency Ballast Errors

Lists all reported errors reported by emergency fixtures on the current DALI interface. Errors will show ballast address, tests failed and reported errors.

Once a ballast has been repaired it can be marked as fixed here or by using triggers. Once a ballast has been marked as fixed it will remain in the Ballast Errors section until a subsequent test has confirmed that the fixture is indeed operational again.

Ballast Status

Lists all ballasts and reported status on the current interface. Standard ballasts show the ballast Address, Name, Status and Level; emergency ballasts also show Battery Charge, Emergency Lamp Hours and Total Lamp Hours.

Refresh Ballast Status

Sends the DALI commands to refresh the status of all ballasts on the interface, and updates the table.

Last Status Check

The date and time when a ballast's status was last updated.

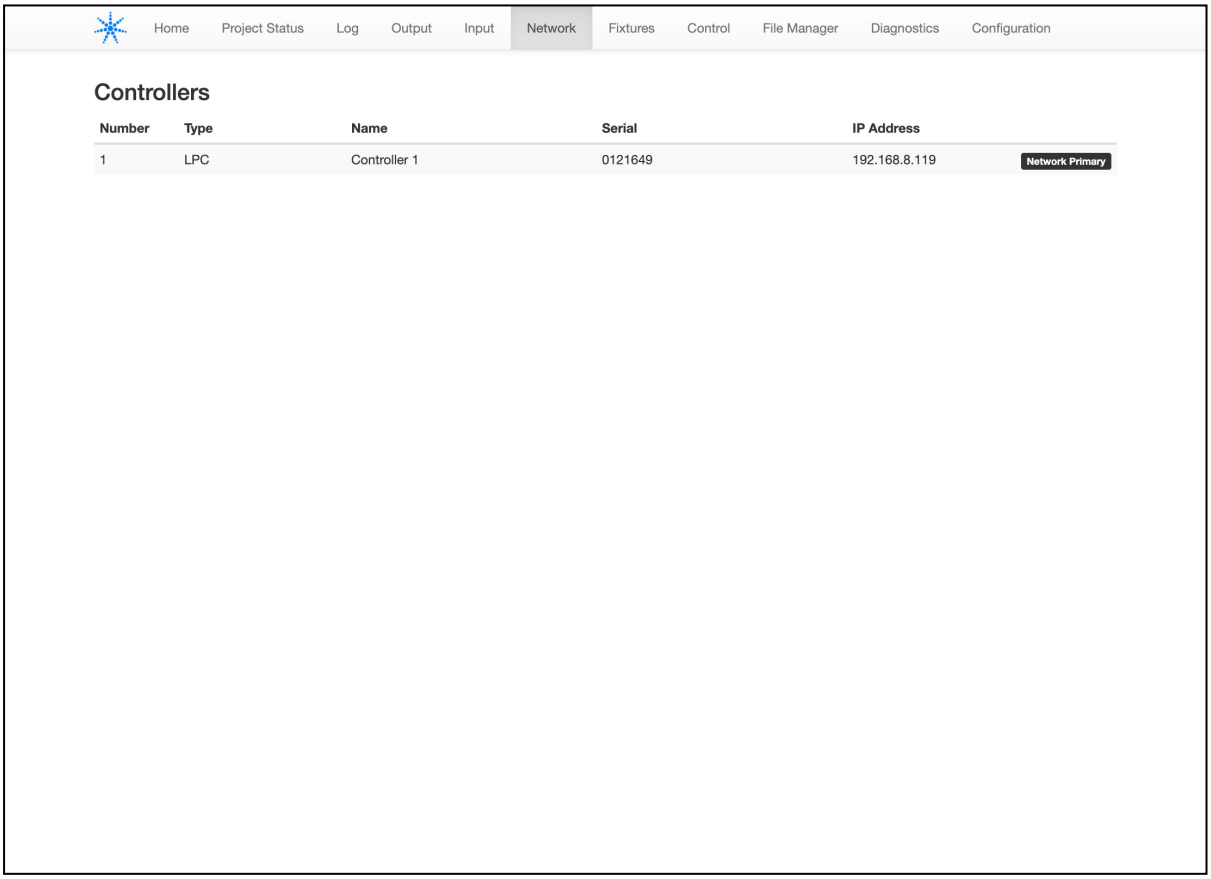
Refresh Status

Sends the DALI commands to refresh the status of a specific ballast on the interface.

Recent Power Failures

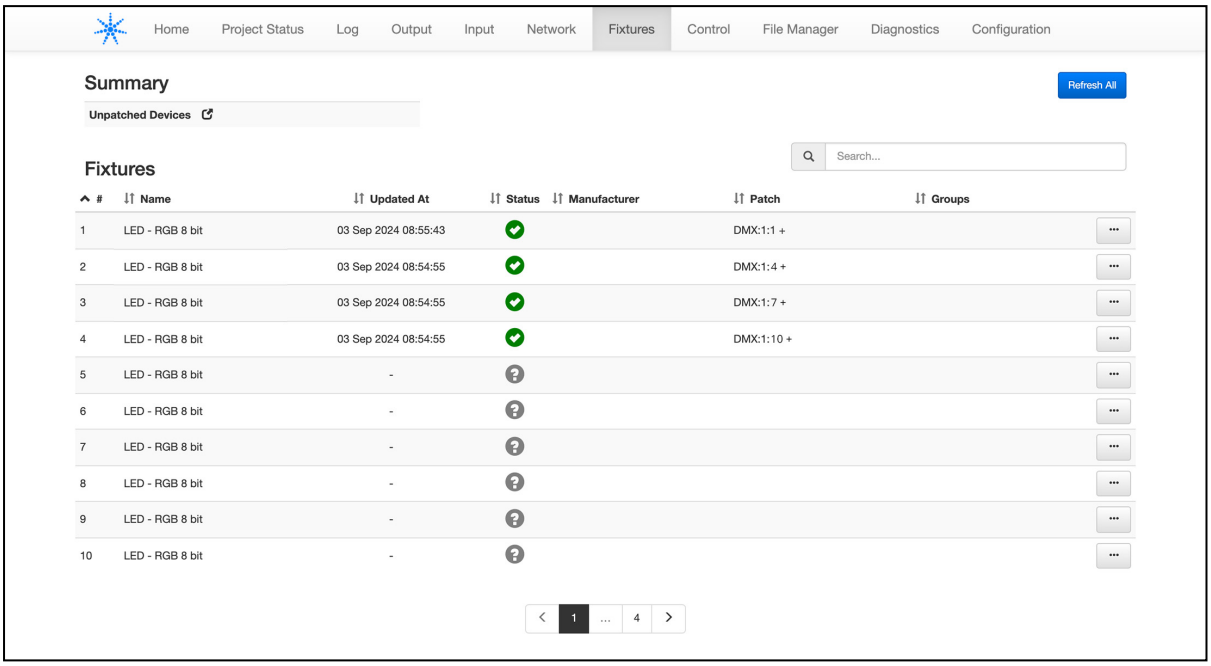
Lists any reports of bus power failures for the current interface.

Network



The Network Page allows access to all the devices in the project.

Fixtures



This page presents an overview of the project's fixtures. Clicking **Refresh** performs RDM discovery and updates the page.

At the top of the page, the summary section shows:

- Time and date of the last refresh
- Number of RDM devices discovered at the last refresh
- Number of unpatched devices discovered at the last refresh

Click **Unpatched Devices** to see devices discovered at the last refresh which are not associated with a fixture in the project. From here, it's possible to identify each unpatched device.

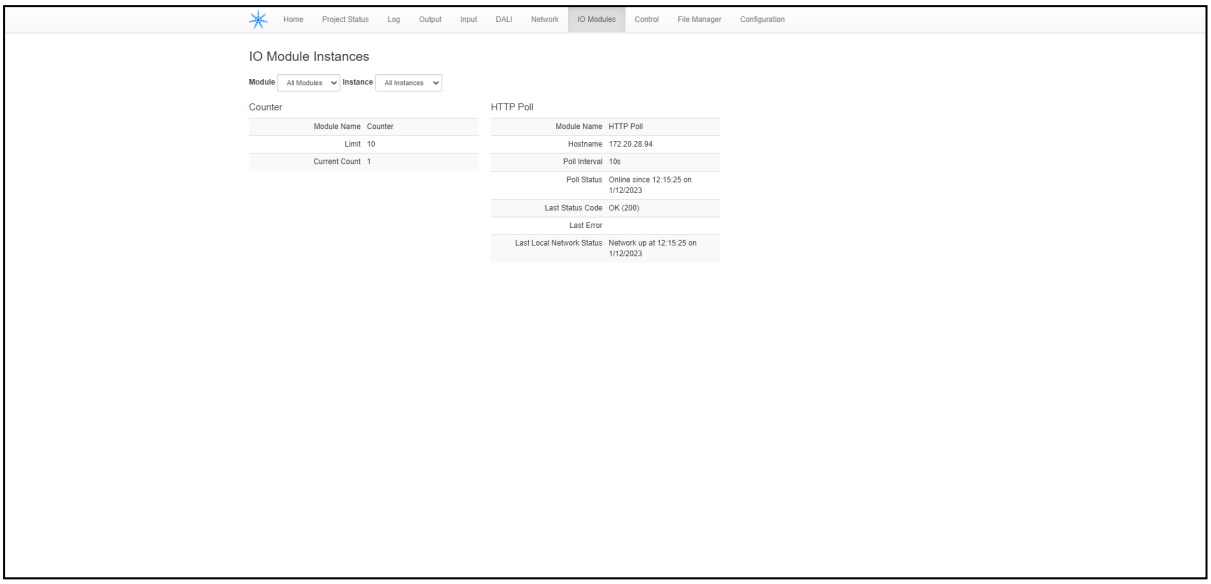
The fixtures table shows the RDM status of each fixture in the project. This can be:

- All RDM UIDs associated with the device have been discovered at the last refresh
- No RDM UIDs associated with the device were discovered at the last refresh
- Some, but not all, RDM UIDs associated with the device were discovered at the last refresh
- There are no RDM UIDs associated with the device, or a refresh has not been performed since the controller was last rebooted
- A refresh is in progress

The fixture table can be sorted by clicking any of the column headers or searched using the search field.

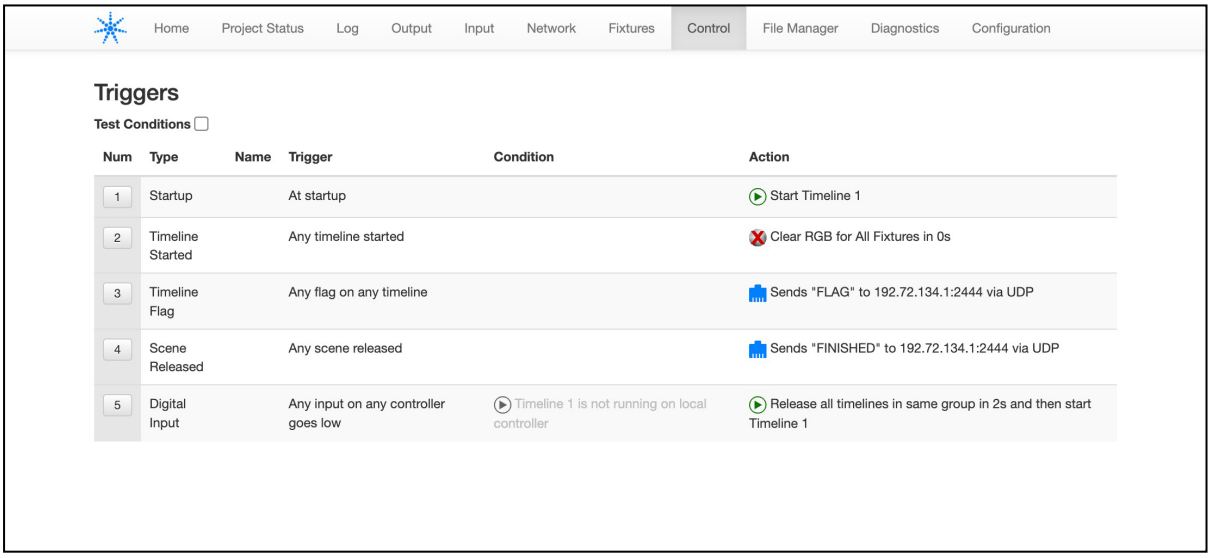
Click the three dots to open a fixture's detail page. This page shows a table with rows for each associated RDM device and the status of each device. Clicking **Refresh** will refresh just the fixtures on this page. From this page, it's possible to edit each device's mode, start address and curve. It's also possible to identify each device from this page. Changes made will be sent to RDM devices but will not modify the controller's project file.

IO Modules



The IO Modules Page will list all the IO Module instances in use that are able to provide feedback of data or settings.

Control



A Controller can be controlled remotely in two ways:

Command line

An advanced feature that allows direct control of a specific Controller's fixtures, timelines and even DMX channels via the script engine, see command line reference.

Alternatively the command line can be customised to run as a Lua Trigger script, see web interface settings for details.

Triggers

Triggers in the project, together with user annotation, are listed here and can be fired by clicking on them. Since a network of multiple Controllers share triggers, firing triggers from one Controller's web interface will trigger all the Controllers in the project.

NOTE: Triggers set to Hidden in the project will not be displayed

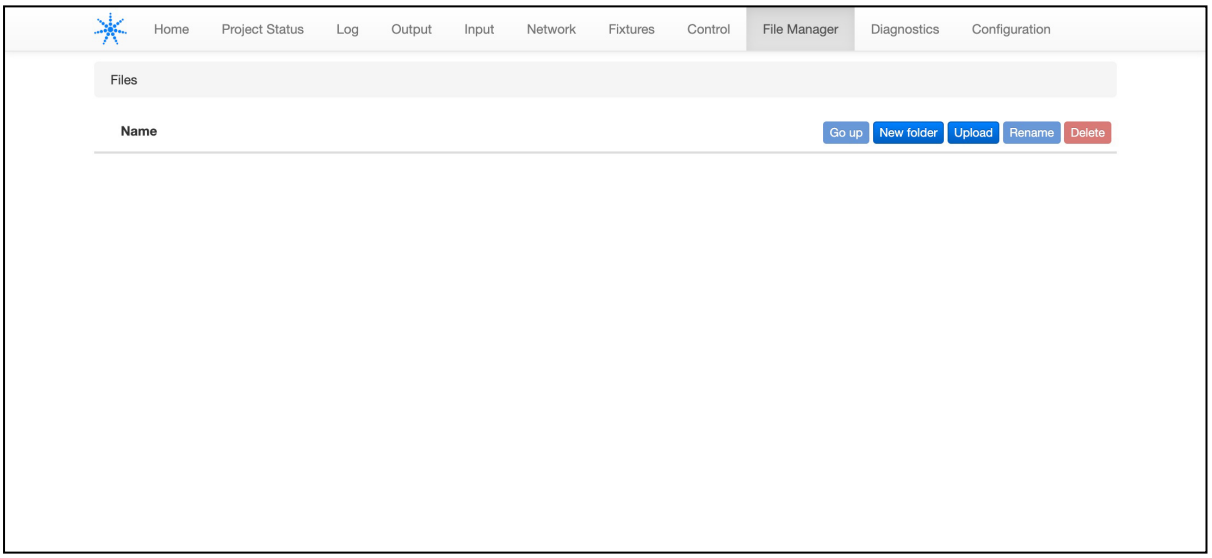
Triggers on the default Control web page will not test conditions by default, check the Test Conditions box to text conditions before firing the trigger.

Dynamic text slots

All the Dynamic text slots are listed with their current value. You can edit any text slot and changes will take effect according to the preset settings on the timeline (immediately, next cycle, on timeline restart).

File Manager

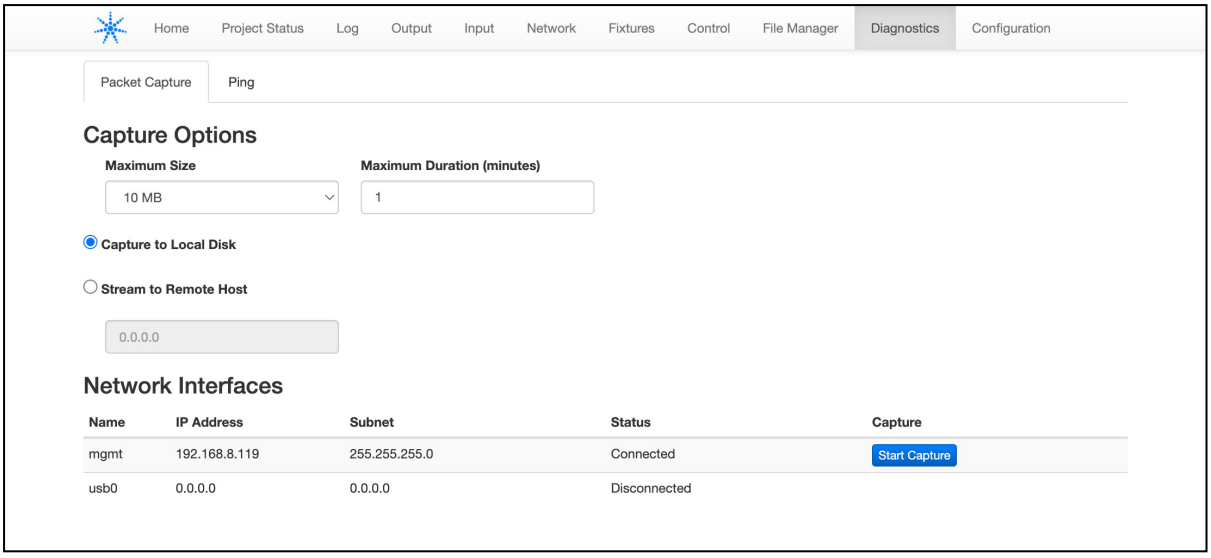
Optionally, files can be uploaded to the controller's SD Card using the File Manager tab within the web interface.



Diagnostics

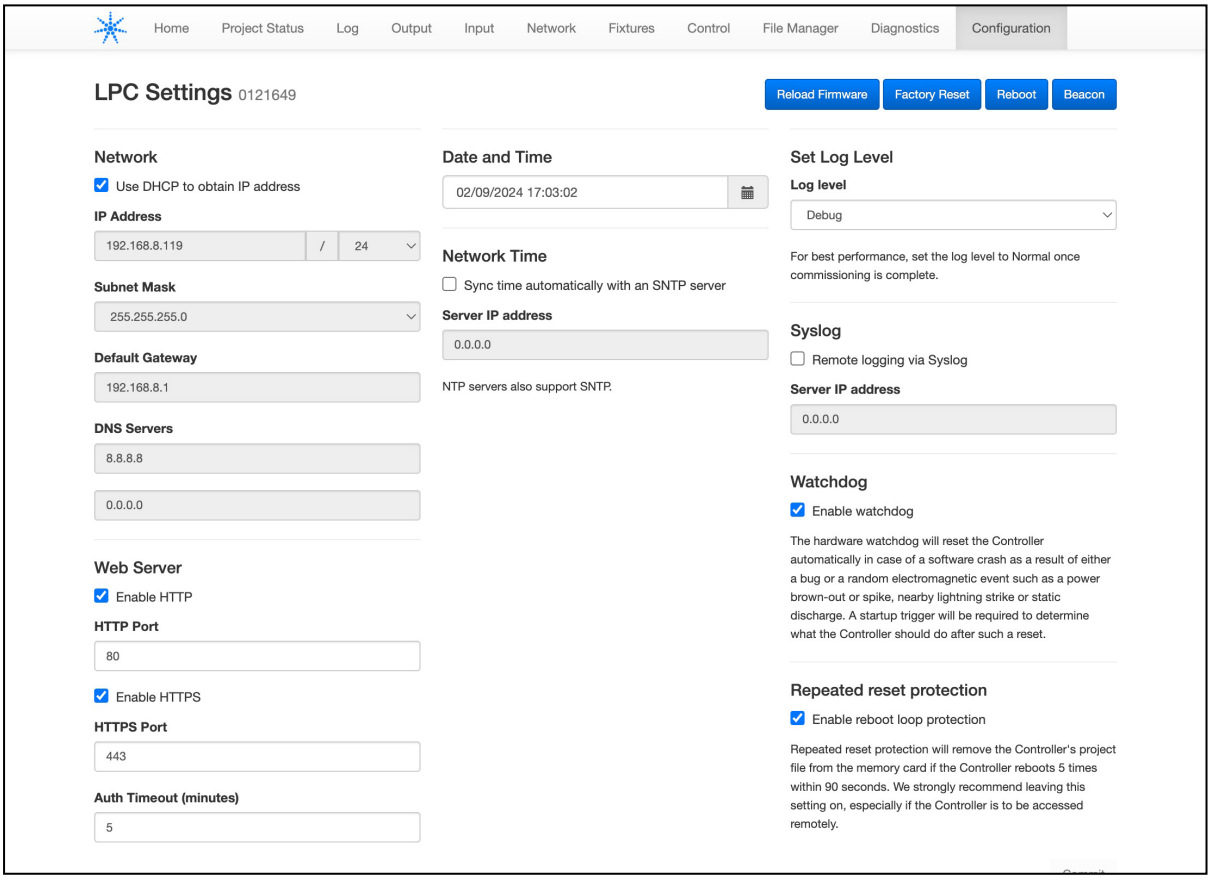
Allows simple diagnostic tasks to be carried out on the controller. The tools available are:

- Ping: Pings the specified IP address from the controller four times and displays the results in the box below.
- Packet capture: Record packets on the controller's network adapter to the controller's SD card (for a short period of time), or continuously stream the captured data to an external PC for recording (allowing longer data capture times).



Configuration

Allows you to change any of the controller configuration options as described in Controller configuration.



All the Controller's configuration settings are displayed and can be changed here, see configuration for details.

Remote upload

In addition, at the bottom of the page, is the means to upload a project file remotely via the web interface as an alternative to uploading directly from Designer. See the network section to learn how to generate a file for

remote uploading.

IMPORTANT: Controllers must be running the same version of firmware as the Designer software. Uploading a project file to a Controller running different firmware may result in the project failing to load and run. Check the Controller's home page to determine compatibility before attempting a remote upload.

Custom Page

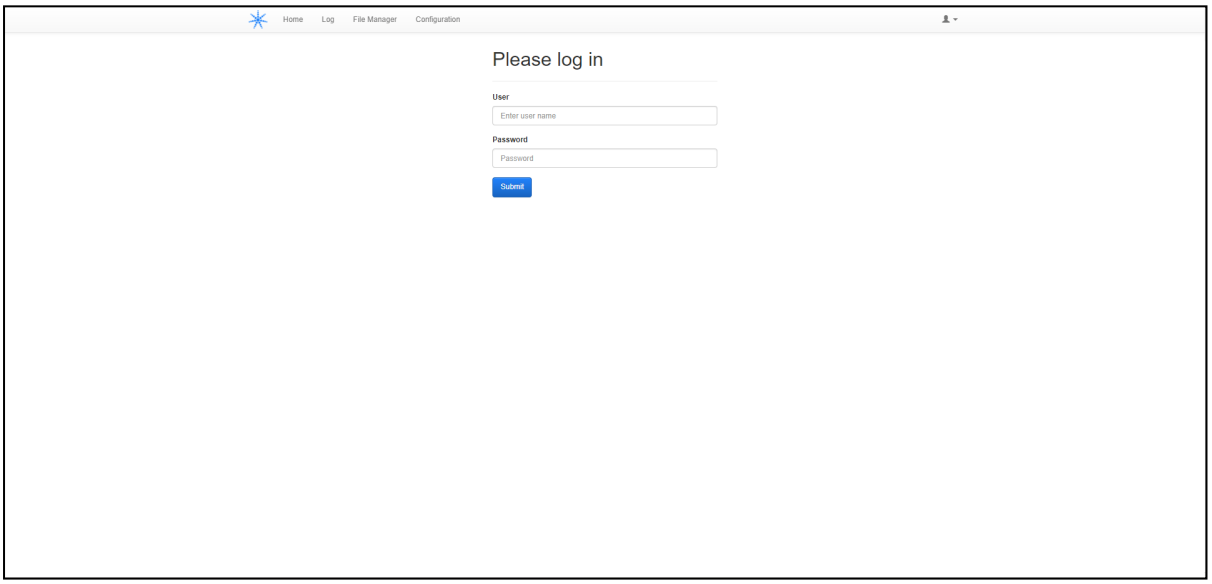
If a Custom Web Interface has been added, this can be accessed from the dropdown in the top right hand corner.

Log In

If the project has been configured with any user access accounts, then to access the web interface, users must use the Log In option in the top right.

This login uses the user names and passwords specified in the Project > Web Interface tab.

If authentication is active on a controller enter the username and password for a user account.



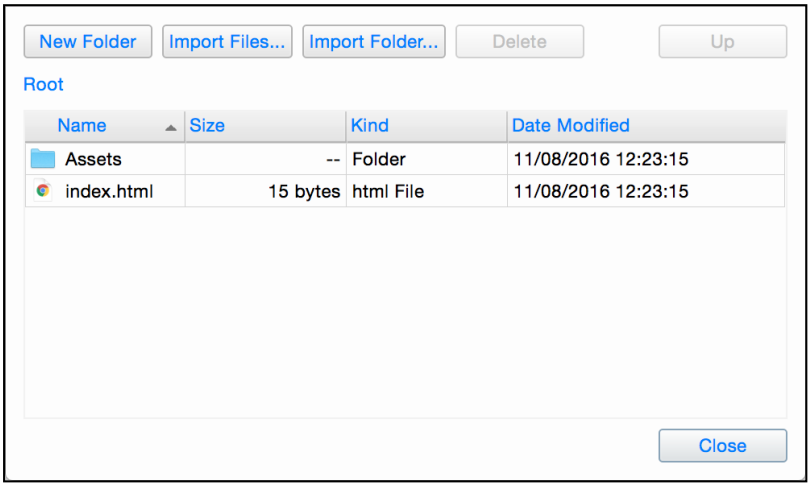
NOTE: The web interface uses Web Socket connections (RFC 6455), and some managed networks and proxy servers block these connections. If you don't see the web interface populating with data, please check whether your network is blocking web sockets

NOTE: Sometimes Ad-blockers can affect access to controllers with passwords set. If issues are seen, the controller's IP Address should be Whitelisted.

Custom Web Interface

To add a custom web page, or set of pages, to the web interface on a Controller, go to Project Mode, Web Interface Tab, Custom Web Interface and select Edit...

This will open a dialog that shows the files that currently make up your custom web interface:



Adding Files

To Add files to the web interface, click Import. This will allow you to add files to the current folder selected within the Custom Web Interface tool.

Any file type can be included, but since they are stored as part of the project file, be aware that they will take away from space available for programming, and will increase show upload times.

Adding Folders

To add sub-directories to the web interface structure, use the New Folder button. This will add the folder to the directory structure and allow you to add files to the new folder.

Removing files or folders

To remove a file or folder, select the file or folder and press Delete.

JavaScript Query Library

Alternatively, the controller's web server includes a JavaScript Query library which can be used to fire triggers and also to query the controller for information about its state and properties.

See the Controller API for more information.

Security Certificates

Recommended security configuration:

Security Certificates	all -SSLv3 -TLSv1 -TLSv1.1
SSLCipherSuite	ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-

	SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256
SSLHonorCipherOrder	on
SSLCompression	off

Examples

Examples of Custom Web Interfaces are available on our website.

Access Files

The information for .webconfig configuration and htaccess files has been added to the API Help, our online resource. This allows us to manage it better, allowing for a faster turn-around of tweaks and improvements to the documentation, as well as fixing minor errors.

The latest version can be found here.

Command Line

The Controller has a command line entry box in the Control page of its Web Interface.

By checking “Parse command line submissions as Lua Commands” (see Custom Command Line Parser), any Lua, including our API (see API v5), can be entered into this field and run during normal playback.

Alternatively, users may write their own scripts which can be used as a Custom Command Line Parse if they wish (see Lua scripts) but a standard script “commandline.lua” is available here.

IMPORTANT: Note that by default there is no command line script installed. Most installations will not require a command line and so it is inactive by default. If you wish to use the standard command line script you must use the “ Command Line Parser” section of the Web Interface section of Project Mode to select the script.

The command line syntax defined in the standard commandline.lua script has the following commands, where x,y and z represent numbers and [] indicates optional syntax:

Selections

x
x-y
x/y
x-y/z

where x, y and z are the fixture number, '-' selects a range and '/' combines discrete selections or ranges.

Setting intensity

x@y[%][tz]

where x is the fixture number, y is the level (either as a DMX value or as a percentage) and z is an optional time in seconds. If a time is not specified then it is treated as a snap change.

Examples:

1@127	Set intensity of fixture 1 to 127 immediately
2@50%	Set intensity of fixture 2 to 50% (127) immediately
3@100%t5.5	Set intensity of fixture 3 to 100% fading over 5.5 seconds

Setting RGB

Setting red, green and blue uses the same syntax as intensity, but replacing the @ with r for red, g for green, and b for blue.

Examples:

1r255	Set red of fixture 1 to 255 immediately
3g0	Set green of fixture 3 to 0 immediately
7b100%t2	Set blue of fixture 7 to 100% fading over 2 seconds

Note that the default values for red, green and blue are 100% (255) to give white. So to make a fixture output the colour red then you will need to set green and blue to zero.

You can also apply multiple settings to the same selection of fixtures in a single command. For example:

1-25@100%r255b255g0	Set fixtures 1 through 25 to 100% intensity, red to 255, blue to 255 and green to 0 immediately

Clearing fixture settings

xc[ty]

where x is the fixture number and y is an optional time in seconds.

Examples:

1c	Clear settings for fixture 1 immediately
5ct6.5	Clear settings for fixture 5 fading over 6.5 seconds

Clearing all fixtures settings

ca[tx]

where x is an optional time in seconds.

Examples:

ca	Clear settings for all fixtures immediately
cat10	Clear settings for all fixtures fading over 10 seconds

Multiple Commands

Multiple commands can be applied from a single command line if separated by commas.

Examples:

1@100%,1r0,1b0,1g255	Set intensity of fixture 1 to 100%, red and blue to 0 and green to 255
1ct5,4r255,4@75%t5	Clear settings for fixture 1 fading over 5 seconds, set red for fixture 4 to 255 immediately and then set intensity of fixture 4 to 75% fading over 5 seconds

Interaction with timeline playback

Settings applied from the command line are applied as if from a high priority timeline, so they will override all normal timeline programming until cleared. Fades to and from command line settings behave just like fades between timelines.

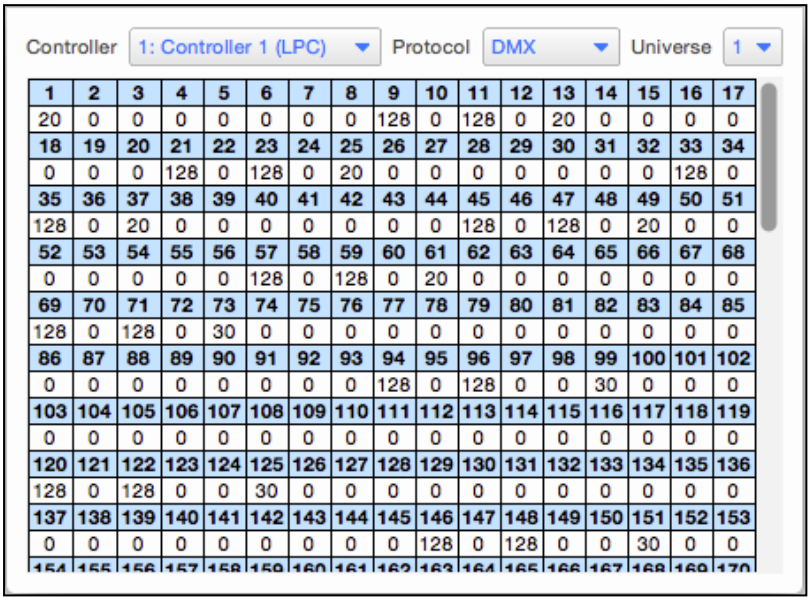
Main Menu Tools Overview 

The Main Menu contains several useful tools which can be used for troubleshooting and high level functions:

- Output Viewer
- Controller Log Viewer
- Import Object
- Export Object
- Preferences

Output viewer

Select Output viewer from the main menu to open this window:



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
20	0	0	0	0	0	0	0	128	0	128	0	20	0	0	0	0
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
0	0	0	128	0	128	0	20	0	0	0	0	0	0	0	128	0
35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
128	0	20	0	0	0	0	0	0	0	128	0	128	0	20	0	0
52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68
0	0	0	0	0	128	0	128	0	20	0	0	0	0	0	0	0
69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85
128	0	128	0	30	0	0	0	0	0	0	0	0	0	0	0	0
86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102
0	0	0	0	0	0	0	0	128	0	128	0	0	30	0	0	0
103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136
128	0	128	0	0	30	0	0	0	0	0	0	0	0	0	0	0
137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153
0	0	0	0	0	0	0	0	0	128	0	128	0	0	30	0	0

Use the Controller, Protocol and Universe pull-downs to select the Controller and DMX universe that you wish to view. What you will see depends on the status of your patch and the simulator:

Unpatched universe

If the universe is not patched then all values will be zero (0) regardless of the simulator's status.

Patched, simulator not running (reset)

If the universe is patched and the simulator is not running then you will see the default values for the fixtures. These are the values that the Controller will output after a reset or power cycle and prior to a timeline running, see Precedent.

Patched, simulator running (playing or paused)

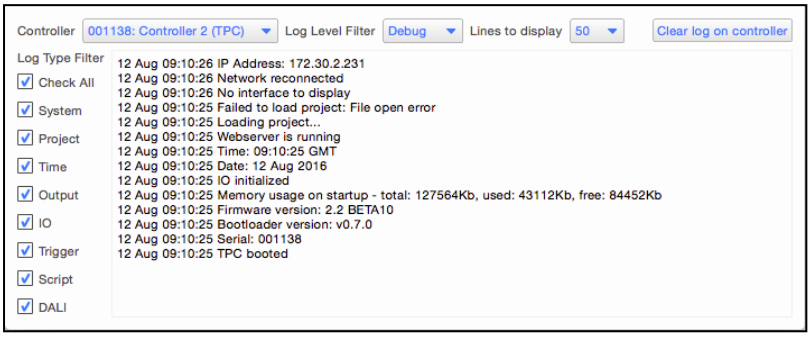
If the universe is patched and the simulator is running then you will see the values that the Controller will generate when it runs this timeline. Designer uses exactly the same playback algorithms as the Controller so what you see with the DMX viewer is what you'll get with the Controller.

Output Live

If you have connected Controllers then you will be able to select Output Live in the simulator to have Designer generate the DMX values directly, the Controllers acting purely as a DMX driver. In this case what you see in the DMX viewer is exactly what is being output to the fixtures in realtime.

Controller Log Viewer

Select Controller Log Window from the Main Menu to open this window:



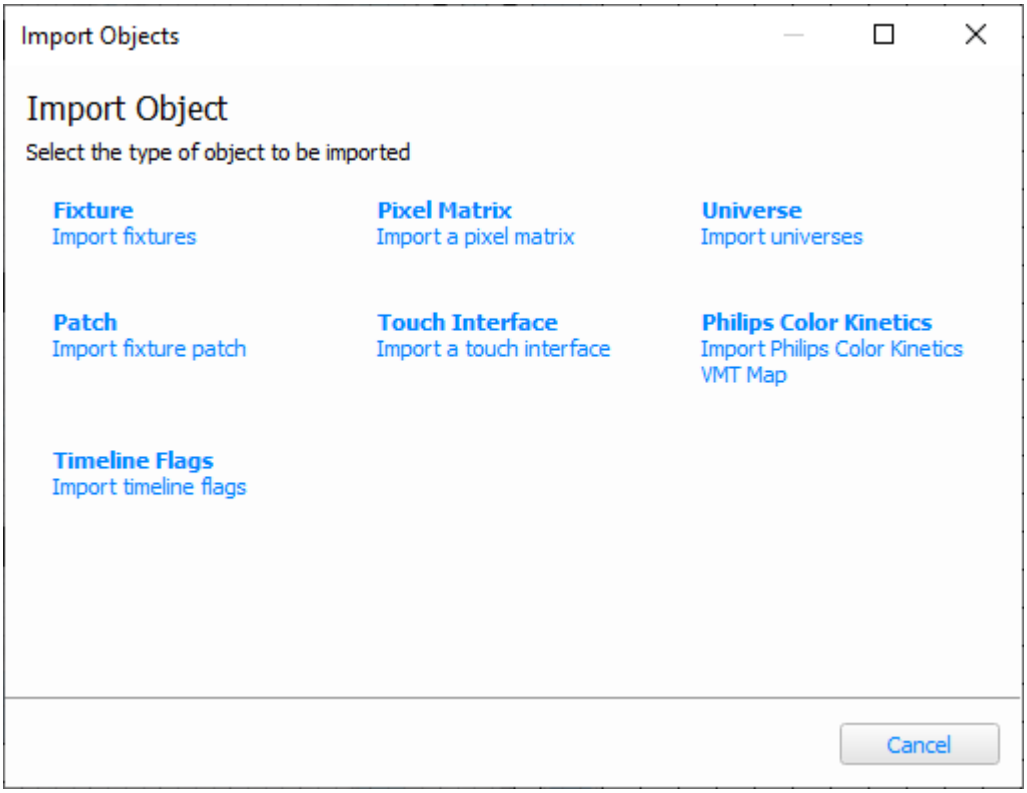
This windows displays the same information as you get on the Log page of the controller's web interface.

The log can be filtered by:

- Controller
- Log Level
- Log Type

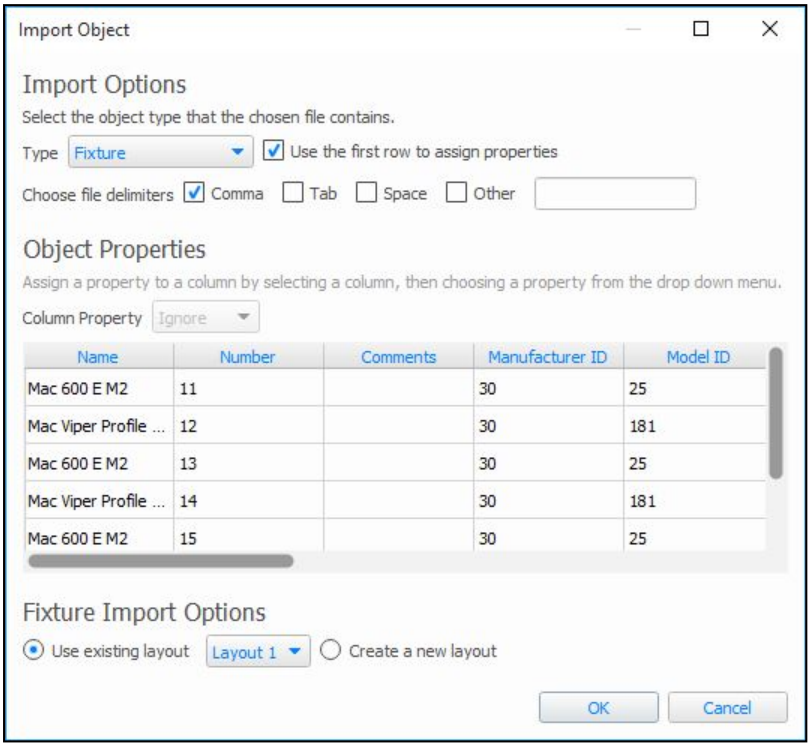
Import Objects Overview

The Import Objects option in the Main Menu can be used to import a .csv, .tsv or .txt file which defines a Fixture Layout, Pixel Matrix, Universes, Patch record, a .ptc or .dat file to import a TPC Interface or a .fap file to import a Philips Color Kinetics VMT Map.



During Import, if the file import has errors, the file can be updated externally. If this is done, then a Reload File button will become available.

Fixture



A Fixture Layout file should include the following required data:

- Number - a unique fixture number
- Manufacturer ID - the manufacturer number for the required fixture (can be found in the fixture configuration)
- Model ID - the model number for the required fixture (can be found in the fixture configuration)

NOTE: These columns are required to import the fixture layout. X and Y position are required columns, but can be left empty for fixtures. This will allow you to add the fixture to the project, but not the layout.

The following columns are optional, and if not provided, the default will be used.

- Name - A name for the fixture
- Groups - Allows importing fixtures into a specific group.
  - This takes the format "{number,'Group Name'}{number2,'Group Name2'}"
- Comments - A comment about the fixture
- Mode ID - the mode number for the required fixture (can be found in the fixture configuration)
- Width - the width of the fixture on the layout
- Height - the height of the fixture on the layout
- X - the x-position of the fixture on the layout
- Y - the y-position of the fixture on the layout
- Angle - the angle of the fixture on the layout, clockwise from vertical.

The following can be set for any columns to ignore them.

- Ignore - use if there is data in the text file which should be ignored

Importing Fixtures with no position

You can import a fixture without any position information. This will create the fixture within the project, but won't place an instance of the fixture on a layout.

NOTE: The text file still requires X and Y columns to import.

Importing Custom Properties

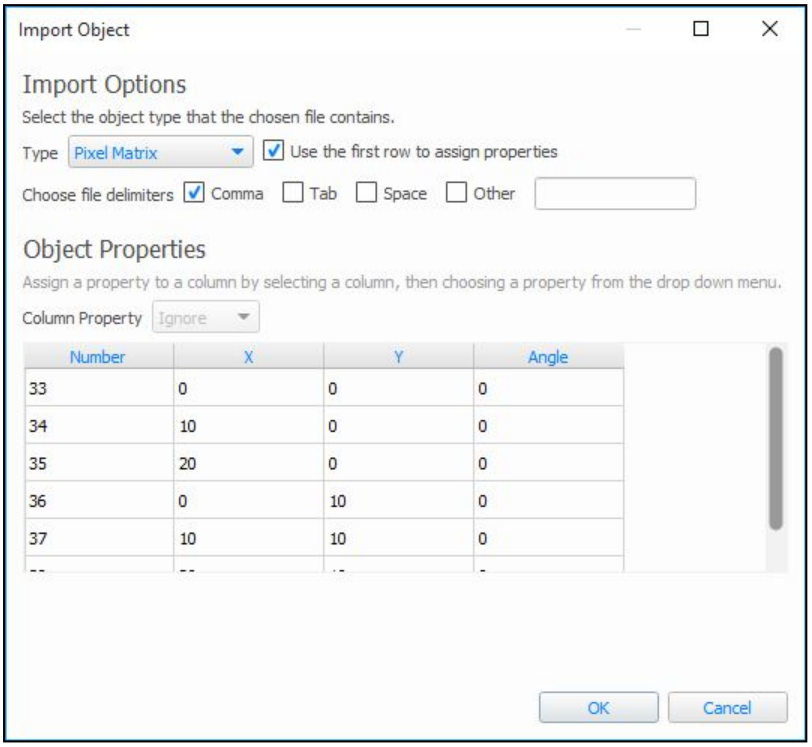
If you have Custom Properties setup within your project, these can be imported by adding a column to your text file containing this data. The Custom Property will be available in the Column Property drop down.

Example

The data below will import 4 LED - RGB 8-Bit fixtures in a line and add them to various groups.

```
Fixture number,Groups,Manufacturer ID,Model ID,X,Y
1,"{1,'Group 1'}{4,'Group\'s Group'}",0,5,960,960
2,"{2,'My 2nd Group'}{4,'Group\'s Group'}",0,5,1440,960
3,"{3,'Group with \ in'}{4,'Group\'s Group'}",0,5,1920,960
4,"{4,'Group\'s Group'}",0,5,0,240,240,2400,960
```

Pixel Matrix



A Pixel Matrix file should include the following required data:

- Number - the fixture number of the required fixture within the project
- X - the x coordinate within the pixel matrix
- Y - the y coordinate within the pixel matrix

The following columns are optional

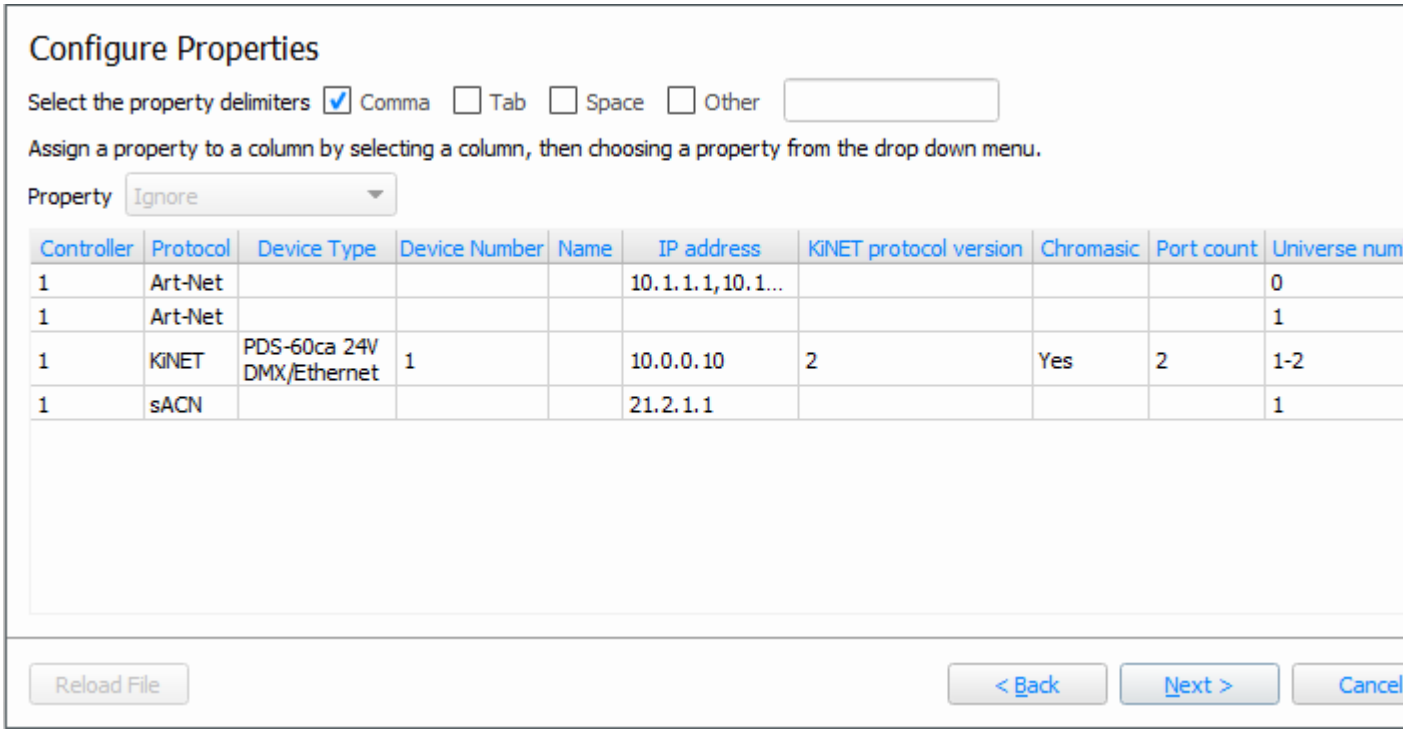
- Angle - the angle of a compound fixture within the pixel matrix
- Ignore - columns within the file which can be ignored

Example

The data below will create a Pixel Matrix with fixtures 1-7 in a vertical line.

```
Number,X,Y
1,1,2
2,1,3
3,1,4
4,1,5
5,1,6
6,1,7
7,1,8
```

Universes



Universes, with their properties, can be imported and exported from a Designer project. This allows for a quick changing of properties via exporting a universe, opening the .csv export file outside of Designer and making the desired change, then reimporting. A good example of this is with setting a unicast IP address to a batch of Art-Net universes or altering KiNET power supply information.

The following headers are required for all protocols:

- Controller
- Protocol
- Universe number

For KiNET, the following is also required:

- Device Types*:

  PDS_150e
  PDS_500e
  PDS_60 24V Ethernet
  PDS_60_24V DMX/Ethernet
  PDS_60ca 12V DMX/Ethernet
  PDS_60ca 7.5V DMX/Ethernet
  PDS_70mr 24V Ethernet
  sPDS_60ca 24V DMX/Ethernet
  sPDS_480ca 7.5V
  sPDS_480ca 12V
  sPDS_480ca 24V

Data Enabler Ethernet
iColor Accent (Data Enabler EO)
PDS_60ca 24V Ethernet
Data Enabler Pro
ColorBlaze TRX
Multi-Protocol Converter

Other types can be used to create custom power supplies.

The following columns are optional, and if not set, will use a default value:

- Name - A human readable name for the power supply (KiNET only)
- IP Address - the IP address of the power supply or universe (Art-Net, sACN or KiNET)
  - If no IP is set for Art-Net or sACN, the universe will default to broadcast
  - To assign multiple IP addresses to the same universe, ensure the IP addresses are in quotes, such as "1.1.1.1,2.2.2.2,3.3.3.3"
- Ports - The number of ports that this power supply has
- Device Number - The user number for this power supply
- Chromasic* - (Yes or No) - Whether the power supply is Chromasic
- Protocol Version* - The KiNET version that the power supply uses

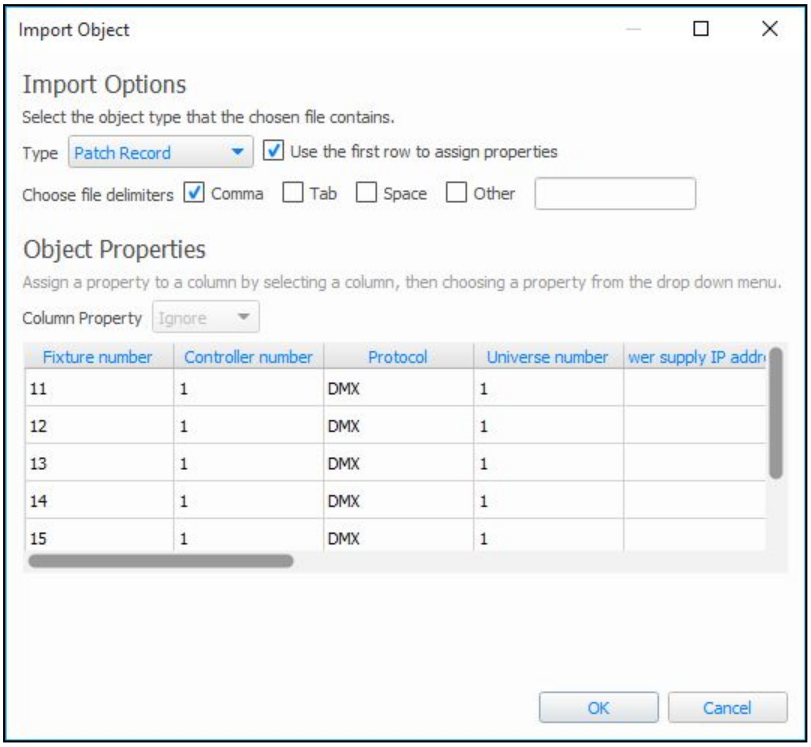
* These can be used to import custom power supplies into the project

Example

The .csv copy below will import Art-Net universe 0 with Unicast set to two IPs, Art-Net universe 1 set to Broadcast, a KiNET Power Supply “PDS-60ca 24V DMX/Ethernet” and an sACN universe with Unicast set to a single IP.

```
Controller number, Protocol, Device type, Device number, Name, IP address,
KiNET protocol version, Chromasic, Port count, Universe number

1,Art-Net,,,,"10.1.1.1,10.1.1.10",,,,0

1,Art-Net,,,,,,,,1

1,KiNET,PDS-60ca 24V DMX/Ethernet,1,,10.0.0.10,2,Yes,2,1-2

1,sACN,,,,21.2.1.1,,,,1
```

Patch



A patch record file should include the following required data:

- Fixture Number - the user number of the fixture
- Controller Number - the controller the fixture is patched to
- Protocol - the protocol to patch to
- Universe Number - The universe of the specified protocol to patch to. If using Art-Net, the universe can be set in three part format (a/b/c = Net/Sub-Net/Universe)
- Channel - the channel to patch to

The following data is optional, depending on the protocol/fixture being used.

- Power Supply IP Address - The IP Address of the the KiNET power supply (if using KiNET)
- Port - the port on the KiNET power supply to patch to (if using KiNET)
- Patch point - the patch point of the fixture (if using a fixture with multiple patch points)
- sACN Priority
- RIO Device Number - the address of the RIO to patch to (if using RIO DMX)
- RIO device type - the type of the RIO to patch to (if using RIO DMX)

Example:

```
Fixture number,Controller number,Protocol,Universe number,Power supply IP
address,Power supply user number,Port,Channel,Patch point,sACN priority,sACN
universe priority,RIO device number,RIO device type
1,1,DMX,1,,,,1,Fixture,,,,
2,1,DMX,1,,,,4,Fixture,,,,
3,1,DMX,1,,,,7,Fixture,,,,
4,1,DMX,1,,,,10,Fixture,,,,
5,1,DMX,1,,,,13,Fixture,,,,
6,1,DMX,2,,,,1,Fixture,,,,
7,1,DMX,2,,,,4,Fixture,,,,
```

```
8,1,DMX,2,,,,7,Fixture,,,,
9,1,DMX,2,,,,10,Fixture,,,,
10,1,DMX,2,,,,13,Fixture,,,,
11,1,Art-Net,0,,,,1,Fixture,,,,
12,1,Art-Net,0,,,,2,Fixture,,,,
13,1,Art-Net,0,,,,3,Fixture,,,,
14,1,Art-Net,0,,,,4,Fixture,,,,
15,1,Art-Net,0,,,,5,Fixture,,,,
16,1,Art-Net,0,,,,6,Fixture,,,,
17,1,Art-Net,0,,,,7,Fixture,,,,
18,1,Art-Net,0,,,,8,Fixture,,,,
19,1,Art-Net,0,,,,9,Fixture,,,,
20,1,Art-Net,0,,,,10,Fixture,,,,
21,1,KiNET,,10.1.2.3,1,1,1,Fixture,,,,
22,1,KiNET,,10.1.2.3,1,1,5,Fixture,,,,
23,1,KiNET,,10.1.2.3,1,1,9,Fixture,,,,
24,1,KiNET,,10.1.2.3,1,1,13,Fixture,,,,
25,1,KiNET,,10.1.2.3,1,1,17,Fixture,,,,
26,1,KiNET,,10.1.2.3,1,2,1,Fixture,,,,
27,1,KiNET,,10.1.2.3,1,2,5,Fixture,,,,
28,1,KiNET,,10.1.2.3,1,2,9,Fixture,,,,
29,1,KiNET,,10.1.2.3,1,2,13,Fixture,,,,
30,1,KiNET,,10.1.2.3,1,2,17,Fixture,,,,
```

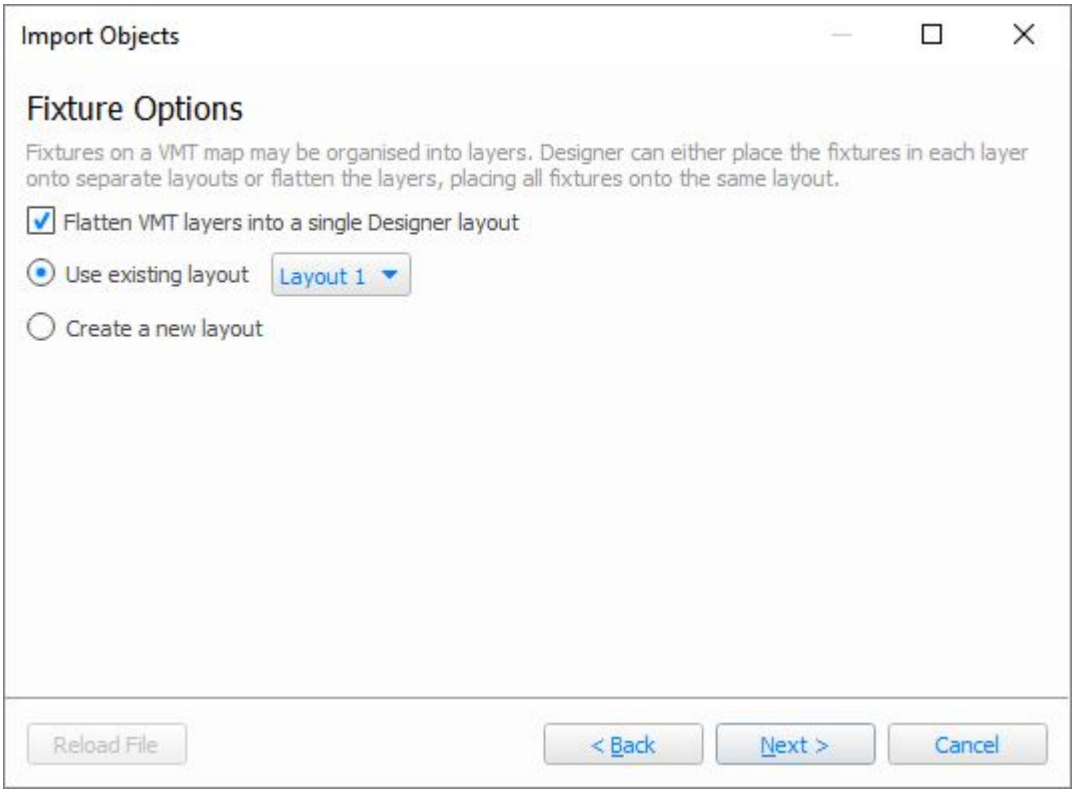
TPC Interface

You can import any TPC Interface that has been exported from Designer 2, as a .dat file or any TPC Interface created using Interface Editor as a .ptc file.

The selected interface will be imported into Designer and added to the project.

You can then go to the Interface Mode to edit it, and associate it with a controller in the project.

Philips Color Kinetics



A VMT map from Philips Color Kinetics Video Management Tool 2 can be imported to bring the fixture map and patch into Designer.

Select Philips Color Kinetics from the Import Object Dialog and browse to your .fap file.

VMT layers can be mapped onto different Layouts in Designer, or flattened onto a single layer.

If you select a VLC Layout, the fixtures will be added to the associated VLC, otherwise, select the controller to patch the fixtures to.

The fixtures in the VMT Map will be brought into Designer, mapped onto the selected layout/s and patched according to the VMT file.

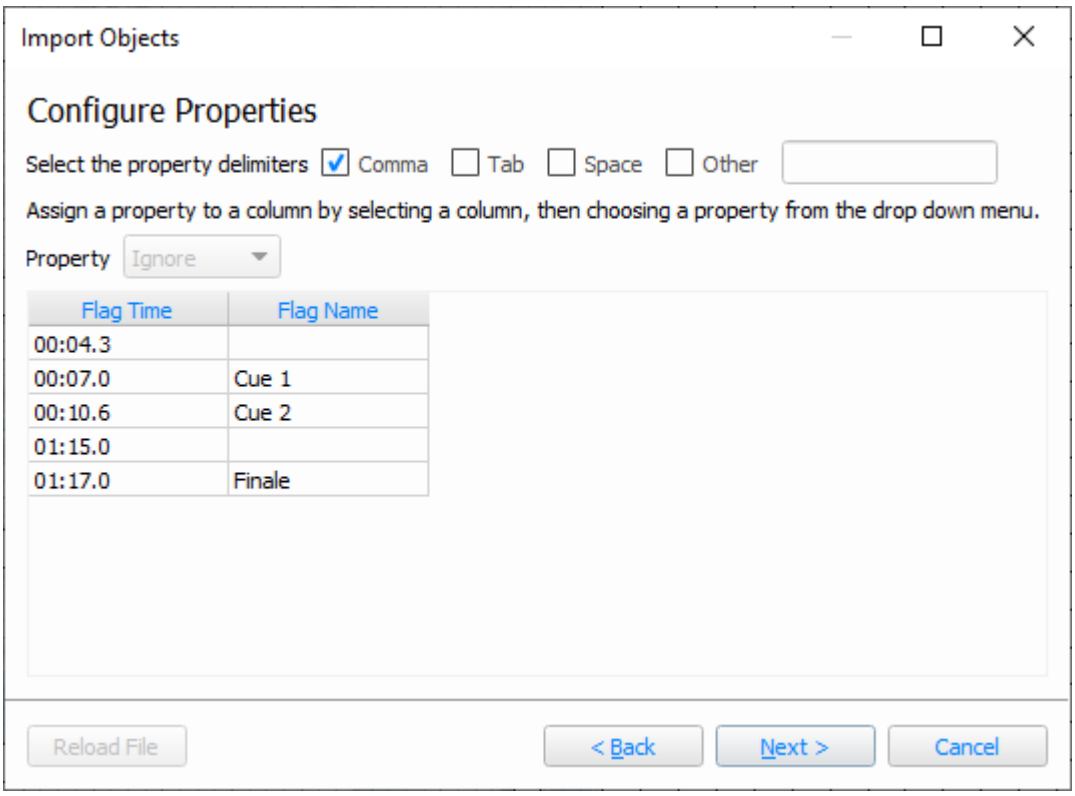
If the KiNET power supplies in the VMT project don't exist in Designer, and are of a known type, they will be added to the project.

PDS_150e
PDS_500e
PDS_60 24V Ethernet
PDS_60_24V DMX/Ethernet
PDS_60ca 12V DMX/Ethernet
PDS_60ca 7.5V DMX/Ethernet
PDS_70mr 24V Ethernet
sPDS_60ca 24V DMX/Ethernet
sPDS_480ca 7.5V
sPDS_480ca 12V
sPDS_480ca 24V

Data Enabler Ethernet
iColor Accent (Data Enabler EO)
PDS_60ca 24V Ethernet
Data Enabler Pro
ColorBlaze TRX
Multi-Protocol Converter

NOTE: Only original .fap files can be use, not exported map files

Timeline Flags



A Timeline Flag import csv require the following required data:

- Flag Time - The time position of a flag
    - This takes the format minutes:seconds.milliseconds

The following columns are optional:

- Flag Name - The name for a flag at the specified time

During the import, choose to create a new timeline to import the flags into, or use an existing timeline. Selecting a new timeline will add a new timeline to the project with default settings.

Selecting to use an existing timeline will provide the options to merge with the current flags in the selected timeline or replace them.

- Merge will retain all existing flags in a given timeline, and if a flag doesn't exist at the time of a new flag one will be added. If a flag does exist at the same time the new flag will replace the existing flag and name data will be updated to match the import. Triggers which use these existing flags will keep their configuration and the flag will remain linked.
- Replace will delete all flags in a timeline and then import the new flags. Triggers which use the original flags will lose their configuration as the flag has been deleted and replaced, even if the new flag is at the same time.

Example

The data below will import 5 flags at the specified times with optional naming.

Time,Name

00:04.3,

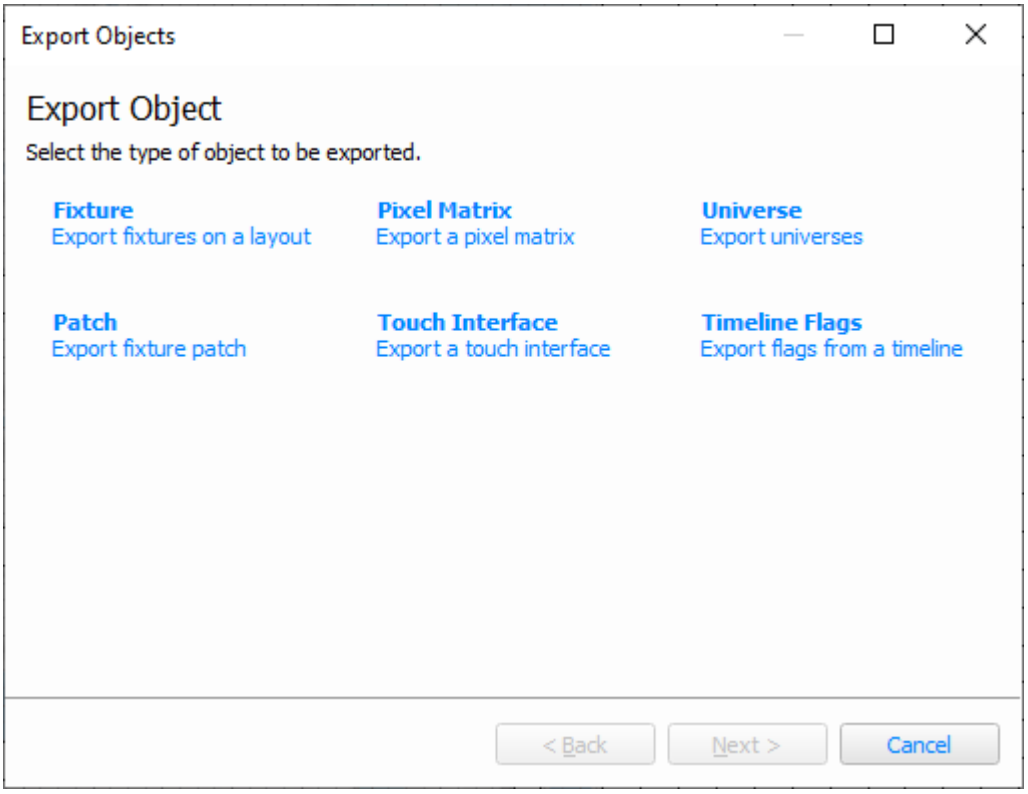
00:07.0,Cue 1

00:10.6,Cue 2

01:15.0,

01:17.0,Finale

Export Object



The Export Object option in the Main Menu can be used to export a .csv file which defines a Fixture Layout, Pixel Matrix, Patch Record or list of KiNET power supplies

When you export a text file, you must specify which type of object it refers to.

You must then select which object you want to export, i.e. which Layout or Pixel matrix should be exported.

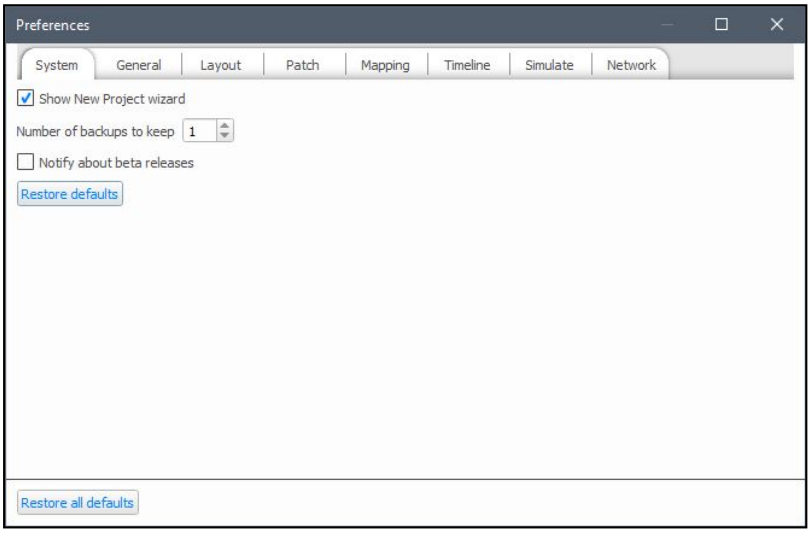
Finally, select the data fields that you want to output to the export file.

Preferences

Select  Main Menu > Preferences on the main toolbar to open the Preferences dialog:

System

Select the System tab to change the default behaviour of Designer:



Show New Project wizard

Choose whether to display the New Project Wizard when a new project is created.

Number of backups to keep

Designer can keep a number old versions of the project file when you save and it is here that you set the number of old files to keep. Before saving your project (Save Project or Ctrl+S), Designer will rename the project file on disk by adding the current time and date to the file name, such as "my_project_bak_2007-04-18_15-58-09.pd2". If you already have the specified number of backups, the oldest backup will be removed from the disk.

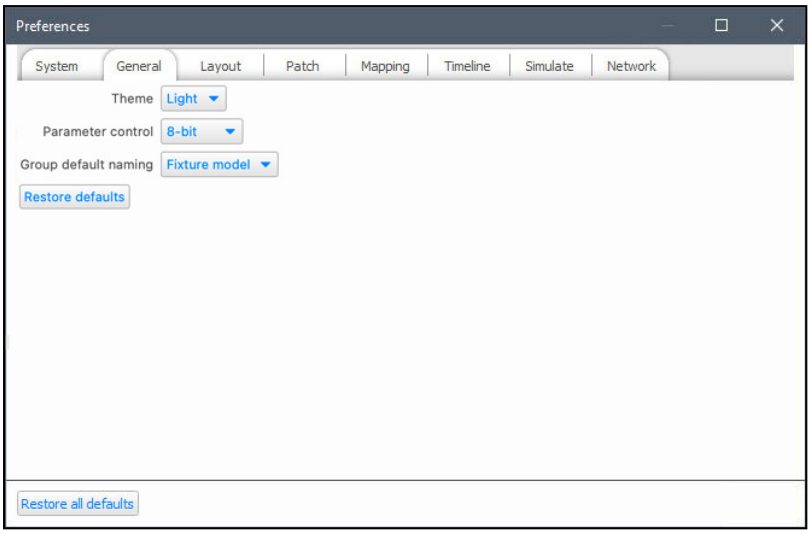
Use Save Project As to produce manual backups of the project at each important programming milestone.

Notify about beta releases

Choose whether to be notified when Pharos release a Beta version.

NOTE: You will always be notified when a stable version is released.

General



Theme (Dark Mode)

Dark mode is an alternative look to the Pharos Designer application with a darker background; ideal for working in low-light environments. It can be enabled by changing the **Theme** dropdown to “Dark”.

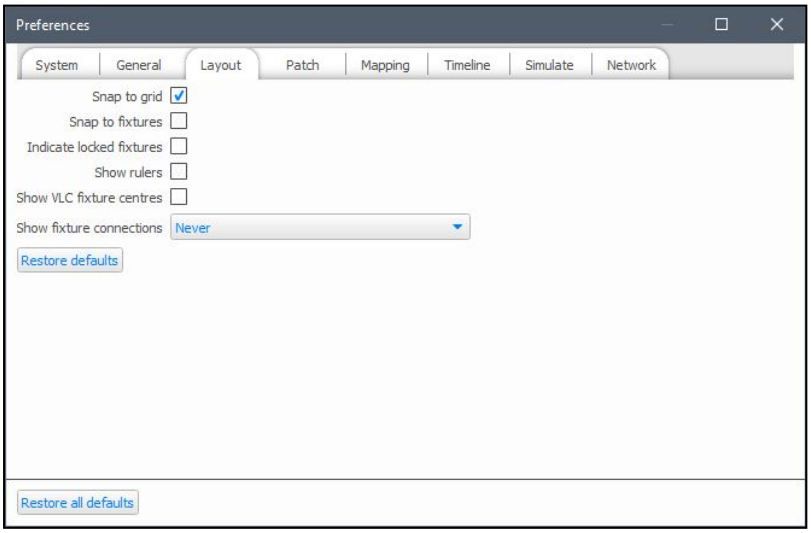
Parameter Control

Which model do you want to use (8 bit = 0->255, Percent = 0->100%), this is a display option only. The editors within Timeline and Scene with display levels in this format.

Group default naming

Choose the default name given to new groups. Select from Number or Fixture model.

Layout



Snap to grid

Automatically snap the centre of a fixture to a grid intersection.

Snap to fixtures

When two fixtures are brought close together, the sides automatically align.

Indicate locked fixtures

Display an icon when you try and drag a locked fixture icon on the Layout

Show rulers

Display a horizontal and vertical scale along the edges of the Layout to place fixture accurately.

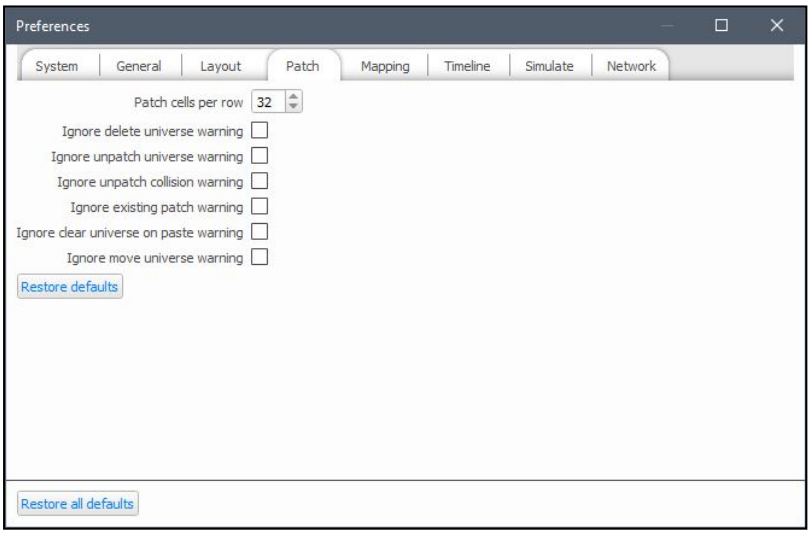
NOTE: The above preferences are also available in the right-click context menu within the Layout view.

Show VLC fixture centres

Display a circle to indicate where the actual VLC fixture is. This is them mapped onto the relevant pixel.

Patch

Select the Patch tab to change the default settings for patching:



Patch Cells per row

The Patch Cells per row entry box lets you determine how many channels are displayed per row. This can be useful for organising the display for complex fixtures; set this number to be a multiple of the number of channels a fixture uses to get a neater, tabulated display.

Ignore delete universe warning

Choose whether to display a warning when a universe is deleted.

Ignore unpatch universe warning

Choose whether to display a warning when a whole universe is unpatched.

Ignore unpatch collision warning

Choose whether to have a warning displayed when patching collisions occur.

Ignore existing patch warning

Choose whether to display a warning when a fixture is patched multiple times.

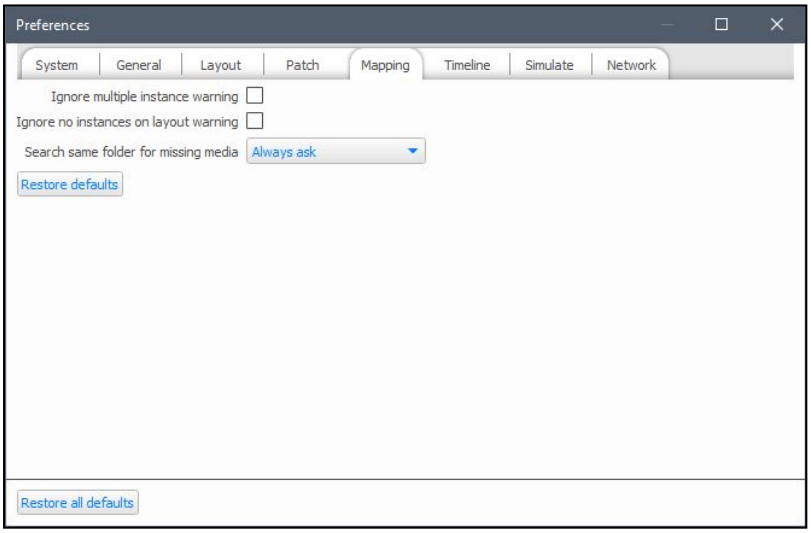
Ignore clear universe on paste warning

Choose whether to display a warning when you paste a copied universe into a universe that already has fixtures patched to it.

Ignore move universe warning

Choose whether to display a warning when a KiNET port is patched from a different controller to the rest of the power supply.

Mapping



Ignore multiple instance warning

Choose whether to display a warning when a pixel matrix is created which includes multiple instances of one fixture.

Ignore no instances on layout warning

Choose whether to display a warning when attempting to create a pixel matrix with fixture that aren't on the current layout.

Ignore additional VLC+ targets warning

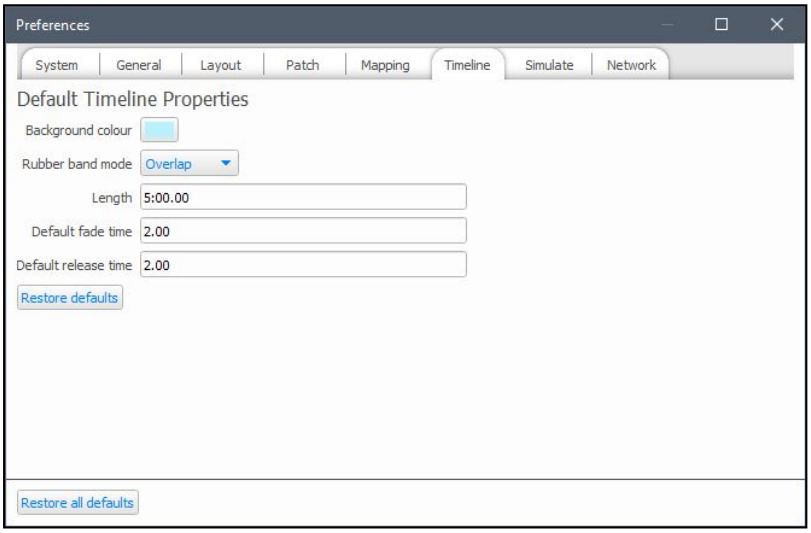
Choose whether to display a warning when enabling Additional Targets on a VLC+.

Search same folder for missing media

When you replace missing media, Designer can search the same folder for other missing media clips.

Timeline

Select the Timeline tab to change the default settings for timelines:



These properties will be applied to any timelines created after the properties are changed, and can be overridden on a per timeline basis.

Background Colour

The background colour of the timeline area of the Timeline window can be chosen here, press the button and select a colour. This is useful to make certain types of programming stand out better, for example a project using mainly intensity presets may be clearer with a dark grey background.

Rubber Band Mode

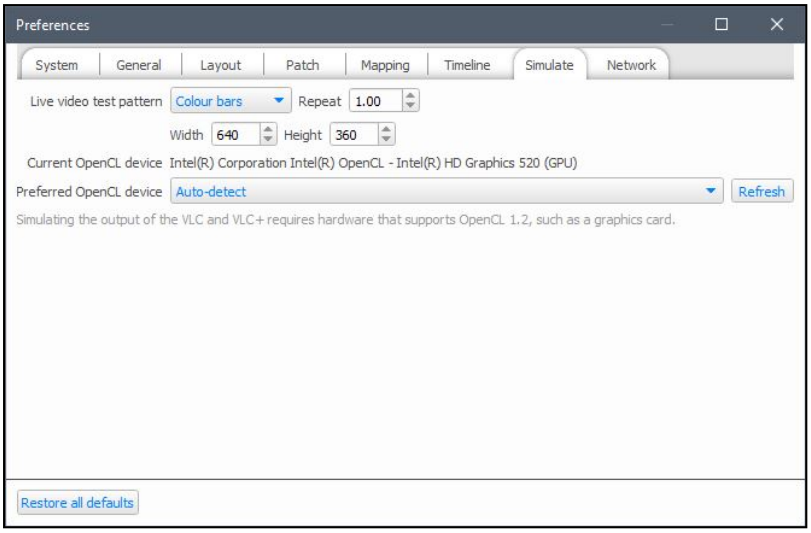
This preference determines how presets should behave when using a rubber band to select them (dragging a box around the presets with the mouse). Overlap will select anything the box touches, whereas Encompass will select only presets wholly enclosed by the box.

Default Timeline Properties

Specify the default length, fade and release times, see timeline properties.

Simulate

Select the Simulate tab to change the default settings for Simulate Mode:



Live video test pattern

These settings allow you to specify the output when a live video input is not being received:

- Pattern - the pattern to display (Colour bars, vertical lines, horizontal lines, grid)
- Repeat - the number of times to repeat the pattern on the output (for colour bars)
- Size - the pixel size of the bar/grid
- Width - the width of the pattern
- Height - the height of the pattern

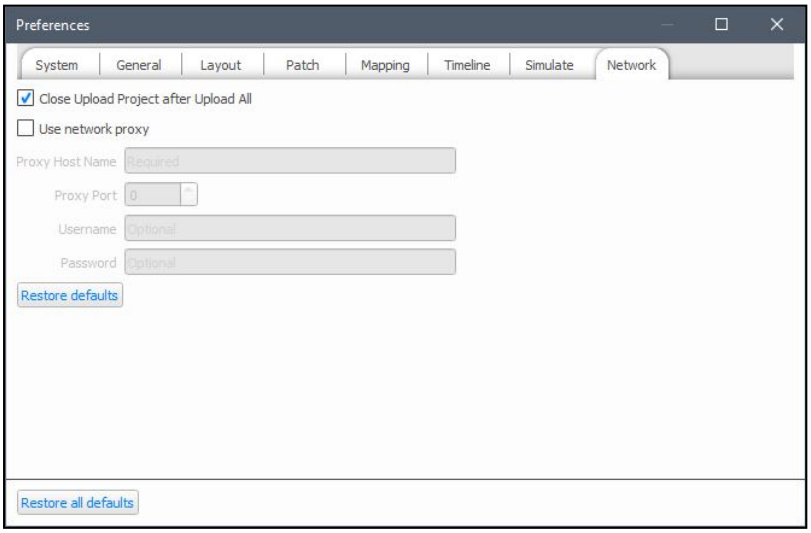
OpenCL device

Use the dropdown to choose hardware (CPU or GPU) that supports OpenCL 1.2. This is only required for VLC or VLC+ simulation.

NOTE: If None is selected, VLC or VLC+ simulation will not be available.

Network

Select the Network tab to change the default settings for networking:



Close Upload Project after Upload All

Specify whether the Upload Project dialog box should be closed when Upload All has completed

Transfer to controllers after Upload to Remote Site

Check this checkbox to automatically transfer the uploaded project to the controller when uploading via the Remote Site Upload feature.

Network Proxy Settings

These Network Proxy settings should be configured when a Proxy server is in use between your computer and the internet and/or your Pharos Controller/s.

Scripting Overview

There are two areas within Designer which utilise scripting, Triggers and Custom Presets.

NOTE: The VLC and VLC+ do not support Custom Presets and thus these will not be available for projects using these controllers.

These scripting environments are a powerful way to increase the functionality of a project beyond the capabilities of the existing Trigger, Condition and Action logic and the standard timeline presets.

Before using scripting within you project, read through the appropriate scripting guide for an overview of the scripting syntax and abilities.

Trigger script programming guide

Custom preset programming guide

There is also a Pharos specific extension to the standard Lua scripting language available which is described in Lua API (Triggering).

Supports Lua v5.3.4.

Custom Preset Programming Guide

Custom Presets use a Lua script to define an effect that can be played back on a Matrix. You can use this to create effects that are not available as standard in Designer. Custom Presets are managed using the Mapping window.

NOTE: The VLC and VLC+ do not support Custom Presets and thus these will not be available for projects using these controllers.

Basics

Custom presets use Lua scripts to define an animation.

For each pixel (x,y) of each frame of that animation, a `pixel` function is called which returns three numbers, between 0 and 255, which represent the red, green and blue components of the colour of that pixel. Pixel (0,0) is in the top left of the frame, with the positive x axis pointing right and the positive y axis pointing down.

Here is the most simple example of a custom preset:

Listing 1

```
function pixel(frame,x,y)
    return 255,0,0
end
```

This fills every pixel of every frame with red. If you do not return all three components of the pixel's colour, the missing components are assumed to be 0, so the following function is equivalent to Listing 1:

Listing 2

```
function pixel(frame,x,y)
    return 255
end
```

A real example

To demonstrate what can be achieved with custom presets, we are going to build up a real example as concepts are introduced throughout this guide.

To start, we are going to create a preset that renders a series of vertical red bands:

Listing 3

```
-- width of the bands in pixels
band_width = 4
-- space between bands in pixels
band_spacing = 1
-- modulo operator (a%b)
function mod(a,b)
    return a - math.floor(a/b)*b
end

-- the pixel function
function pixel(frame,x,y)
    -- use the modulo operator to split the horizontal axis into bands and
```

```
    -- decide if we are in the band or in the separator between bands
    if (mod(x,band_width+band_spacing)<band_width) then
        -- in band
        return 255,0,0
    else
        -- in band separator
        return 0,0,0
    end
end
```

You will note that we have defined a new function, `mod`, to implement the modulo operator. This was done to make the script more readable. We will discuss user-defined functions again later.

We also defined two variables, `band_width` and `band_spacing`. These we placed outside of the `pixel` function because they are the same for every pixel of every frame of the effect, so it is more efficient to not execute the assignment for every pixel. Any code outside of the pixel function is executed once, before the `pixel` function is called for the first time.

Animation

Filling every frame of an animation with a single colour is not very exciting, so we can use the `frame` argument to change the colour of a given pixel (x,y) based on the current frame.

Here is an example:

Listing 4

```
function pixel(frame,x,y)
    if (x<frame) then
        return 255,0,0
    else
        return 0,0,0
    end
end
```

This creates a red horizontal wipe, advancing 1 pixel towards the right for each frame. You may have noted that once the wipe reaches the right side of the frame, the whole frame stays red for a period of time before the animation loops back to the beginning. This is because the number of frames exceeded the number of pixels across the frame.

Ideally, we want our effects to loop seamlessly. To do this, we introduce three global variables that have been already been defined for you:

- `frames` - the total number of frames in the animation
- `width` - the width of the animation in pixels
- `height` - the height of the animation in pixels

We can rewrite Listing 4 as follows:

Listing 5

```
function pixel(frame,x,y)
    -- calculate the progress through the animation
    local t = frame/frames
    -- compare the fraction across the effect with the animation progress
    if (x/width<t) then
```

```
        return 255,0,0
    else
        return 0,0,0
    end
end
```

Now, once the red wipe reaches the right side of the frame, it immediately jumps back to the start. Returning to our vertical band example, we are going to introduce animation by changing the height of each band over time:

Listing 6

```
-- width of the bands in pixels
band_width = 4
-- space between bands in pixels
band_spacing = 1

-- get the combined width of band and separator
local total_band_width = band_width+band_spacing
-- get the number of visible bands
local bands = width/total_band_width

-- modulo operator (a%b)
function mod(a,b)
    return a - math.floor(a/b)*b
end

-- the pixel function
function pixel(frame,x,y)

    if (mod(x,total_band_width)>=band_width) then
        -- in band separator
        return 0,0,0
    end

    -- get the band in which this pixel falls
    local band = math.floor(x/total_band_width)

    -- get the fraction through the effect
    local t = frame/frames

    -- get the height of the band in which this pixel falls
    local band_height = (math.sin((band/bands+t)*math.pi*2)+1)/2

    -- adjust y to be relative to the center of the effect
    y = y-(height/2)+0.5

    -- decide if this pixel is inside the band
    if (math.abs(y)/(height/2) <= band_height) then
        return 255,0,0
    else
        return 0,0,0
    end
end
```

We are using a sine function to set the height of each band, where the argument to the sine function is offset based on the index of the band and the current fraction through the effect. The result of this is that the height of each band differs from its neighbour according the sine function, and this relationship is modified over time to create a ripple.

More colours than just red

So far, we have just been creating red effects, but there are more colours than red, so why should we stick with that? We will modify the vertical band example to show how different colours can be created. For this example, we introduce the built-in function, `hsi_to_rgb`, which converts an HSI (hue, saturation, intensity) colour into an RGB (red, green, blue) colour:

Listing 7

```
-- width of the bands in pixels
band_width = 4
-- space between bands in pixels
band_spacing = 1

-- get the combined width of band and separator
local total_band_width = band_width+band_spacing
-- get the number of visible bands
local bands = width/total_band_width

-- modulo operator (a%b)
function mod(a,b)
    return a - math.floor(a/b)*b
end

-- rainbow lookup
function rainbow(hue)
    return hsi_to_rgb(hue*math.pi*2,1,1)
end

-- the pixel function
function pixel(frame,x,y)

    if (mod(x,total_band_width)>=band_width) then
        -- in band separator
        return 0,0,0
    end

    -- get the band in which this pixel falls
    local band = math.floor(x/total_band_width)

    -- get the fraction through the effect
    local t = frame/frames

    -- get the height of the band in which this pixel falls
    local band_height = (math.sin((band/bands+t)*math.pi*2)+1)/2

    -- adjust y to be relative to the center of the effect
    y = y-(height/2)+0.5

    -- decide if this pixel is inside the band
    local h = math.abs(y)/(height/2)
    if (h <= band_height) then
        return rainbow(band/bands+t)
    else
        -- offset hue by quarter
        return rainbow((band/bands+t)+0.25)
```

```
    end
end
```

We have defined a new function, `rainbow`, which returns a fully saturated r,g,b value for a given hue. This function is then called with different arguments depending on whether on not a pixel falls inside or outside of a band.

User-defined functions can be used whenever you want to use a similar piece of code in multiple places with differing arguments.

Running this script, you will see that the bands are now coloured with a rainbow which changes over time, and the area above and below the band is filled with a colour that is pi/2 radians out of phase with the band's colour.

Working with colours

Working with colours as 3 separate components can produce a wide variety of effects, but sometimes it is more convenient to treat a colour as a single entity. We can do that with the colour library.

To create a variable of type colour, call `colour.new()`, passing in three values between 0 and 255 which represent the red, green and blue components of the colour, i.e:

```
local c = colour.new(255,0,0)
```

The variable `c` has the type colour and represents red. Colours have three properties, `red`, `green` and `blue`, which can be used to access and alter that colour. Here is a simple example using the colour type:

Listing 8

```
function pixel(frame,x,y)
    local c = colour.new(255,0,0)
    return c.red,c.green,c.blue
end
```

This fills every pixel of every frame with red.

Earlier in this document, we stated that the `pixel` function should return 3 numbers, representing the red, green and blue components of a colour. This was not the entire truth. We are also allowed to return a single variable of type colour. This function is therefore equivalent to Listing 8:

Listing 9

```
function pixel(frame,x,y)
    local c = colour.new(255,0,0)
    return c
end
```

Once again, we return to our vertical band example and use colour variables to specify the band colour and the background colour:

Listing 10

```
-- width of the bands in pixels
band_width = 4
-- space between bands in pixels
```

```
band_spacing = 1
-- the colour of the band
band_colour = colour.new(255,0,0)
-- the colour of the space between bands
background_colour = colour.new(0,0,255)

-- get the combined width of band and separator
local total_band_width = band_width+band_spacing
-- get the number of visible bands
local bands = width/total_band_width

-- modulo operator (a%b)
function mod(a,b)
    return a - math.floor(a/b)*b
end

-- the pixel function
function pixel(frame,x,y)

    if (mod(x,total_band_width)>=band_width) then
        -- in band separator
        return background_colour
    end

    -- get the band in which this pixel falls
    local band = math.floor(x/total_band_width)

    -- get the fraction through the effect
    local t = frame/frames

    -- get the height of the band in which this pixel falls
    local band_height = (math.sin((band/bands+t)*math.pi*2)+1)/4

    -- adjust y to be relative to the center of the effect
    y = y-(height/2)+0.5

    -- decide if this pixel is inside the band
    if (math.abs(y)/height<=band_height) then
        return band_colour
    else
        return background_colour
    end
end
```

We have added two variables, `band_colour` (red) and `background_colour` (blue) and are now returning those values rather than the r,g,b values that we were using previously. You should now see red bands rippling over a blue background.

A simple gradient

The colour library also includes an `interpolate` function, which takes two colours and a fraction and returns a new colour that is linearly interpolated between the two colours. For example:

Listing 11

```
local red = colour.new(255,0,0)
local blue = colour.new(0,0,255)

function pixel(frame,x,y)
    -- interpolate between red and blue using the horizontal displacement of
x
    -- note that we use (width-1) so the rightmost pixel is completely blue
    return colour.interpolate(red,blue,x/(width-1))
end
```

This creates a horizontal red to blue gradient. We could have created the same gradient without the colour library as follows:

Listing 12

```
function pixel(frame,x,y)
    local f = x/(width-1)
    return 255*(1-f),0,(255*f)
end
```

However, if you changed your mind about the colours that you wanted for your gradient, it would be significantly harder to alter Listing 12 than it would be to change the colours in the first two lines of Listing 11.

Working with gradients

The gradient library adds support for more complicated gradients that cannot be achieved by interpolating between two colours.

To create a new variable of type gradient, call `gradient.new()`, passing in two colours, i.e:

```
local c1 = colour.new(255,0,0)
local c2 = colour.new(0,0,255)
local g = gradient.new(c1, c2)
```

To find the colour of the gradient at a specific point, use the `lookup` function, passing in a number between 0 and 1. For example:

Listing 13

```
local red = colour.new(255,0,0)
local blue = colour.new(0,0,255)
local g = gradient.new(red, blue)

function pixel(frame,x,y)
    -- note the use of the colon operator
    return g:lookup(x/(width-1))
end
```

This creates a horizontal gradient from red to blue, but we have already seen that there are other ways to generate the same result which will probably be more efficient. To show where the gradient library offers more power:

Listing 14

```
local red = colour.new(255,0,0)
local blue = colour.new(0,0,255)
local g = gradient.new(red,blue)

-- add a third point to the middle of the gradient
local green = colour.new(0,255,0)
g:add_point(0.5,green)

function pixel(frame,x,y)
    return g:lookup(x/(width-1))
end
```

We used the `add_point` function to insert a green colour midway between the red and the blue colours. This generates a horizontal gradient that fades from red to green to blue.

Back to the vertical band example, we will use a gradient to colour the bands:

Listing 15

```
-- width of the bands in pixels
band_width = 4
-- space between bands in pixels
band_spacing = 1
-- the colour of the band
band_gradient = gradient.new(colour.new(255,0,0), colour.new(255,255,0))
-- the colour of the space between bands
background_colour = colour.new(0,0,0)

-- get the combined width of band and separator
local total_band_width = band_width+band_spacing
-- get the number of visible bands
local bands = width/total_band_width

-- modulo operator (a%b)
function mod(a,b)
    return a - math.floor(a/b)*b
end

-- the pixel function
function pixel(frame,x,y)

    if (mod(x,total_band_width)>=band_width) then
        -- in band separator
        return background_colour
    end

    -- get the band in which this pixel falls
    local band = math.floor(x/total_band_width)

    -- get the fraction through the effect
    local t = frame/frames

    -- get the height of the band in which this pixel falls
    local band_height = (math.sin((band/bands+t)*math.pi*2)+1)/2
```

```
    -- adjust y to be relative to the center of the effect
    y = y-(height/2)+0.5

    -- decide if this pixel is inside the band
    local h = math.abs(y)/(height/2)
    if (h<=band_height) then
        return band_gradient:lookup(h)
    else
        return background_colour
    end
end
```

The `band_gradient` variable is initialised as a red to yellow gradient, and we use `band_gradient:lookup(h)` to determine the colour of the band at height h.

Working with properties

Custom presets can have properties which will be exposed in Designer whenever the preset is placed on a timeline. This allows a single custom preset to create a wide variety of effects. It also means that you do not have to create near-identical copies of custom presets just to change one parameter, for example, a colour. You can just expose a colour property and specify the desired colour when the preset is placed on a timeline.

To define a property, you would call the function:

```
property(name, type, default_value, ...)
```

This must be added to your script outside of any function call.

`name` is a string and must be unique within a custom preset and must not contain spaces. This name will be used as the name of a global variable that is available in your script, whose value will depend on what has been set for a given instance of your custom preset.

`type` is the type of the property. It can be one of the following values: `BOOLEAN`, `INTEGER`, `FLOAT`, `COLOUR` and `GRADIENT`. This determines what sort of control is presented to the user when placing a custom preset on a timeline.

`default_value` is the initial value of a property when first added to a timeline. The value passed in here depends on the type of the property, and this is outlined below.

Certain types of properties also allow some addition arguments to be specified, and these will also be described for each type below:

Boolean properties

```
property("invert", BOOLEAN, true)
```

The default value should be `true` or `false`.

Integer properties

```
property("count", INTEGER, number, [min], [max], [step])
```

The default value should be a number between `min` and `max`.

- `min` is the minimum allowed value (default: -2147483648)
- `max` is the maximum allowed value (default: 2147483647)
- `step` is the difference between allowed values (default: 1)

`min`, `max` and `step` are optional.

Float properties

```
property("count", FLOAT, number, [min], [max], [resolution])
```

The default value should be a number between `min` and `max`.

- `min` is the minimum allowed value
- `max` is the maximum allowed value
- `resolution` is the number of decimal places to display (default: 2)

`min`, `max` and `resolution` are optional.

Colour properties

```
property("background", COLOUR, red, green, blue)
```

`red`, `green` and `blue` are the default values of the components of the colour.

Gradient properties

```
property("Gradient", GRADIENT, {fraction, red, green, blue}, ...)
```

The default value of a gradient is a list of fractions and colours, where `fraction` is in the range [0-1] and specifies where in the gradient the colour is, and `red`, `green` and `blue` is the colour at that position and are in the range [0-255]. You can specify multiple points. For example:

```
property("Gradient", GRADIENT, 0.0, 255, 0, 0, 1.0, 0, 0, 255)
```

creates a red (255,0,0) point at the start (0.0) and a blue ((0,0,255) point at the end (1.0).

To demonstrate a real example of using properties in scripts:

Listing 16

```
property("g", GRADIENT, 0.0, 255, 0, 0, 1.0, 0, 0, 255)

function pixel(frame,x,y)
    return g:lookup(x/(width-1))
end
```

This, by default, creates a horizontal gradient from red to blue, as we saw in Listing 13. However, when this preset is placed on a timeline, there will be a gradient editor available, and you will be able to alter the gradient to be any colour you wish, without having to recompile the script or having to duplicate the custom preset with some small alterations.

We will now modify our vertical band example to expose some properties to make a very versatile effect:

Listing 17

```
-- width of the bands in pixels
property("band_width", INTEGER, 4, 1)
-- space between bands in pixels
property("band_spacing", INTEGER, 1, 0)
-- the wavelength of the ripple (in terms of current width)
property("wavelength", FLOAT, 1, 0, 16, 2)
```

```
-- the direction of the ripple
property("reverse", BOOLEAN, false)
-- the colour of the band
property("band_gradient", GRADIENT, 0, 255, 0, 0, 1, 255, 255, 0)
-- the colour of the space between bands
property("background_colour", COLOUR, 0, 0, 0)

-- get the combined width of band and separator
local total_band_width = band_width+band_spacing
-- get the number of visible bands
local bands = width/total_band_width

-- modulo operator (a%b)
function mod(a,b)
    return a - math.floor(a/b)*b
end

-- the pixel function
function pixel(frame,x,y)

    if (mod(x,total_band_width)>=band_width) then
        -- in band separator
        return background_colour
    end

    -- get the band in which this pixel falls
    local band = math.floor(x/total_band_width)

    -- get the fraction through the effect
    local t = frame/frames

    -- optionally reverse the ripple
    if (reverse) then t = -t end

    -- get the height of the band in which this pixel falls
    local band_height = (math.sin((band/bands/wavelength+t)*math.pi*2)+1)/2

    -- adjust y to be relative to the center of the effect
    y = y-(height/2)+0.5

    -- decide if this pixel is inside the band
    local h = math.abs(y)/(height/2)
    if (h<=band_height) then
        return band_gradient:lookup(h)
    else
        return background_colour
    end
end
```

You will notice that adding properties to the example involved little more than changing the variable definitions at the start of the script. There are also two new properties, `wavelength`, for setting the wavelength of the ripple, and `reverse`, for changing the direction of the ripple.

By adjusting the values of the properties, we can now create a variety of different effects without having to alter the script again.

Colour library summary

`colour.new(r,g,b)`

Returns a new colour that represents the RGB color specified by the components `r`, `g` and `b`. `r`, `g` and `b` will be limited to the range [0,255].

`colour.interpolate(c1,c2,f)`

Returns the colour that is linearly interpolated between colour `c1` and colour `c2` at fraction `f`. `f` can fall outside of the range [0,1] and the returned colour will be extrapolated accordingly.

Properties

`c.red`

The value of the red component [0-255] of colour `c`.

`c.green`

The value of the green component [0-255] of colour `c`.

`c.blue`

The value of the blue component [0-255] of colour `c`.

Gradient library summary

`gradient.new(c1,c2)`

Returns a new gradient with colour `c1` at the start and colour `c2` at the end.

Functions

`g:lookup(f)`

Returns the colour at fraction `f` through the gradient `g`. `f` will be limited to the range [0,1].

`g:add_point(f, c)`

Adds the colour `c` to the gradient `g` at fraction `f`.

Built-in functions

`dist(x1,y1,x2,y2)`

Returns the distance between coordinate (`x1`,`y1`) and coordinate (`x2`,`y2`)

`dist_from_center(x,y)`

Returns the distance between coordinate (`x`,`y`) and the center of the frame. This is not the same as calling `dist(x,y,width/2,height/2)`. It takes into account the fact that the center of the frame may fall in the middle of a pixel. For example, if `width` and `height` were equal to 5, the center of the frame is the center of the pixel at coordinate (2,2), but calling `dist(2,2,width/2,height/2)` will return 0.707, which is the distance between the top left of pixel (2,2) and its center. Calling `dist_from_center(2,2)`, where `width` and `height` are equal to 5, will return 0.

`print(message)`

Prints `message` in the debugger's Output window.

You are advised to remove calls to this function when you have finished debugging because it will allow the script to run faster when used in programming.

`rgb_to_hsi(red,green,blue)`

Converts an RGB (red, green, blue) colour to an HSI (hue, saturation, intensity) colour. `red`, `green` and `blue` are in the range [0-255]. Returns three numbers, hue is in [0-2PI] radian, saturation and intensity are in the range [0-1].

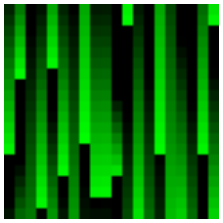
```
hsi_to_rgb(hue,saturation,intensity)
```

Converts an HSI (hue, saturation, intensity) colour into an RGB (red, green, blue) colour. `hue` is in [0-2PI] radians, `saturation` and `intensity` are in the range [0-1]. Returns three numbers in the range [0-255].

Custom Preset Scripting Examples

NOTE: The VLC and VLC+ do not support Custom Presets and thus these will not be available for projects using these controllers.

Matrix



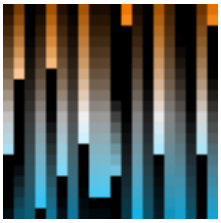
The green randomly scrolling bars from the Matrix film.

Show Code

```
-- seed for the random number generator
property("seed", INTEGER, 0, 0)
-- initialise the random number generator
math.randomseed(seed)
-- generate a random period,active,offset value for each column of the effect
table = {}
local i = 0
while (i < width) do
table[i] = { math.random(1, 3), math.random(4, 10)/10, math.random() }
i = i+1
end
-- ramp-down effect curve lookup
function ramp_down(f,period,active,offset)
f = f/period
f = f+offset
f = f-math.floor(f)
if (f>active) then return 0 else return f/active end
end
-- the pixel function
function pixel(frame, x, y)
local t = frame/frames
local period = table[x][1]
local active = table[x][2]
local offset = table[x][3]
local f = ramp_down(y/height,period,active,1-t+offset)
```

```
return 0,255*f,0
end
```

Colour Rain



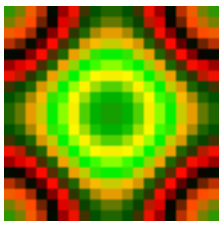
Similar to Matrix, but the random bars fade between two colours as they drop, also the direction can be set.

Show Code

```
-- seed for the random number generator
property("seed", INTEGER, 0, 0)
property("gradient", GRADIENT, 0.0, 255, 128, 0, 0.5, 255, 255, 255, 1.0, 0, 192, 255)
property("reverse", BOOLEAN, false)
property("rotate", BOOLEAN, false)
-- initialise the random number generator
math.randomseed(seed)
-- generate a random period,active,offset value for each column of the effect
table = {}
local i = 0
local dimension = width
if (rotate) then
   dimension = height
end
while (i < dimension) do
   table[i] = { math.random(1, 3), math.random(4, 10)/10, math.random() }
   i = i+1
end
-- ramp-down effect curve lookup
function ramp_down(f,period,active,offset)
   f = f/period
   f = f+offset
   f = f-math.floor(f)
   if (f>active) then
      return 0
   else
```

```
      return f/active
    end
  end
  -- the pixel function
  function pixel(frame, x, y)
    local t = frame/frames
    local col = x
    local row = y
    local rowWidth = width
    local colHeight = height
    if (rotate) then
      col = y
      row = x
      rowWidth = height
      colHeight = width
    end
    if (reverse) then
      col = rowWidth - col - 1
      row = colHeight - row - 1
    end
    local period = table[col][1]
    local active = table[col][2]
    local offset = table[col][3]
    local f = ramp_down(row/height,period,active,1-t+offset)
    local c = gradient:lookup(row / colHeight)
    return colour.new(c.red * f, c.green * f, c.blue * f)
  end
```

Plasma



Generates a classic plasma effect.

Show Code

```
  -- return a multi-frequency noise value
  function plasma(x, y, x_period, y_period)
```

```
    local cx = width/2-0.5
    local cy = height/2-0.5
    return (math.cos(((cx-x)/x_period)*math.pi*2)+
    math.cos(((cy-y)/y_period)*math.pi*2))/2
end
-- return a colour for a given noise value
function colour(f)
    return (math.sin(f*math.pi*4)+1)/2*255,(math.sin(f*math.pi)+1)/2*255,0
end
-- the pixel function
function pixel(frame, x, y)
    -- calculate a noise value
    local f = plasma(x,y,width,height)
    -- offset the noise to animate the effect
    local offset = frame/frames*2
    -- lookup the colour for the noise value
    return colour(f+offset)
end
```

More Examples

More examples are available on our website.

Trigger Script Programming Guide

Introduction

The Pharos Controllers offer many useful show control capabilities. Frequently it is the ability to cope with the particular show control needs of a project that is the critical factor in selecting a control system.

Show control broadly consists of two tasks. First we need to be able to interface with other devices, which may either be triggering us or be under our control. The Pharos Controller supports most of the core interfaces typically used for show control, either directly on the unit (contact closures, RS232, MIDI, TCP/UDP, time and date) or Remote Devices. Within the Triggers screen of the Designer software we can configure the Controller to detect particular triggers and how to respond to them.

Second we need to be able to make decisions. These could be simple choices between two alternatives - perhaps a contact closure needs to trigger a different timeline depending on whether it is during the day or during the night. Within the Triggers screen we support a range of conditions that can be used to quickly implement this sort of logical decision making. We also provide a facility to treat values received on an input as a variable that can be used to alter the behaviour of actions - such as using a number received via RS232 to select a particular timeline.

The standard capabilities offered in the Triggers screen are extensive, but a good show control system has the ability to cope with situations that are anything but standard. Within the Pharos system when things get non-standard then we can use scripting.

Lua is a simple programming language that allows users to extend the functionality of the Pharos system themselves. We use a freely available programming language called Lua. Anyone who has ever worked with a programming language will find all the typical tools are available, and it should be straightforward to pick up for those who have not. On top of the core Lua syntax we have added some dedicated Pharos functions that allow scripts to work directly with the capabilities of a Controller.

Not every problem requires script, but there are few show control problems that can't be solved using script where necessary. A few examples of situations where you might want to use script include:

- Making a single contact closure start a different timeline each time
- Make a timeline loop a set number of times and then release
- Track motion sensor activity over a period of time
- Inverting a DMX input before it is used with a Set Intensity action
- Interpreting data from a wind direction sensor
- Using a table of times for high and low tide to control bridge lighting
- Implementing an interactive game for a science museum

We will use some of the situations as examples below.

The Basics

There are a few basic things you need to know straight away. If any of them are not immediately clear then don't worry - there are lots of examples of how to apply them in the following section.

Lua scripts are written as simple text files using any text editor. It is standard practice to use a .lua filename extension though this is not required. These text files can be loaded directly into the Script Editor within Designer.

Comments

It is good practice to include readable comments in your scripts so that you (or anyone else) will be able to easily tell what you were aiming to achieve. In Lua everything after two dashes on a line is treated as a comment.

```
-- This is a comment
This = is + not - a * comment -- but this is!
```

The whole point of comments is that they have no effect on the behaviour of the script. But I am introducing them first so that I can use them within the examples that follow.

Variables

If you want to store a piece of data - whether it is a number (referred to as an integer, float or real), some text (referred to as a string) or just true or false (refer to as a boolean)- then you use a variable. You create a variable simply by giving it a name and using it in your script. A variable can store any type of data just by assigning it.

```
firstVariable = 10 -- assign a number
anotherVariable = "Some text" -- assign a string
```

When you next use these names then they will have the values that you assigned to them:

```
nextVariable = firstVariable + 5 -- value of nextVariable will be 15
```

Note that names are case-sensitive (i.e. capitals matter!), and once you have named a variable once then any time you use the same name you will be referring to the same variable - in programming terms it is *global*. This even applies across different scripts - so you can assign a number to a variable called `bob` in one script and then use the number in another script by referencing `bob`.

One of the most common errors when writing scripts is trying to use a named variable before it has been assigned a value - this will result in an error when the script is run. It is also very easy to use the same name in two different places and not realise that you are actually reusing a single variable. (There is a way of dealing with this for names you want to reuse that we will touch on later.)

Arithmetic

Scripts will often need to do some arithmetic - even if it is something very basic like keeping a counter of how many times it is run:

```
myCount = myCount + 1
```

All of the standard arithmetic operations are available. There is also a library of mathematical functions available should it be required, which includes things like random number generators.

Flow of Control

In most scripts there will be one or more points where you want to make choices. Lua provides four useful structures for this. The most common is `if`, where you can choose which path to take through the script by performing tests.

```
if myNumber < 5 then -- tests whether myNumber is less than 5
    -- first choice
elseif myNumber < 15 and myNumber > 10 then
    -- second choice
else
    -- third choice
end
```

The other control structures all involve blocks of script that need to be repeated a certain number of times. The most straightforward is the `while` loop, which will repeat the enclosed block of script as long as the test at the start is true:

```
myNumber = 10
while myNumber > 0 do
    -- some useful script
    myNumber = myNumber - 1 -- myNumber counts down
end
```

The `repeat until` loop is really exactly the same, but here the test is done at the end of each loop and it will repeat while the test is false.

```
myNumber = 1
maxNumber = 4096
repeat
    -- some useful script
    myNumber = myNumber * 2
until myNumber == maxNumber
```

Here it is worth noting the use of two equal signs == to mean 'is equal to' in a test. This is different from a single equal sign, which is used for assigning values. It is another very common mistake to assign a value when you meant to test if it was equal, and it can be hard to spot because it is valid syntax that will not generate an error. The opposite of == meaning 'is equal to' is ~= meaning 'is not equal to'.

The other control structure is the `for` loop, which has a number of powerful options beyond the scope of what we need here. But it is worth seeing how it can be used to do basic loops in a slightly neater way:

```
for i = 1,10 do
    -- some useful script where i has value 1 to 10
    -- i increments at the end of each loop
end
```

A final word of caution regarding loops: be careful that you do not write a loop that will never exit! This is all too easy to do by forgetting to increment a counter value that you are using in the test for the loop. If your script has one of these 'infinite loops' then the Controller will get stuck when it runs the script and be reset by the watchdog feature (provided this is enabled). Make sure you test your scripts carefully before leaving them to run.

Tables

Often you will need to store a set of values within a script - these might be a list of timeline numbers or the current states of all the contact closure inputs. Lua allows us to store multiple values within a single named variable and this is called a Table.

A table has to be created before it can be used:

```
firstTable = {} -- creates an empty table
secondTable = { 5,3,9,7 } -- a table with 4 entries
```

You can then access entries within the table by indexing into it - signified by square brackets. The number within the square brackets identified which entry within the table you want to use or modify.

```
x = secondTable[3] -- x now equals 9 (3rd entry)
firstTable[1] = 5 -- entry 1 now has value 5
firstTable[7] = 3 -- entry 7 now has value 3
x = firstTable[1] + firstTable[7] -- x now equals 5 + 3
```

Note that we are allowed to assign values to entries within the table without doing anything special to change the size of the table. We can keep adding elements to the table as needed and Lua will take care of it for us. This makes it possible to write scripts using tables that will work regardless of how many entries there are in the table (e.g. a list of 4 timeline numbers or of 40).

Tables are particularly powerful when used together with the loops we looked at in the previous section. For example if I have a table of numbers and I wanted to find the smallest then I could use the following script:

```
numbers = { 71,93,22,45,16,33,84 }

smallest = numbers[1] -- initialise with the first value
i = 1 -- use to count loops
while numbers[i] do -- loop while numbers[i] exists
    if numbers[i] < smallest then
        smallest = numbers[i]
    end
    i = i+1
end
```

This is our first really functional piece of script and there are a couple of things worth noting.

- The first entry in a table is accessed using the number one (i.e. `myTable[1]`). This may seem obvious - but some other programming languages start counting from zero.
- As we increment the variable `i` each time around the loop this means we will be looking at a different entry in the table each time around. The test at the start of my while loop is written to work regardless of how many entries there are in the table. When you use a table entry in a test like this then it will be true as long as the entry has some value (even if the value is zero) and false if there is no value there at all.

Functions

Within script there are a whole range of pre-defined operations that you can call when writing your own scripts. Some of these are provided by the Lua language and are fully described in its documentation. Others have been provided by Pharos to allow you to interact with the Controller from script and are fully described in the manual. They are all called functions and accessed using a similar syntax. For example:

```
x = math.random(1,100)
```

This will assign variable `x` a value that is a random number between 1 and 100. The function `math.random()` is a standard function provided by Lua and we can control its behaviour by passing in an argument - in this case the values 1 and 100 to tell it the range within which we want our random number to fall.

```
t = 5
get_timeline(t):start()
```

`get_timeline(num):start()` is one of the functions provided by Pharos and it will start the timeline with the number passed in as an argument.

It is also possible to define your own functions as part of script. You might do this if there is a block of script that you know you will need to reuse in a lot of different places. It will be much easier to write the script in one place and then call it from wherever you need it.

```
function diff(a, b)
    if a > b then
        return a - b
    else
        return b - a
    end
end

v1 = 10
v2 = 6
v3 = diff(v1,v2) -- v3 == 4
```

Note that the script containing the function definition must have been run before we try to call the function. It is often useful to have a script that is run by the Controller startup trigger which defines your functions and creates any tables - other scripts that are run by triggers can make use of those functions and tables.

More information

In this document we have only covered the basic concepts that are needed to understand or write useful scripts for the Controllers. For more extensive information on the Lua language there are two documents, both of which are available online at http://www.lua.org or can be bought as books from Amazon.

- Lua 5.3 Reference Manual
- Programming in Lua

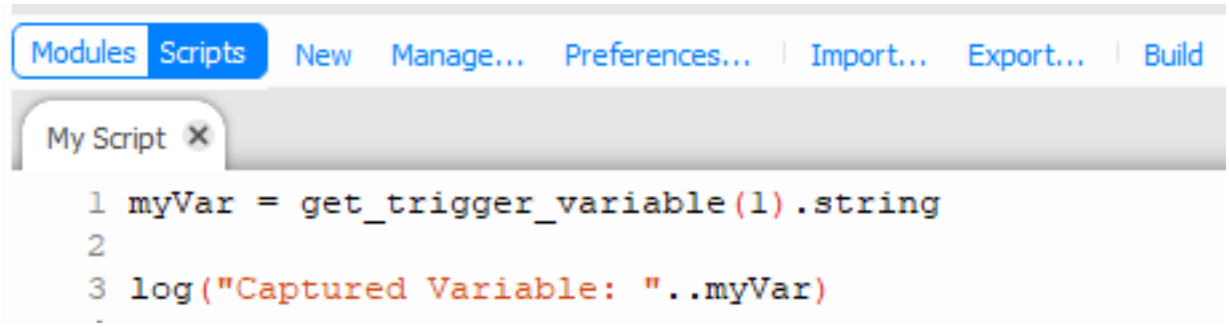
Lua API (Triggering)

We use a scripting language called Lua, which has been extended to provide functionality specific to the Pharos Controllers. Tutorials and reference manuals for the Lua language can be found at www.lua.org. We will not attempt to document the Lua language here, but just the Pharos specific extensions. Please contact support if you need assistance with preparing a script or if you would like some examples as a starting point.

Supports Lua v5.3.4.

Lua script editor

The Lua Script Editor allows you to edit scripts from Triggers, Conditions and Actions within Designer. The Script Editor is launched by pressing the Scripts & Modules button on the Trigger Toolbar, and selecting Scripts in the bottom pane:



The main area of the editor is the code editor where you enter the source code of the script. The code editor will colour the Lua syntax to aid readability. Standard clipboard shortcuts and undo/redo are supported.

To create a new script for use in Conditions or Actions click New Script.

Scripts can be opened using the Open option and closed with the on the Script Tab.

To import a Lua script from an external file, use Import.

To save a Lua script to a file, use Export.

To compile the script and check for syntax errors, use Build. If there are errors in the script, they will be displayed at the bottom of the window.

Changes to scripts are saved automatically.

Find

Pressing Ctrl(Cmd) + F will open the find bar in the script editor.

This allows you to search for text within your script.

Aa	If selected, the case must match
|Abc|	If selected, the whole word must match
.*	If selected, Regular Expressions can be used in the search box

Pharos Trigger Scripts

Syntax

Where a function returns an Object (e.g. get_timeline(num)) additional functions and variables become available. To access a function, add a colon (:) between the functions:

```
get_timeline(1):start() -  This will get the timeline object for timeline 1
and apply the start function to it (starting timeline 1)
```

To access a variable, add a period (.) between the function and the variable:

```
get_timeline(1).is_running - This will return a boolean value indicating
whether timeline 1 is running
```

Pharos Lua API

This is documented along with the rest of the API here.

Scripting Examples

In this section we will go through a number of practical examples of how scripts can be used with a Controller. These examples are all based on real projects that are installed and working. They do get progressively more involved, so do not worry if you don't follow the later ones - you will still be able to use script successfully to solve many problems.

Conditions

Running a Trigger 50% of the time

The script below can be used to only run the trigger 50% of the time randomly

```
-- returns true randomly, 50% of the time

return math.random(1,2) == 1
```

Actions

Cycling through different timelines

We are installing a wall of RGB LED fixtures in a children's play area. There is a single large button that the kids are supposed to press. Each time they press it they should get a different colour or effect on the wall.

Each colour or effect would be programmed as a different timeline in Designer. The button will connect to a contact closure and so we will have a single Digital Input trigger. Rather than starting a timeline directly we will instead run the following script:

```
-- which timelines should we cycle through?
timeline = { 22, 14, 24, 16, 15, 17, 21 }

-- on first time of running, initialise index
if not index then
    index = 1
end

-- start the timeline whose number is at entry 'index'

get_timeline(timeline[index]):start()

-- increment index
index = index + 1

-- should we go back to the beginning of the table?
if index > #timeline then -- #timeline returns the number of values in the
table
    index = 1
end
```

```
-- which timelines should we cycle through?
timeline = { 22, 14, 24, 16, 15, 17, 21 }
-- on first time of running, initialise index
if not index then
    index = 1
```

```
end
-- start the timeline whose number is at entry 'index'
get_timeline(timeline[index]):start()

-- increment index
index = index + 1
-- should we go back to the beginning of the table?
if index > #timeline then -- #timeline returns the number of values in the table
    index = 1
end
```

How would this change if we wanted each button press to choose a timeline at random rather than cycling through them in order?

```
-- which timelines should we cycle through?
timeline = { 22, 14, 24, 16, 15, 17, 21 }

-- use the random function to set index
index = math.random(1,#timeline)

-- start the timeline whose number is at entry 'index'
get_timeline(timeline[index]):start()
```

Of course if the timeline selection is truly random then it will sometimes select the same timeline twice in a row. If we wanted to prevent this from happening how could we do it?

```
-- which timelines should we cycle through?
timeline = { 22, 14, 24, 16, 15, 17, 21 }

-- find an index different from the old one
while index == oldIndex do
    -- use the random function to set index
    index = math.random(1,#timeline)
end

-- store the index for next time round
oldIndex = index

-- start the timeline whose number is at entry 'index'
get_timeline(timeline[index]):start()
```

Stopping a Range of Timelines

We need to stop a large number of timelines in one go, but not all of them.

You can use up to 32 Actions on a Trigger, so if you need to stop more than 32 Timelines at once, you will need to use a script.

You can stop a single timeline from a script with the following:

```
get_timeline(1):stop()
```

This allows you to stop a single timeline from a script, but if you have a large number to stop, adding this for each timeline is a lot of work.

A FOR loop can be used to reduce the amount of scripting required.

```
for i=1,10 do -- run through the values 1-10
    get_timeline(i):stop() -- Stop the timeline defined by i
end
```

This script can be used to run through from 1 to 10 (or a different range by changing the values), and will stop the timeline with those numbers. To make this more useful, you can put it in a function which allows you to call it with any range of timeline numbers.

```
function stop_range(a, b) -- this defines the script as a function with two
variables (a and b)
    for i=a,b do -- a FOR loop which runs through from a to b
        get_timeline(i):stop() -- stop the timeline defined by i
    end
end

stop_range(1,10) -- call the function with the variables 1 and 10
```

NOTE: It is generally best practice to define the function in a script that is run at startup, and then call the function when it is needed

Make a timeline loop N times

The designer has requested that a particular timeline runs once at sunset on a Monday, but twice at sunset on a Tuesday, three times at sunset on Wednesday, etc. He is planning to keep changing the timeline so does not want to have lots of copies.

There are actually lots of perfectly reasonable ways to solve this using script. Let's assume we have a single astronomical clock trigger that fires at sunset and runs the following script:

```
N = time.get_current_time().weekday -- 1 is Monday, 7 is Sunday

get_timeline(1):start()
```

The timeline would be set to loop when it was programmed. We also put a flag on the timeline at the end and make a flag trigger that runs a second script:

```
-- decrement N
N = N - 1

if N == 0 then
    -- release timeline 1 in time 5s
    get_timeline(1):stop(5)
end
```

Note how this works by setting the value of the variable `N` in one script and then using that variable in another script, which is often a useful technique.

We have used two scripts here, but it is possible to do the same job using only one.

In this case you would have the sunset trigger start the timeline directly and use the following script on the flag trigger:

```
-- is this the first time round?
if not N or N == 0 then
    N = time.get_current_time().weekday -- 1 is Monday, 7 is Sunday
end

-- decrement N
N = N - 1
```

```
if N == 0 then
    enqueue_trigger(2) -- runs action on trigger 2
end
```

The trick here is to detect whether it is the first time round the loop - if the Controller has started up today then `N` will have no value and so `not N` will be `true`, otherwise `N` will have been left with the value zero when the script ran yesterday. When we detect it is the first time then we set its initial value in the same way as before.

We have also used a different method to do the timeline release. Rather than calling `get_timeline(num):stop()` directly from the script we are causing Trigger number 2 to fire. We can then configure Trigger number 2 to have an Action that releases the correct timeline. It is sometimes easier to write scripts like this, as it can be easier to see where to change properties. In this case all the you need to do is to modify the Start and Release Timeline Actions in the Trigger list if you want to change which timeline is run.

Storing Data to the Memory Card

In the event that a controller reboots, we want it to start running the timeline that was running prior to the reboot.

The Lua library contains functions that make it possible to read and write files to the device the Lua is running on. This includes reading and writing files on the Controller's Memory Card.

```
running = 1
```

This variable will be used to store the number of the timeline that was started most recently, using a Timeline Started Trigger (set to Any) with a Run Script action as below:

```
running = get_trigger_variable(1)
```

Storing data on the memory card involves two steps, writing to the card and reading back from the card.

```
function writeToCard() -- Write the running timelines table to the memory
card
       file = io.open(get_resource_path("timeline.txt"), "w+") -- Open or create a
file (in write mode) called timeline.txt
       if (file ~= nil) then -- Ensure the file has been opened
               io.output(file) -- Set the open file as the default output location for the
io library
               io.write("running = " .. running) -- Write the line "running = [value]" to
the file. The value will be the value of the variable
               io.flush() -- Clear the output buffer
               io.close() -- Close the file
       end
end
```

Whenever the function writeToCard is called it will store the current value of the variable running to the memory card in a file called timeline.txt. The file will be in the format:

running = [value]

This is the syntax for defining a variable in Lua and means that if we run the file on startup, it will set the variable running to be the value stored in the file (with no parsing required)

```
function readFromCard() -- Read the stored running timelines table and start
the timelines specified
       file = io.open(get_resource_path("timeline.txt"),"r") -- Open the file
timeline.txt (in read mode) if it exists
       if file ~= nil then -- Ensure the file has been opened
```

```
                dofile(get_resource_path("timeline.txt")) -- Run the file to set the
variable
       end
       get_timeline(running):start() -- Start the timeline stored in the running
variable
end
```

Whenever the function readFromCard is run, it will find the file called timeline.txt, run it so that the stored variable is set on the controller and then start the relevant timeline

NOTE: Functions should be placed in a Run Script action at Startup to ensure they are declared. They can then be used at any time within the show file.

Push data to the web interface

If you are using a Custom Web Interface, it is possible to push data to it from the project file e.g. when a TPC Slider is moved. You will then need to set up some JavaScript within your custom web interface to read the data in.

If we want to send the level of a TPC slider to the web interface, we would use a Touch Slider Move Trigger set to match the Slider's Key. This would have a Run Script Action attached with the following Lua Script

```
level = get_trigger_variable(1) -- Capture the level set on the slider

push_to_web("slider_level",level) -- creates a JSON packet in the form
{slider_level:level}, where level is the value stored previously
```

Then within the web interface, we need to use the *subscribe_lua()* function to process this data.

Implementing an interactive game for a Science Museum

In an exhibit children are posed questions and have to select answers from an array of numbered buttons. The buttons are large with RGB backlights that are controlled by a Controller to highlight choices and indicate right and wrong answers. Questions are displayed by a slide projector which is under RS232 control from the Controller. The buttons are wired to contact closures on the Controller and on RIOs, so that the Controller can check answers and determine the progress of the game accordingly. The lighting in the rest of the room is designed to mimic a popular TV quiz show to retain the children's interest, with different timelines for each stage of the game.

I am not going to work through this example - but the key point is that it should now be clear to you that a Controller could be used to implement this sort of advanced interactive exhibit with the use of script. Try breaking down the problem into discrete parts and you will find that no individual part of this is difficult - although getting it all to function together reliably would no doubt require a lot of work. The Controller is a viable alternative to custom software running on a PC and has clear advantages in terms of durability and cost.

Examples

Some Trigger Scripting examples are available on our website.

Variants

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.

Pharos 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.

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
```

.integer can be used to either Get or Set the value of var 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 the variant should be set if you intend to use the variant to set an intensity or colour value.

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

Some captured variables have a range attribute, and this is indicated in the log, as shown above.

Fraction

A fraction can be called to return the value of the integer value over the range.

```
var = Variant() -- create variant

var.integer = 127 -- set value

var.range = 255 -- set range

log(var.fraction) -- return fraction
```

This will then return a float (real) value, which can easily be used to denote the percentage of the integer against the range on a 0.0-1.0 scale.

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
```

.real can be used to either Get or Set the value of var 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
```

.string can be used to either Get or Set the value of var 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)
```

.ip_address can be used to either Get or Set the value of var 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

Variants 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

Variant Definition

In general, the Variant object contains the following variables and functions:

`Variant()`	Create new variant
`.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
`: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

Default Variants

Some script functions return a Variant:

```
get_trigger_variable()

        e.g. get_trigger_variable(1).integer

get_group(1).master_intensity_level

        e.g. get_group(1).master_intensity_level.integer

             get_group(1).master_intensity_level.range

get_content_target(1).master_intensity_level
```

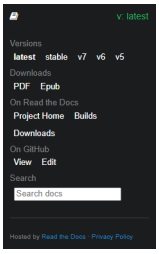
```
e.g. get_content_target(1).master_intensity_level.integer
     get_content_target(1).master_intensity_level.range
```

API

API Help has been moved to a new, online resource. This allows us to manage it better, allowing for a faster turn-around of tweaks and improvements to the API examples, as well as fixing minor errors.

The latest version can be found here.

To view older versions, select an option from the pop up located at the bottom of the navigation panel.



API Authentication

NOTE: API v6 onwards only. Please contact Pharos Support if API v5 or earlier version is required.

If there is a controller password or Web Interface Access Users setup in the project, then Authorisation will be required to use the HTTP or JavaScript API.

There are two types of Authorisation available, which can be used in different situations:

1. Cookie Authentication
2. Token Authentication

NOTE: Both these authentication methods will expire after 5 mins of inactivity

Cookie Authentication

Cookie Authentication is typically used by the web interface (either Default or Custom), and stores a small file on your computer, which lets the controller know that you are currently signed in.

This authentication is provided through the Default Login page, when using the Default Web Interface, or a Custom Web Interface, without a Custom login page defined.

Custom Login Page

If a Custom Web Interface is being used, then a Custom Login page can be configured. Typically this will be a HTML based page with a form element containing a username and password entry field. The code below can be used to generate these fields, and send the data to the controller's web server to authenticate the user and store the Cookie:

```
<form action="/authenticate" method="POST">
  <input type="text" name="username">
  <input type="password" name="password">
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Token Authentication

Token Authentication is typically used by the HTTP API, where there isn't necessarily a method to enter the username and password manually.

Token Authentication works by the user requesting a Bearer Token from the controller, with the username and password, and this token is then used in future requests.

To request a Bearer Token:

1. Send an HTTP request to http://[ip address]/token in the following format:

```
Method: POST
Headers:
  Content-Type: application/json
Body:
```

```
{"username":[username], "password":[password]}
```

2. If successful you will receive a response containing the following JSON Object:

```
{
   "token": "[some_access_token]",
}
```

3. Subsequent requests will require the following header

```
Authorization: Bearer [some_access_token]
```

Legacy API

The Legacy API documentation is available here.

These APIs can be used if the Controller API Setting is set to Legacy.

API v5

The Pharos system includes multiple API options:

- Lua (used internally with Conditions and Actions)
- HTTP (used with external devices/software to communicate with a controller)
- JavaScript (used with Custom Web Interfaces)

These APIs have been unified to simplify their use as much as possible.

Note: Firing JavaScript queries from the controller "Default Web Interface" on page 299 will now propagate to all controllers.

Glossary:

- Object - A collection of key value pairs e.g. "name" = "Controller 1" (syntax will differ between languages).
- String - A series of characters e.g."Th1s_is-4(string)"
- Number - Any whole or floating point(decimal) number e.g. 1,2,3,1.5,12.3456)
- Integer - A whole number
- Bounded integer - An integer with a range (e.g. 10:100 = 10%)
- Float/real/number - A decimal number (e.g. 3.2 or 1.0)
- JSON - (JavaScript Object Notation) a way of transferring information in the form of a JavaScript Object
- GET - A HTTP method to request data from a server
- POST - A HTTP method to request data in a more secure way
- PUT - A HTTP method to send data to a server
- Variant - See here
- [] - anything shown within square brackets is optional. The square brackets should be omitted if the optional section is used.
- callback - A function to run when the javascript function has been run, or a reply has been received.

HTTP Requests

Please note, when an HTTP POST request is sent, it must include a Content-Type header set to "application/json", otherwise it will be treated as invalid.

API Propagation

Custom web interfaces offer end users a useful device-based override control solution.

Prior to 2.8.0, triggering web API (JavaScript/HTTP) commands via a custom web interface would only affect the controller on which it was fired. For projects with multiple controllers, which all needed to respond to a web UI command, this necessitated many additional triggers to send commands system-wide.

From API v5 onwards, the following will propagate to all controllers in the project when fired from any controller via JavaScript or HTTP:

- Start Timeline
- Start Scene
- Release Timeline
- Release Scene
- Toggle Timeline
- Toggle Scene
- Pause Timeline
- Resume Timeline
- Pause All
- Resume All
- Release All
- Set Timeline Rate
- Set Timeline Position
- Master Intensity
- Set RGB
- Clear RGB
- Disable Output
- Set Text Slot
- Set Timeline Source Bus
- Enable Timecode Bus

Please note, the propagating functions will always propagate when called through JavaScript or HTTP. The same functions are available in Lua which will not if called via Run Script in the web API, so in cases where non-propagating functionality is needed, please revert to Lua that can be added to your controller within Designer.

The following commands will not propagate, so will fire only on the controller on which the command was run:

- Enqueue Trigger
- Run Script
- Hardware Reset
- Set BPS Button LED
- Set TPC Control Value
- Set TPC Control State
- Set TPC Control Caption
- Set TPC Page
- Disable TPC
- Lock TPC
- Transition Content Targetl
- Transition Adjustment
- Beacon controller
- Output To Log
- Send Variable To Web Interface
- Park a Channel
- Unpark a Channel
- Edit Config

API Queries

Below are the ways of getting data from the controller. show

System

Returns data about the controller. show

Lua

The system namespace has the following properties:

Property	Return type	Return 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	"2.14.0"
.reset_reason	string	"Software Reset"
.last_boot_time	DateTime object	
.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	“192.168.1.3”,”10.10.10.1”

Example:

```
capacity = system.channel_capacity

boot_time = system.last_boot_time.time_string
```

HTTP

```
GET /api/system
```

Returns an object with the following properties:

Property	Return type	Return Example
hardware_type	string	"LPC"
channel_capacity	integer	512
serial_number	string	"006321"
memory_total	string	"12790Kb"
memory_used	string	"24056Kb"
memory_free	string	"103884Kb"
storage_size	string	"1914MB"
bootloader_version	string	"0.9.0"
firmware_version	string	"2.14.0"
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"

JavaScript

```
get_system_info(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_system_info( function(system){
   var capacity = system.channel_capacity
})
```

Project

Returns data about the project. Show

Lua

get_current_project()

Returns an object with the following properties:

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

Example:

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

HTTP

```
GET /api/project
```

Returns an object with the following properties:

Property	Return type	Return Example
name	string	"Help Project"
author	string	"Pharos"
filename	string	"help_project_v1.pd2"
unique_id	string	"{6b48627a-1d5e-4b2f-81e2-481e092a6a79}"
upload_date	string	"2017-01-30T15:19:08"

JavaScript

```
get_project_info(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_project_info( function(project){
 var author = project.author
})
```

Replication

Returns data about the install replication. Show

Lua

get_current_replication()

Returns an object with the following properties:

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

Example:

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

HTTP

GET /api/replication

Returns an object with the following properties:

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

JavaScript

get_replication(callback)

Returns an object with the same properties as in the HTTP call

Location

Returns data about the install location. Show

Lua

get_location()

Returns an object with the following properties:

Property	Return type	Return Example
lat	float	51.512
long	float	-0.303

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Network 2

Returns data about the Network 2 (Protocol) Interface. Show

Lua

The protocol_interface namespace has the following properties:

Property	Return type	Return 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"

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Time

Returns data about the time stored in the controller. Show

Lua

The time namespace has the following functions which 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()

and the following properties

Property	Return Type	Return Example
is_dst	boolean	
gmt_offset	string	

Each function returns a DateTime object, with the following properties:

Property	Return Type	Return Example
.year	integer	2017
.month	integer (1-12)	5
.monthday	integer (1-31)	8
.weekday	integer(1-7)	1
.hour	integer(0-23)	13
.minute	integer (0-59)	21
.second	integer (0-59)	46
.utc_timestamp	integer	1494249706
.time_string	string	
.date_string	string	

Example:

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

HTTP

```
GET /api/time
```

Returns an object with the following properties:

Property	Return Type	Return Example
datetime	string	"01 Feb 2017 13:44:42"
local_time	integer (controller's local time in milliseconds)	1485956682
uptime	integer (time since last boot)	493347

JavaScript

```
get_current_time(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_current_time( function(time){
 var uptime = time.uptime
})
```

Timeline

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

Lua

```
get_timeline(timelineNum)
```

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

The returned object has the following properties

Property	Return Type	Return Example
name	string	"Timeline 1"
group	string ('A', 'B', 'C', 'D','')	'A'
length	integer	10000
source_bus	integer (equivalent to constants: DEFAULT, TCODE_1 ... TCODE_6, AUDIO_1 ... AUDIO_4)	1
timecode_format	string	"SMPTE30"
audio_band	integer (0 is equivalent to constant VOLUME)	0
audio_channel	integer (equivalent to constants: LEFT, RIGHT or COMBINED)	1
audio_peak	boolean	false
time_offset	integer	5000
state	integer (equivalent to constants: Timeline.NONE, Timeline.RUNNING, Timeline.PAUSED, Timeline.HOLDING_AT_END, Timeline.RELEASED)	1
onstage	boolean	true
position	integer	5000
priority	integer (equivalent to constants: HIGH_PRIORITY, ABOVE_NORMAL_PRIORITY, NORMAL_PRIORITY, BELOW_NORMAL_PRIORITY or LOW_PRIORITY)	0
custom_properties table;	keys and values correspond to custom property names and values, where key = property name and value = property value	

Example:

```
tl = get_timeline(1)

name = tl.name

state = tl.state

if (tl.source_bus == TCODE_1) then

    -- do something

end
```

Example:

```
timelineProperty = get_timeline(1).custom_properties

propertyValue = timelineProperty["Custom Property Name"]
```

HTTP

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

- num can be used to filter which timelines are returned and can be a single number or a string representing the required timelines (e.g. "1,2,5-9")

Returns an object with the following properties:

timelines array of timeline objects

Each timeline object contains the following properties:

Property	Return Type	Return Example
num	integer	1
name	string	"Timeline 1"
group	string('A', 'B', 'C', 'D' or empty)	"A"
length	integer	10000
source_bus	string ('internal', 'timecode_1',...'timecode_6', 'audio_1',...'audio_4')	100
timecode_format	string	"SMPTE30"
audio_band	integer (0 is volume band)	1
audio_channel	string ('left', 'right', 'combined')	"combined"
audio_peak	boolean	false
time_offset	integer	5000
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	"running"
onstage	boolean	true
position	integer	10000
priority	string ('high', 'above_normal', 'normal', 'below_normal', 'low')	"normal"
custom_properties	(object, properties and property values correspond to custom property names and values)	

JavaScript

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

- num can be used to filter which timelines are returned and is defined as a JSON object which can contain a single number or a string representing the required timelines (e.g. "1,2,5-9")

Returns an array of timelines in the same way as the HTTP call

Example:

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

Scene

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

Lua

```
get_scene(sceneNum)
```

Returns a single Scene object for the Scene with user number SceneNum.

The returned object has the following properties

Property	Return Type	Return Example
name	string	"Scene 1"
group	string (A-H) or empty	"A"
state	integer (equivalent to constants: Scene.NONE, Scene.STARTED, Scene.RELEASED)	1
onstage	boolean	false
custom_properties	(object, properties and property values correspond to custom property names and values)	

Example:

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

HTTP

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

num can be used to filter which scenes are returned and can be a single number or array

Returns an object with the following properties:

scenes array of scene objects

Each scene object contains the following properties:

Property	Return Type	Return Example
name	string	"Scene 1"
num	integer	1

state	string ('none', 'started')	"none"
onstage	boolean	false

JavaScript

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

filter may contain a num property which is used to filter which scenes are returned

Returns an array of scenes in the same way as the HTTP call

Example:

```
Query.get_scene_info( function(s){
 var name = s.scenes[0].name //name of the first scene
}, {"num":"1-4"})
```

Group

Returns data about the groups in the project. Show

Lua

```
get_group(groupNum)
```

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

The returned object has the following properties:

Property	Return Type	Return Example
name	string	"Group 1"
master_intensity_level	Variant	

Example:

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

HTTP

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

num can be used to filter which groups are returned and can be a single number or array

Returns an object with the following properties:

groups array of group objects

Each group object contains the following properties:

Property	Return Type	Return Example
num	integer (only included for user created groups)	1
name	string	"Group 1"
level	integer (0-100)	100

JavaScript

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

filter may contain a num property which is used to filter which groups are returned

Returns an array of groups in the same way as the HTTP call

Example:

```
Query.get_group_info( function(g){
 var name = g.groups[0].name //name of the first group
}, {"num":"1-4"})
```

NOTE: Group 0 will return data about the 'All Fixtures' group

Controller

Returns data about the controller. Show

Lua

```
get_current_controller()
```

Returns either the controller, the script, or module it is being run on.

```
get_network_primary()
```

Returns the controller which is currently the network primary.

Both return an object containing the following properties:

Property	Return Type	Return Example
.number	integer	1
.name	string	"Controller 1"
.vlan_tag	string	“192.168.1.1”
.is_network_primary	bool	true

Example:

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

```
is_controller_online(controllerNumber)
```

Returns true if the controller with user number controllerNum has been discovered, and false otherwise

Example:

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

HTTP

```
GET /api/controller
```

Returns an object with the following properties:

controllers array of controller objects (one for each controller in the project)

Each controller object contains the following properties:

Property	Return Type	Return Example
num	number	1
type	string	"LPC"
name	string	"Controller 1"
serial	string	"009060"
ip_address	string (if the controller is discovered)/empty (if the controller is not discovered or is the queried controller)	"192.168.1.3" or ""
online	boolean	true
is_network_primary	boolean	true

JavaScript

```
get_controller_info(callback)
```

Returns an array of controllers in the same way as the HTTP call

Example:

```
Query.get_controller_info( function(controller){
 var name = controller[0].name // name of the first controller
})
```

Temperature

Returns data about the controller's temperature. Show

Lua

```
get_temperature()
```

Returns an object with the following properties

Property	Return Type	Return Example
sys_temp	number (only for LPC X and VLC/VLC+)	40
core_temp	number (only for LPC X and VLC/VLC+)	44
ambient_temp	number (only for TPC, LPC X rev 1)	36.900001525878906
cc_temp	number (only for LPC X rev 2 and VLC/VLC+)	44
gpu_temp	number (only for VLC/VLC+)	44

Example:

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

HTTP

```
GET /api/temperature
```

Returns an object with the following properties:

Property	Return Type	Return Example
sys_temp	number (only for LPC X and VLC/VLC+)	40
core1_temp	number (only for LPC X and VLC/VLC+)	44
core2_temp	number (only for LPC X rev 1	44
ambient_temp	number (only for TPC, LPC X rev 1)	36.900001525878906
cc_temp	number (only for LPC X rev 2 and VLC/VLC+)	44
gpu_temp	number (only for VLC/VLC+)	44

JavaScript

```
get_temperature(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_temperature( function(temp){
 var ambient = temp.ambient_temp // ambient temperature of the controller
})
```

Remote Device

Returns data about the Remote Device/s in the project. Show

Lua

```
get_rio(type, num):get_input(inputNum)
```

- type can be RIO80, RIO44 or RIO08
- num is the remote device number
- inputNum is the number of the input

Returns a boolean if the input is set to Digital or Contact Closure, or an integer if the input is set to Analog.

Example:

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

```
get_rio(type, num):get_output(outputNum)
```

- type can be RIO80, RIO44 or RIO08
- num is the remote device number
- outputNum is the number of the output

Returns a boolean showing the current state of the output.

Example:

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

`get_bps(num):get_state(buttonNum)`

- num is the BPS number
- buttonNum is the number of the button

Returns the state of the button, which can be RELEASED, PRESSED, HELD or REPEAT

Example:

```
bps = get_bps(1)

btn = bps:get_state(1)
```

HTTP

`GET /api/remote_device`

Returns an array of all remote devices in the project.

The returned object has the following structure

remote_devices array of Remote Device objects

Each Remote Device object contains the following properties:

Property	Return Type	Return Example
num	integer	1
type	string ('RIO08', 'RIO44', 'RIO80', 'BPS', 'BPI', 'RIO A', 'RIO D')	"RIO 44"
serial	array (all discovered serial number for the address and type)	["001234"]
outputs	array (of Output objects, only present for RIO44 and RIO08 that are on the queried controller)	[{"output":1,"value":true},{"output":2,"value":true},{"output":3,"value":true},{"output":4,"value":true}]
inputs	array (of Input objects, only present for RIO44 and RIO80 that are 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 (if the remote device is detected as being online)	true

The Output object has the following properties:

Property	Return Type	Return Example
output	integer	1
state	boolean (true means the output is on, false means it is off)	false

The Input object has the following properties:

Property	Return Type	Return Example
input	integer	1

type	string ('Analog', 'Digital', 'Contact Closure')	'Digital'
value	integer or bool (depends on type)	true

JavaScript

```
get_remote_device_info(callback)
```

Returns an array of all remote devices in the project with the same properties as in the HTTP call.

Example:

```
Query.get_remote_device_info( function(remote){
 var type = remote[0].type // type of the first remote device
})
```

Text Slots

Returns data about the text slots in the project. Show

Lua

```
get_text_slot(slotName)
```

- slotName is the name of the text slot

Returns the value of slotName

Example:

```
log(get_text_slot("test_slot"))
```

HTTP

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

slotNames can be used to filter which text slots are returned and can be a single name or array

The returned object has the following structure

text_slots array of Text Slot objects

Each Text Slot object contains the following properties:

Property	Return Type	Return Example
name	string	"text"
value	string	"example"

JavaScript

```
get_text_slot(callback[, filter])
```

filter may contain a names property which is used to filter which text slot values are returned

Returns an array of all text slots in the project with the same properties as in the HTTP call.

Example:

```
Query.get_text_slot( function(text){
 var value = text[0].value // value of the first text slot
```

```
}, {names: "test_slot1", "test_slot2"})
```

Get Log

Returns the log from the queried controller. Show

Lua

Not currently available.

HTTP

```
GET /api/log
```

The returned object has the following structure

Property	Return Type
log	string (containing the whole log of the controller)

JavaScript

Not currently available.

Protocol

Returns the protocols and universes being output from the queried controller. Show

Lua

Not currently available.

HTTP

```
GET /api/protocol
```

Returns all the universes on the queried controller

The returned object has the following structure

outputs	array of Protocol objects

Each Protocol object has the following properties:

Property	Return Type	Return Example
type	integer	1
name	string	"DMX"
disabled	boolean (whether the output has been disabled via an Action)	true
universes	array (Universe objects)	{"key":{"index":1},"name":"1"},{"key":{"index":2},"name":"2"}
dmx_proxy	DMX Proxy Object (where appropriate)	{ "ip_address": "192.168.1.17", "name": "Controller 1" }

Each Universe object has the following properties:

Property	Return Type	Return Example

name	string	"1"
key	Universe Key object	{"index":1}

Each DMX Proxy object has the following properties:

Property	Return Type	Return Example
name	string (name of the controller that is outputting this universe)	'Controller 1'
ip_address	string IP Address of the controller outputting this universe	'192.168.1.23'

The properties of the Universe Key object depends upon the type:

For DMX, Pathport, sACN and Art-Net:

Property	Return Type	Return Example
index	integer	1

For KiNET:

Property	Return Type	Return Example
kinet_port	integer	1
kinet_power_supply_num	integer	1

For RIO DMX:

Property	Return Type	Return Example
remote_device_num	integer	1
remote_device_type	integer (corresponding to values in query.js)	1

For EDN:

Property	Return Type	Return Example
remote_device_num	integer	1
remote_device_type	integer (corresponding to values in query.js)	1
port	integer	1

JavaScript

```
get_protocols(callback)
```

Returns all the universes on the queried controller, with the same structure as the HTTP response.

Output

Returns the levels being output from the queried controller. Show

Lua

```
get_dmx_universe(idx)
```

```
get_artnet_universe(idx)
```

```
get_pathport_universe(idx)
```

```
get_sacn_universe(idx)
```

- idx is the required universe number

```
get_kinet_universe(power_supply_num, port_num)
```

- power_supply_num is the power supply to return the output from
- port_num is the port to return the output from

```
get_edn_universe(remote_device_type, remote_device_num, port_num)
```

- remote_device_type will be enums: EDN10 or EDN 20 (no quotation marks)
- remote_device_num is the number of the EDN
- port_num is the port to return the output from

These all return a Universe object, which has the following function

```
get_channel_value(chnl)
```

- chnl is the channel to get the value from

Example:

```
uni = get_dmx_universe(1) -- get DMX Universe 1

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

HTTP

```
GET /api/output?universe=universeKey
```

- universeKey is a string in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET, protocol:remoteDeviceType:remoteDeviceNum for RIO DMX and protocol:remoteDeviceNum:port for EDN.
- protocol can be dmx, pathport, sacn, art-net, kinet, rio-dmx or edn
- remoteDeviceType can be rio08, rio44 or rio80

Example:

GET /api/output?universe=dmx:1

GET /api/output?universe=rio-dmx:rio44:1

The returned object has the following structure

Property	Return Type	Return Example
channels	array (of channel values)	[0,0,0,0,0,0,0,0,0,255,255,255...255,0,255]
disabled	boolean (whether the output has been disabled via an Action)	true
proxied_tpc_name	string (only if controller is LPC, universe is DMX 2, DMX Proxy has been enabled and the TPC is offline)	'Controller 2'

JavaScript

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

Argument	Type	Example
universekey	string or an object containing protocol and either index, kinet_power_supply_num and kinet_port or remote_device_type and remote_device_num	dmx:1

- universeKey can be either a string, or an object containing protocol and either index, kinet_power_supply_num and kinet_port or remote_device_type and remote_device_num as received from get_protocols

Example:

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

Query.get_output({protocol:DMX, num:1}, function(u){
   console.log(u)
})

Query.get_output("dmx:1", function(u){
   console.log(u)
})
```

Returns an object with the same structure as in the HTTP call

Input

Returns the inputs on the queried controller. Show

Lua

```
get_input(idx)
```

Argument	Type	Example
idx	integer	1

Returns the value of the controllers input as a boolean or integer

Example:

```
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)
```

```
get_dmx_input(chnl)
```

Argument	Type	Example
chnl	integer	1

- chnl is the required channel number

Returns the value of the DMX input at channel chnl as an integer. If no DMX Input is detected, it will return nil.

HTTP

```
GET /api/input
```

The returned object has the following structure

Property	Return Type	Return Example
gpio	array (of Input objects, on LPC or TPC+EXT)	[{"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},{"input":5,"type":"Contact Closure","value":true},{"input":6,"-type":"Contact Closure","value":true},{"input":7,"type":"Contact Closure","value":true},{"input":8,"type":"Contact Closure","value":true}]
dmxIn	object (DMX Input object, if DMX Input is configured)	[0,0,0,0,0,0,0,0,0,255,255,255...255,0,255]

The Input object has the following properties:

Property	Return Type	Return Example
input	integer	1
type	string ('Analog', 'Digital', 'Contact Closure')	"Contact Closure"
value	integer or bool (depends on type)	true

The DMX Input object has the following properties:

Property	Return Type	Return Example
error	string (if DMX Input is configured but no DMX is received)	"No DMX received"
dmxInFrame	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"

JavaScript

Not currently available.

Trigger

Returns the triggers in the project. Show

Lua

Not currently available.

HTTP

```
GET /api/trigger?[type=[triggerType]]
```

- triggerType is a string defining the trigger types to be returned.

The returned object has the following structure

triggers	array (of Trigger objects)

The Trigger object has the following properties:

Property	Return Type	Return Example
type	string	"Startup"
num	integer	1
name	string	"Startup"
description	string	""
trigger_text	string	"At startup"
conditions	array (of Condition objects)	[{"text":"Before 12:00:00 every day"}]
actions	array (of Action objects	[{"text":"Start Timeline 1"}]

The Condition and Action objects have the following properties:

Property	Return Type	Return Example
text	string	"Start Timeline 1"

JavaScript

Not currently available.

Lua Variable

Returns the current value of the specified Lua variable. Show

Lua

Not currently available.

HTTP

```
GET /api/lua?variables=luaVariables
```

Argument	Type	Example
luaVariables	string or comma separated list	'myVar' or 'myVar, myVar2, myVar3'

Returns an object containing all the Lua variables requested and their values.

JavaScript

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

Argument	Type	Example
luaVariables	string or array	'myVar' or 'myVar,myVar2,myVar3'

Returns an object containing all the Lua variables requested and their values.

Example:

```
Query.get_lua_variables("myVar", function(lua){
 var value = lua.myVar
```

```
})
```

Trigger Variable

Returns the value of a variables from the trigger that ran the script. Show

Lua

```
get_trigger_variable(idx)
```

Argument	Type	Example
idx	integer	1

Returns the trigger variable at idx as a Variant object.

Example:

```
-- Use with a TPC 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
```

HTTP

Not currently available.

JavaScript

Not currently available.

Trigger Number

Returns the triggers in the project. Show

Lua

```
get_trigger_number()
```

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

HTTP

Not currently available.

JavaScript

Not currently available.

Resources

Use to locate resources in the controller's memory. Show

Lua

```
get_resource_path(filename)
```

Argument	Type	Example
filename	string	'settings.txt'

Returns a path to the resource filename.

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Content Target

Returns information about a Content Target in the project. Show

Lua

On a VLC

get_content_target(compositionNum)

On a VLC+

```
get_content_target(compositionNum, type)
```

- compositionNum is the usernumber of the composition to return
- type is the type of target within the composition to return (PRIMARY, SECONDARY, OVERLAY_1, OVERLAY_2)

Returns a Content Target object with the following properties:

master_intensity_level	Variant
rotation_offset (VLC+ only)	float
x_position_offset (VLC+ only)	float
y_position_offset (VLC+ only)	float

Example:

```
target = get_content_target(1)

current_level = target.master_intensity_level

target = get_content_target(1,PRIMARY)

current_angle = target.rotation_offset
```

HTTP

Not currently available.

JavaScript

Not currently available.

Config

Returns information about a controller's Configuration. Show

Lua

`get_log_level()`

Returns the current log level of the controller

`get_syslog_enabled()`

Returns true if Syslog is enabled, false otherwise

`get_syslog_ip_address()`

Returns the IP address of the Syslog server

`get_ntp_enabled()`

Returns true if NTP is enabled

`get_ntp_ip_address()`

Returns the IP address of the NTP server

Example:

HTTP

`GET /api/config`

Returns an object with the following properties:

Property	Return Type	Return Example
ip	string	"192.168.1.3"
subnet_mask	string	"255.255.255.0"
gateway	string	"192.168.1.1"
dhcp_enabled	boolean	true
name_server_1	string	"192.168.1.1"
name_server_2	string	"8.8.8.8"
http_port	integer	80
https_port	integer	443
year	integer	2019
month	integer (1-12)	4
day	integer (1-31)	25
hour	integer (0-23)	13
minute	integer (0-59)	21
second	integer (0-59)	46
watchdog_enabled	boolean	true
log_level	integer (1-5)	3
syslog_enabled	boolean	true
syslog_ip	string	"192.168.1.2"

ntp_enabled	boolean	true
ntp_ip	string	"192.168.1.1"

JavaScript

```
get_config(callback)
```

Returns the controller's configuration

The returned object has the same structure as in the HTTP call

API Actions

Below are the ways of changing properties or changing output on the controller. show

Start Timeline

Start a timeline in the project Show

Lua

```
get_timeline(timelineNum):start()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "start",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.start_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Start Scene

Start a scene in the project Show

Lua

```
get_scene(sceneNum):start()
```

Argument	Type	Example
sceneNum	integer	1

HTTP

```
POST /api/scene
{
   "action": "start",
   "num": sceneNum
}
```

Argument	Type	Example
sceneNum	integer	1

JavaScript

```
Query.start_scene({ "num": sceneNum}, callback)
```

Argument	Type	Example
sceneNum	integer	1

Release Timeline

Release a timeline in the project Show

Lua

```
get_timeline(timelineNum):release([fade])
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

HTTP

```
POST /api/timeline
{
  "action": "release",
  "num": timelineNum[,
  "fade": fade]
}
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

JavaScript

```
Query.release_timeline({ "num": timelineNum[, "fade": fade]}, callback)
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

Release Scene

Release a scene in the project Show

Lua

```
get_scene(sceneNum):release([fade])
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

HTTP

```
POST /api/scene
{
  "action": "release",
  "num": sceneNum[,
  "fade": fade]
}
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

JavaScript

```
Query.release_scene({ "num": sceneNum[, "fade": fade]}, callback)
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

Toggle Timeline

Toggle a timeline in the project (if it is running, stop it, and if it is not running, start it) Show

Lua

```
get_timeline(timelineNum):toggle([fade])
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

HTTP

```
POST /api/timeline
{
  "action": "toggle",
  "num": timelineNum[,
```

```
  "fade": fade]
}
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

JavaScript

```
Query.toggle_timeline({ "num": timelineNum[, "fade": fade]}, callback)
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

Toggle Scene

Toggle a scene in the project (if it is running, stop it, and if it is not running, start it) Show

Lua

```
get_scene(sceneNum):toggle([fade])
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

HTTP

```
POST /api/scene
{
  "action": "toggle",
  "num": sceneNum[,
  "fade": fade]
}
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

JavaScript

```
Query.toggle_scene({ "num": sceneNum[, "fade": fade]}, callback)
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

Pause Timeline

Pause a timeline in the project Show

Lua

```
get_timeline(timelineNum):pause()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "pause",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.pause_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Resume Timeline

Resume a timeline in the project Show

Lua

```
get_timeline(timelineNum):resume()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "resume",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.resume_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Pause All

Pause all timelines in the project Show

Lua

```
pause_all()
```

HTTP

```
POST /api/timeline

{

 "action": "pause"

}
```

JavaScript

```
Query.pause_all(callback)
```

Resume All

Resume all timelines in the project Show

Lua

```
resume_all()
```

HTTP

```
POST /api/timeline

{

 "action": "resume"

}
```

JavaScript

```
Query.resume_all(callback)
```

Release All

Release all timelines, scenes or timelines, scenes and overrides in the project Show

Lua

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

release_all_timelines([fade,] [group])

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

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

HTTP

```
POST /api/release_all

POST /api/timeline

POST /api/scene

{

  "action": "release"[, (not required for release all)

  "group": group][,

  "fade": fade]

}
```

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

JavaScript

```
release_all_timelines({["fade": fade]}, callback)

release_all_scenes({["fade": fade]}, callback)

release_all({ ["fade": fade,] ["group": group] }, callback)
```

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

Set Timeline Rate

Set the rate of a timeline in the project Show

Lua

```
get_timeline(timelineNum):set_rate(rate)
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

HTTP

```
POST /api/timeline

{

  "action": "set_rate",

  "num": timelineNum,

  "rate": rate

}
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

JavaScript

```
Query.set_timeline_rate({"num": timelineNum, "rate": rate }, callback)
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

Set Timeline Position

Set the position of a timeline in the project Show

Lua

```
get_timeline(timelineNum):set_position(position)
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

HTTP

```
POST /api/timeline

{

  "action": "set_position",

  "num": timelineNum,

  "position": position

}
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

JavaScript

```
Query.set_timeline_position({"num": timelineNum, "position": position }, call-
back)
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

Enqueue Trigger

Fire a trigger in the project Show

Lua

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

Argument	Type	Example
num	integer	1
var...	comma separated variables	1,2,"string"

Example

```
enqueue_trigger(1,1,2,"string")
```

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

Enqueues a trigger without testing its conditions first (it will always be fired).

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

```
force_local_trigger(num[, var...]
```

- The "local" functions will only enqueue/force the desired trigger on the controller that ran the function, so will not propagate.

Example

```
if get_current_controller().number == 1 then
  enqueue_local_trigger(100,"Fired on Controller 1")
end
```

HTTP

```
POST /api/trigger
{
  "num": num[,
  "var": var...][,
  "conditions": test_conditions]
}
```

Argument	Type	Example
num	integer	1
var...	comma separated variables	1,2,"string"
test_conditions	boolean	true

- num - the trigger number to enqueue
- var... - 0 or more variables to pass to the trigger
- test_conditions - Should the conditions on the trigger be tested?

Example

String:

```
"var":'"String"'
```

Multiple Integers:

```
"var":'1,2,3'
```

Multiple Strings:

```
"var":'"string1","string2","string3"'
```

JavaScript

```
Query.fire_trigger({"num": num[, "var": var...][, "conditions": test_conditions]
}, callback)
```

Argument	Type	Example
num	integer	1
var...	comma separated variables	'1,2,"string"'
test_conditions	boolean	true

- var... - 0 or more variables to pass to the trigger. If passing multiple variables, they must be a single string surrounded by single quotes ('), string variables should be surrounded by double quotes (").

Run Script

Run a script or parse into the command line on the controller Show

Lua

Not currently available.

HTTP

```
POST /api/cmdline

{

  "input": chunk,

}
```

NOTE: returns "Executed" if successful, or an error string if not

Argument	Type	Example
chunk - the script to parse or run	string	"tl = 1 get_timeline(tl):start()"

JavaScript

```
Query.run_command({ "input": chunk }, callback)
```

NOTE: returns "Executed" if successful, or an error string if not

Argument	Type	Example
chunk - the script to parse or run	string	"tl = 1 get_timeline(tl):start()"

Hardware Reset

Reset the controller (power reboot) Show

Lua

Not currently available.

HTTP

```
POST /api/reset
```

JavaScript

Not currently available.

Master Intensity

Master the intensity of a group or content target (applied as a multiplier to output levels) Show

Lua

Non-VLC

```
get_group(groupNum):set_master_intensity(level[, fade[, delay]])
```

VLC

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

VLC+

```
get_content_target(compositionNum, type):set_master_intensity(level, [fade,
[delay]])
```

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', 'overlay_1', 'overlay_2'
level	float or integer (0-255)	128 or 0.5
fade	float	2.0
delay	float	2.0

Example:

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

HTTP

Non-VLC

```
POST /api/group

{

 "action": "master_intensity",

    "num": groupNum,

    "level": level,

   ["fade": fade,]

   ["delay": delay]

}
```

VLC/VLC+

```
POST /api/content_target
```

```
{
"action": "master_intensity",
"level": level,
["fade": fade,]
["delay": delay,]
"type": type
}
```

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', 'overlay_1', 'overlay_2'
level	float or bounded integer	0.5 or "50:100"
fade	float	2.0
delay	float	2.0

JavaScript

Non-VLC

```
master_intensity({ "num": groupNum, "level": level, ["fade": fade,] ["delay": delay] }, callback)
```

VLC/VLC+

master_content_target_intensity({ "type":type, "level": level, ["fade": fade,] ["delay": delay] }, callback)

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', 'overlay_1', 'overlay_2'
level	float or bounded integer	0.5 or "50:100"
fade	float	2.0
delay	float	2.0

Example:

```
Query.master_intensity({"num":1,"level":"50:100","fade":3) -- master group 1 to 50% (50/100 = 0.5) in 3 seconds)
```

NOTE: Group 0 will master the intensity of the 'All Fixtures' group. This can also be used on VLC family projects to master the intensity of the entire unit.

Set RGB

Set the Intensity, Red, Green, Blue levels for a fixture or group. Show

Lua

```
get_fixture_override(num)
get_group_override(num)
```

```
:set_irgb(intensity, red, green, blue, [fade, [path]])
:set_intensity(intensity, [fade, [path]])
:set_red(red, [fade, [path]])
:set_green(green, [fade, [path]])
:set_blue(blue, [fade, [path]])
:set_temperature(temperature, [fade, [path]])
```

Argument	Type	Example
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated", "Damped", "Overshoot", "Col At Start", "Col At End", "Int At Start", "Int At End", "Colour First", "Intensity First"

Example:

```
ov = get_fixture_override(1) -- Get fixture 1
ov:set_irgb(255, 255, 0, 0) -- Set the fixture to Red
```

HTTP

```
PUT /api/override
{
   "target": target,
   "num": num,
   ["intensity": intensity,]
   ["red": red,]
   ["green": green,]
   ["blue": blue,]
   ["temperature": temperature,]
```

```
   ["fade": fade,]
   ["path": path]
}
```

Argument	Type	Example
target	string (from options)	"group", "fixture"
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated", "Damped", "Overshoot", "Col At Start", "Col At End", "Int At Start", "Int At End", "Colour First", "Intensity First"

Javascript

```
set_group_override({ "num": num, ["intensity": intensity,] ["red": red,]
["green": green,] ["blue": blue,] ["temperature": temperature,] ["fade": fade,]
["path": path] }, callback)
```

```
set_fixture_override({ "num": num, ["intensity": intensity,] ["red": red,]
["green": green,] ["blue": blue,] ["temperature": temperature,] ["fade": fade,]
["path": path] }, callback)
```

Argument	Type	Example
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated", "Damped", "Overshoot", "Col At

Start", "Col At
End", "Int At
Start", "Int At
End", "Colour
First", "Intensity
First"

Example:

```
Query.set_fixture_override({ "num": 1, "intensity": 255, "red": 255, "green":
0, "blue": 0});
```

NOTE: Group 0 will set the levels of the 'All Fixtures' group

Clear RGB

Remove any overrides on fixtures or groups. Show

Lua

```
get_fixture_override(num)

get_group_override(num)

   :clear([fade])

clear_all_overrides([fade])
```

Argument	Type	Example
num - group or fixture	integer	1
fade	float	2.0

Example:

```
ov = get_fixture_override(1) -- Get fixture 1

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

HTTP

```
DELETE /api/override

{

   ["target": target,]

   ["num": objectNum,]

   ["fade": fade]

}
```

If num is not included, target is ignored and all overrides are cleared.

Argument	Type	Example
target	string (from options)	"group" or "fixture"
num - group or fixture	integer	1
fade	float	2.0

JavaScript

```
clear_group_overrides({ ["num" :num,] ["fade": fade] }, callback)
clear_fixture_overrides({ ["num" :num,] ["fade": fade] }, callback)
clear_overrides({ ["fade": fade] }, callback)
```

Argument	Type	Example
num	integer	1
fade	float	2.0

Example:

```
Query.clear_all_overrides({"fade":3})
```

Set Text Slot

Set the value of a text slot used in the project. Show

Lua

```
set_text_slot(name, value)
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

Example:

```
set_text_slot("myTextSlot", "Hello World!")
```

HTTP

```
PUT /api/text_slot
{
   "name": name,
   "value": value
}
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

JavaScript

```
set_text_slot({"name": name, "value": value}, callback)
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

Example:

```
Query.set_text_slot("name:"myTextSlot", "value":"Hello World!")
```

Set BPS Button LED

Set the effect and intensity on BPS button LEDS. Show

Lua

```
get_bps(num):set_led(button, effect, [intensity], [fade])
```

Argument	Type	Example
num	integer	1
button	integer	1
effect	OFF, ON, SLOW_FLASH, FAST_FLASH, DOUBLE_FLASH, BLINK, PULSE, SINGLE, RAMP_ON, RAMP_OFF	FAST_ FLASH
intensity	integer (1-255)	255
fade	float	0.0

Example:

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

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Touch Control Value

Set the value on a Touch Slider or Color Picker. Show

Lua

```
set_control_value(name, [index,] value[, emitChange])
```

Argument	Type	Example
name - control Key	string	"slider001"
index - axis of movement (slider has 1, colour picker has 3)	integer (1-3) (default 1)	1
value	integer (0-255)	128
emitChange	boolean (default false)	false

Example:

```
set_control_value("slider001", 1, 128) -- set slider001 to half and don't fire associated triggers
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Touch Control State

Set the state on a Touch control. Show

Lua

```
set_control_state(name, state)
```

Argument	Type	Example
name - control Key	string	"slider001"
state - the state name form Interface	string (from options in Interface)	"Green"

Example:

```
set_control_state("slider001", "Green") -- set slider001 to a state called
"Green"
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Touch Control Caption

Set the caption on a Touch control. Show

Lua

```
set_control_caption(name, caption)
```

Argument	Type	Example
name - control Key	string	"button001"
caption - text to display	string	"On"

Example:

```
set_control_caption("button001", "On") -- set button001's caption to "On"
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Touch Page

Change the page on a Touch interface. Show

Lua

```
set_interface_page(number[, transition])
```

Argument	Type	Example
number	integer	4
transition	SNAP, PAN_LEFT, PAN_RIGHT	PAN_LEFT

Example:

```
set_interface_page(4) -- change the page on the TPC's interface to page 4
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Disable Page

Disable the touchscreen. Show

Lua

```
set_interface_enabled([enable])
```

Argument	Type	Example
enable	boolean (default true)	true

Example:

```
set_interface_enabled(false) -- disable the TPC's touch screen

set_interface_enabled() -- enable the TPC's touch screen
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Lock Touch Device

Lock the Touch Device (requires Lock code to be set within Interface). Show

Lua

```
set_interface_locked([lock])
```

Argument	Type	Example
lock	boolean (default true)	true

Example:

```
set_interface_enabled(false) -- disable the TPC's touch screen
set_interface_enabled() -- enable the TPC's touch screen
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Transition Content Target

Move or rotate a Content Target. Show

Lua

```
get_content_target(compositionNum, type)
  :transition_rotation([offset], [count], [period], [delay], [useShortestPath])
  :transition_x_position([offset], [count], [period], [delay])
  :transition_y_position([offset], [count], [period], [delay])
```

Argument	Type	Example
compositionNum	integer	1
type	PRIMARY, SECONDARY, OVERLAY_1, OVERLAY_2	PRIMARY
offset	integer	10
count	integer	1

period	integer	5
delay	integer	0
useShortestPath	boolean (default false)	false

Example:

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

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Transition Adjustment Target

Move or rotate a Adjustment Target. Show

Lua

```
get_adjustment(num)
  :transition_rotation([offset], [count], [period], [delay], [useShortestPath])
  :transition_x_position([offset], [count], [period], [delay])
  :transition_y_position([offset], [count], [period], [delay])
```

Argument	Type	Example
num	integer	1
offset	integer	10
count	integer	1
period	integer	5
delay	integer	0
useShortestPath	boolean (default false)	false

Example:

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

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Beacon Controller

Beacons the controller (flashes Status LEDs or screen). Show

Lua

Not currently available.

HTTP

```
POST /api/beacon
```

JavaScript

```
toggle_beacon(callback)
```

Example:

```
Query.toggle_beacon()
```

Output to Log

Writes a message to the controller's Log. Show

Lua

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

Argument	Type	Example
level	option (LOG_DEBUG, LOG_TERSE, LOG_NORMAL, LOG_EXTENDED, LOG_VERBOSE, LOG_CRITICAL, default LOG_NORMAL)	LOG_CRITICAL
message	string	"Some message to log."

Example:

log(LOG_CRITICAL, "This is a criticial message!") -- logs the message at Critical log level

log("This is a normal message.") -- logs the message at Normal log level.

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Send variables to Web Interface

Sends data to the web interface in a JSON Object. Show

Lua

```
push_to_web(name, value)
```

Argument	Type	Example
name	string	"myVar"
value	variable	"Some value"

Example:

```
myVar = 15

push_to_web("myVar", myVar) -- will push the object {"myVar": 15}
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Park a Channel

Parks an output channel at a specified level. Show

Lua

```
Universe:park(channel, value)
```

Argument	Type	Example
Universe	Universe object	get_dmx_universe(1)
channel	integer (1-512)	1
value	integer (0-255)	128

Example:

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

HTTP

```
POST /api/channel

{
```

```
    "universe": universeKey,
    "channels": channelList,
    "level": level
}
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"
level	integer (0-255)	128

JavaScript

```
park_channel({ "universe": universeKey, "channels": channelList, "level": level
}, callback)
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"
value	integer (0-255)	128

Example:

```
park_channel({ "universe": "dmx:1","channels": 1, "level":128}); // Park chan-
nel 1 of DMX Universe 1 at 128 (50%)
```

Unpark a Channel

Unparks an output channel. Show

Lua

```
Universe:unpark(channel)
```

Argument	Type	Example
Universe	Universe object	get_dmx_universe(1)
channel	integer (1-512)	1

Example:

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

HTTP

```
DELETE /api/channel

{

    "universe": universeKey,

    "channels": channelList

}
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"

JavaScript

unpark_channel({ "universe": universeKey, "channels": channelList }, callback)

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"

Example:

```
unpark_channel({ "universe": "dmx:1","channels": 1}); //Unpark channel 1 of DMX Universe 1 (it will go back to normal output levels)
```

Disable an Output

This disables the output of a selected protocol from the chosen Controller. Show

Lua

```
disable_output(protocol)

enable_output(protocol)
```

Argument	Type	Example
protocol	option (DMX, PATHPORT, ARTNET, KINET, SACN, DVI, RIO_DMX)	DMX

Example:

disable_output(DMX) -- Disable the DMX output from the controller

HTTP

```
POST /api/output
{
    "protocol": protocol,
    "action": action
}
```

Argument	Type	Example
protocol	string ("dmx", "pathport", "art-net", "kinet", "sacn", "dvi", "rio-dmx")	"dmx"
action	string ("enable", "disable")	"disable"

JavaScript

```
disable_output({ "protocol": protocol }, callback)
enable_output({ "protocol": protocol }, callback)
```

Argument	Type	Example
protocol	string ("dmx", "pathport", "art-net", "kinet", "sacn", "dvi", "rio-dmx")	"dmx"

Example:

```
disable_output({ "protocol": "dmx"}); // Disable the DMX output
enable_output({ "protocol": "art-net"}); // Enable the Art-Net Output
```

Set Timeline Source Bus

Set the time source for a timeline. Show

Lua

```
Timeline:set_default_source()
Timeline:set_timecode_source(timecodeBus[, offset])
Timeline:set_audio_source(audioBus, band, channel[,peak])
```

Argument	Type	Example
Timeline	Timeline Object	get_timeline(1)
timecodeBus	TCODE_1 ... TCODE_6	TCODE_1
audioBus	AUDIO_1 ... AUDIO_4	AUDIO_1
band	integer (0=volume)	0
channel	LEFT, RIGHT or COMBINED	LEFT
peak	boolean (default false)	false

Example:

```
get_timeline(1):set_timecode_source(TCODE_1) -- Set the timecode source of timeline 1 to timecode bus 1
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Enable Timecode Bus

Enables or disables a timecode bus. Show

Lua

```
set_timecode_bus_enabled(bus[, enable])
```

- bus is the timecode bus to enable or disable (TCODE_1 ... TCODE_6)
- enable determines whether the bus should be enabled or disabled (boolean, default true)

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Digital Output

Sets the output of a RIO to on or off. Show

Lua

```
get_rio(type, num):set_output(outputNum, state)
```

- type can be RIO44 or RIO08 (RIO80 has no outputs)
- num is the remote device number
- outputNum is the number of the output
- state is the state to set the output to and can be any of: 0, 1, true, false, ON, OFF

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Log Level

Changes the log level of the controller, showing more or less information on the Log page. Show

Lua

set_log_level(log_level)

Sets the log level of the controller to log_level

- log_level can be LOG_DEBUG, LOG_TERSE, LOG_NORMAL, LOG_EXTENDED, LOG_VERBOSE, LOG_CRITICAL or 0-5.

HTTP

Not currently available.

Use Edit Config

JavaScript

Not currently available.

Use Edit Config

Trigger Number

Sets the trigger number. Show

Lua

get_trigger_number()

Returns the number as an integer of the trigger that ran this script. Will return nil if run from another context.

get_trigger_number()

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

Edit Config

Edits the configuration of the current controller. Show

Lua

Not currently available.

HTTP

```
POST /api/config
{
   "ip": ipAddress,
   "subnet_mask": subnetMask,
   "gateway": gateway,
   "dhcp_enabled": dhcpEnabled,
   "name_server_1": dnsServer1IPAddress,
   "name_server_2": dnsServer2IPAddress,
   "http_port": httpPort,
   "https_port": httpsPort,
```

```
   "year": year,
   "month": month,
   "day": day,
   "hour": hour,
   "minute": minute,
   "second": second,
   "watchdog_enabled": watchdogEnabled,
   "log_level": logLevel,
   "syslog_enabled": syslogEnabled,
   "syslog_ip": syslogIpAddress,
   "ntp_enabled": ntpEnabled,
   "ntp_ip": ntpIpAddress,
   "password": password
}
```

Argument	Type	Example
ipAddress	string	"192.168.1.2"
subnetMask	string	"255.255.255.0"
gateway	string	"192.168.1.1"
dhcpEnabled	boolean	true
dnsServer1IPAddress	string	"192.168.1.1"
dnsServer2IPAddress	string	"8.8.8.8"
httpPort	integer	80
httpsPort	integer	443
year	integer	2019
month	integer (0-11)	4
day	integer (1-31)	25
hour	integer (0-23)	13
minute	integer (0-59)	22
second	integer (0-59)	40
watchdogEnabled	boolean	true
logLevel	integer (0-5)	3
syslogEnabled	boolean	true
syslogIpAddress	string	"192.168.1.4"
ntpEnabled	boolean	true
ntpIpAddress	string	"192.168.1.1"
password	string	"myPassword"

If the response is 200 OK, the response body will be

```
{
   "restart": restart
}
```

`restart` is a boolean. If true, the controller will reset imminently in order to apply the changes

JavaScript

```
edit_config(params, callback)
```

Sends a request to change the controller's configuration

params is an object of the same format as in the HTTP request

The callback parameter will contain the same object as a response to the HTTP request

API Subscriptions

Subscriptions allow data to be pushed to the web interface whenever there is a change within the project. show

Subscribe Timeline Status

Subscribes to changes in the timeline status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_timeline_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	'running'
onstage	boolean	true
position	number (milliseconds)	5000

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_timeline_status(function(t){
        alert(t.num + ": " + t.state)
})
```

Subscribe Scene Status

Subscribes to changes in the scene status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_scene_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	'running'
onstage	boolean	true

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_scene_status(function(s){
        alert(s.num + ": " + s.state)
})
```

Subscribe Group Status

Subscribes to changes in group level, as set by the Master Intensity action (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_group_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
name	string	'Group 1'
level	integer (0-255)	128

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_group_status(function(g){
        alert(g.num + ": " + g.level)
})
```

Subscribe Remote Device Status

Subscribes to changes in Remote Device Online/Offline Status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_remote_device_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
type	string ('RIO 08', 'RIO 44', 'RIO 80','RIO D', 'RIO A', 'BPS')	'Group 1'
online	boolean	true
serial	string (of serial number)	'001001'

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_remote_device_status(function(r){
        alert(r.num + ": " + r.level)
})
```

Subscribe Beacon

Subscribes to Beacons (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_beacon(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
on	boolean	true

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_beacon(function(b){
        if (b.on){
                alert("Beacon Turned On")
        else {
                alert("Beacon Turned Off")
        }
})
```

Subscribe Lua

The receiver for the push_to_web() Lua function. Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_lua(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
key	as defined by push_to_web()	value

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_lua(function(l){
        key = Object.keys(l)[0]
        value = l.key
        alert(key + ": " + value)
})
```

API Objects

Below are the helper functions and objects in the project. show

Variant

A Lua object that allows a type and range to be associated with a variable. Show

Lua

See here.

HTTP

Not currently available.

JavaScript

Not currently available.

DateTime

A Lua object containing time data. Show

Lua

The DateTime object contains the following properties:

Property	Return Type	Return Example
.year	integer	2017
.month	integer (0-11)	5
.monthday	integer (0-30)	8
.weekday	integer(0-6)	1
.hour	integer(0-23)	13
.minute	integer (0-59)	21
.second	integer (0-59)	46
.utc_timestamp	integer	1494249706
.time_string	string	
.date_string	string	

HTTP

Not currently available.

JavaScript

Not currently available.

Printing an enum

Lua functions to convert integers returned from some functions as text. Show

Lua

```
digital_input_to_string()
```

```
button_state_to_string()
```

Examples:

```
log(digital_input_to_string(get_input(1)))
```

```
str = button_state_to_string(get_bps(1):get_state(1))
```

HTTP

Not currently available.

JavaScript

Not currently available.

API v4

The Pharos system includes multiple API options:

- Lua (used internally with Conditions and Actions)
- HTTP (used with external devices/software to communicate with a controller)
- JavaScript (used with Custom Web Interfaces)

These APIs have been unified to simplify their use as much as possible.

Glossary:

- Object - A collection of key value pairs e.g. "name" = "Controller 1" (syntax will differ between languages).
- String - A series of characters e.g."Th1s_is-4(string)"
- Number - Any whole or floating point(decimal) number e.g. 1,2,3,1.5,12.3456)
- Integer - A whole number
- Bounded integer - An integer with a range (e.g. 10:100 = 10%)
- Float/real/number - A decimal number (e.g. 3.2 or 1.0)
- JSON - (JavaScript Object Notation) a way of transferring information in the form of a JavaScript Object
- GET - A HTTP method to request data from a server
- POST - A HTTP method to request data in a more secure way
- PUT - A HTTP method to send data to a server
- Variant - See here
- [] - anything shown within square brackets is optional. The square brackets should be omitted if the optional section is used.
- callback - A function to run when the javascript function has been run, or a reply has been received.

HTTP Requests

Please note, when a HTTP POST request is sent, it must include a Content-Type header set to "application/json", otherwise it will be treated as invalid.

API Queries

Below are the ways of getting data from the controller. show

System

Returns data about the controller. show

Lua

The system namespace has the following properties:

Property	Return type	Return Example
.hardware_type	string	"lpc"
.channel_capacity	integer	512
.serial_number	string	"006321"
.memory_total	string	"12790Kb"
.memory_used	string	"24056Kb"
.memory_free	string	"103884Kb"
.storage_size	string	"1914MB"

.bootloader_version	string	"0.9.0"
.firmware_version	string	"2.14.0"
.reset_reason	string	"Software Reset"
.last_boot_time	DateTime object	
.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"

Example:

```
capacity = system.channel_capacity

boot_time = system.last_boot_time.time_string
```

HTTP

```
GET /api/system
```

Returns an object with the following properties:

Property	Return type	Return Example
hardware_type	string	"LPC"
channel_capacity	integer	512
serial_number	string	"006321"
memory_total	string	"12790Kb"
memory_used	string	"24056Kb"
memory_free	string	"103884Kb"
storage_size	string	"1914MB"
bootloader_version	string	"0.9.0"
firmware_version	string	"2.14.0"
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"

JavaScript

```
get_system_info(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_system_info( function(system){

   var capacity = system.channel_capacity

}
```

Project

Returns data about the project. Show

Lua

get_current_project()

Returns an object with the following properties:

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

Example:

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

HTTP

`GET /api/project`

Returns an object with the following properties:

Property	Return type	Return Example
name	string	"Help Project"
author	string	"Pharos"
filename	string	"help_project_v1.pd2"
unique_id	string	"{6b48627a-1d5e-4b2f-81e2-481e092a6a79}"
upload_date	string	"2017-01-30T15:19:08"

JavaScript

`get_project_info(callback)`

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_project_info( function(project){
 var author = project.author
}
```

Replication

Returns data about the install replication. Show

Lua

get_current_replication()

Returns an object with the following properties:

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

Example:

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

HTTP

GET /api/replication

Returns an object with the following properties:

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

JavaScript

get_replication(callback)

Returns an object with the same properties as in the HTTP call

Location

Returns data about the install location. Show

Lua

get_location()

Returns an object with the following properties:

Property	Return type	Return Example
lat	float	51.512
long	float	-0.303

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Network 2

Returns data about the Network 2 (Protocol) Interface. Show

Lua

The protocol_interface namespace has the following properties:

Property	Return type	Return 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"

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Time

Returns data about the time stored in the controller. Show

Lua

The time namespace has the following functions which 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()

and the following properties

Property	Return Type	Return Example
is_dst	boolean	
gmt_offset	string	

Each function returns a DateTime object, with the following properties:

Property	Return Type	Return Example
.year	integer	2017
.month	integer (1-12)	5
.monthday	integer (1-31)	8
.weekday	integer(1-7)	1
.hour	integer(0-23)	13
.minute	integer (0-59)	21
.second	integer (0-59)	46
.utc_timestamp	integer	1494249706
.time_string	string	
.date_string	string	

Example:

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

HTTP

```
GET /api/time
```

Returns an object with the following properties:

Property	Return Type	Return Example
datetime	string	"01 Feb 2017 13:44:42"
local_time	integer (controller's local time in milliseconds)	1485956682
uptime	integer (time since last boot)	493347

JavaScript

```
get_current_time(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_current_time( function(time){
 var uptime = time.uptime
}
```

Timeline

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

Lua

```
get_timeline(timelineNum)
```

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

The returned object has the following properties

Property	Return Type	Return Example
name	string	"Timeline 1"
group	string ('A', 'B', 'C', 'D','')	'A'
length	integer	10000
source_bus	integer (equivalent to constants: DEFAULT, TCODE_1 ... TCODE_6, AUDIO_1 ... AUDIO_4)	1
timecode_format	string	"SMPTE30"
audio_band	integer (0 is equivalent to constant VOLUME)	0
audio_channel	integer (equivalent to constants: LEFT, RIGHT or COMBINED)	1
audio_peak	boolean	false
time_offset	integer	5000
state	integer (equivalent to constants: Timeline.NONE, Timeline.RUNNING, Timeline.PAUSED, Timeline.HOLDING_AT_END, Timeline.RELEASED)	1

onstage	boolean	true
position	integer	5000
priority	integer (equivalent to constants: HIGH_PRIORITY, ABOVE_NORMAL_PRIORITY, NORMAL_PRIORITY, BELOW_NORMAL_PRIORITY or LOW_PRIORITY)	0
custom_properties	(table, keys and values correspond to custom property names and values)	

Example:

```
tl = get_timeline(1)

name = tl.name

state = tl.state

if (tl.source_bus == TCODE_1) then

    -- do something

end
```

HTTP

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

- num can be used to filter which timelines are returned and can be a single number or a string representing the required timelines (e.g. "1,2,5-9")

Returns an object with the following properties:

timelines array of timeline objects

Each timeline object contains the following properties:

Property	Return Type	Return Example
num	integer	1
name	string	"Timeline 1"
group	string('A', 'B', 'C', 'D' or empty)	"A"
length	integer	10000
source_bus	string ('internal', 'timecode_1',...'timecode_6', 'audio_1',...'audio_4')	100
timecode_format	string	"SMPTE30"
audio_band	integer (0 is volume band)	1
audio_channel	string ('left', 'right', 'combined')	"combined"
audio_peak	boolean	false
time_offset	integer	5000
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	"running"
onstage	boolean	true
position	integer	10000
priority	string ('high', 'above_normal', 'normal', 'below_normal', 'low')	"normal"
custom_properties	(object, properties and property values correspond to custom property names and values)	

JavaScript

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

- num can be used to filter which timelines are returned and is defined as a JSON object which can contain a single number or a string representing the required timelines (e.g. "1,2,5-9")

Returns an array of timelines in the same way as the HTTP call

Example:

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

Scene

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

Lua

```
get_scene(sceneNum)
```

Returns a single Scene object for the Scene with user number SceneNum.

The returned object has the following properties

Property	Return Type	Return Example
name	string	"Scene 1"
group	string (A-H) or empty	"A"
state	integer (equivalent to constants: Scene.NONE, Scene.STARTED, Scene.RELEASED)	1
onstage	boolean	false
custom_properties	(object, properties and property values correspond to custom property names and values)	

Example:

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

HTTP

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

num can be used to filter which scenes are returned and can be a single number or array

Returns an object with the following properties:

scenes array of scene objects

Each scene object contains the following properties:

Property	Return Type	Return Example
name	string	"Scene 1"
num	integer	1
state	string ('none', 'started')	"none"
onstage	boolean	false

JavaScript

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

filter may contain a num property which is used to filter which scenes are returned

Returns an array of scenes in the same way as the HTTP call

Example:

```
Query.get_scene_info( function(s){
 var name = s.scenes[0].name //name of the first timeline
}, {"num":"1-4"})
```

Group

Returns data about the groups in the project. Show

Lua

```
get_group(groupNum)
```

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

The returned object has the following properties:

Property	Return Type	Return Example
name	string	"Group 1"
master_intensity_level	Variant	

Example:

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

HTTP

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

num can be used to filter which groups are returned and can be a single number or array

Returns an object with the following properties:

groups array of group objects

Each group object contains the following properties:

Property	Return Type	Return Example
num	integer (only included for user created groups)	1

name	string	"Group 1"
level	integer (0-100)	100

JavaScript

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

filter may contain a num property which is used to filter which groups are returned

Returns an array of groups in the same way as the HTTP call

Example:

```
Query.get_group_info( function(g){
 var name = g.groups[0].name //name of the first timeline
}, {"num":"1-4"})
```

NOTE: Group 0 will return data about the 'All Fixtures' group

Controller

Returns data about the controller. Show

Lua

```
get_current_controller()
```

Returns either the controller, the script, or module it is being run on.

```
get_network_primary()
```

Returns the controller which is currently the network primary.

Both return an object containing the following properties:

Property	Return Type	Return Example
number	integer	1
name	string	"Controller 1"

Example:

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

```
is_controller_online(controllerNumber)
```

Returns true if the controller with user number controllerNum has been discovered, and false otherwise

Example:

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

HTTP

```
GET /api/controller
```

Returns an object with the following properties:

controllers array of controller objects (one for each controller in the project)

Each controller object contains the following properties:

Property	Return Type	Return Example
num	number	1
type	string	"LPC"
name	string	"Controller 1"
serial	string	"009060"
ip_address	string (if the controller is discovered)/empty (if the controller is not discovered or is the queried controller)	"192.168.1.3" or ""
online	boolean	true
is_network_primary	boolean	true

JavaScript

```
get_controller_info(callback)
```

Returns an array of controllers in the same way as the HTTP call

Example:

```
Query.get_controller_info( function(controller){
 var name = controller[0].name // name of the first controller
}
```

Temperature

Returns data about the controller's temperature. Show

Lua

```
get_temperature()
```

Returns an object with the following properties

Property	Return Type	Return Example
sys_temp	number (only for LPC X and VLC/VLC+)	40
core_temp	number (only for LPC X and VLC/VLC+)	44
ambient_temp	number (only for TPC, LPC X rev 1)	36.900001525878906
cc_temp	number (only for LPC X rev 2 and VLC/VLC+)	44
gpu_temp	number (only for VLC/VLC+)	44

Example:

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

HTTP

```
GET /api/temperature
```

Returns an object with the following properties:

Property	Return Type	Return Example
sys_temp	number (only for LPC X and VLC/VLC+)	40
core1_temp	number (only for LPC X and VLC/VLC+)	44
core2_temp	number (only for LPC X rev 1	44
ambient_temp	number (only for TPC, LPC X rev 1)	36.900001525878906
cc_temp	number (only for LPC X rev 2 and VLC/VLC+)	44
gpu_temp	number (only for VLC/VLC+)	44

JavaScript

```
get_temperature(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_temperature( function(temp){
 var ambient = temp.ambient_temp // ambient temperature of the controller
}
```

Remote Device

Returns data about the Remote Device/s in the project. Show

Lua

```
get_rio(type, num):get_input(inputNum)
```

- type can be RIO80, RIO44 or RIO08
- num is the remote device number
- inputNum is the number of the input

Returns a boolean if the input is set to Digital or Contact Closure, or an integer if the input is set to Analog.

Example:

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

```
get_rio(type, num):get_output(outputNum)
```

- type can be RIO80, RIO44 or RIO08
- num is the remote device number
- outputNum is the number of the output

Returns a boolean showing the current state of the output.

Example:

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

`get_bps(num):get_state(buttonNum)`

- num is the BPS number
- buttonNum is the number of the button

Returns the state of the button, which can be RELEASED, PRESSED, HELD or REPEAT

Example:

```
bps = get_bps(1)

btn = bps:get_state(1)
```

HTTP

`GET /api/remote_device`

Returns an array of all remote devices in the project.

The returned object has the following structure

remote_devices array of Remote Device objects

Each Remote Device object contains the following properties:

Property	Return Type	Return Example
num	integer	1
type	string ('RIO08', 'RIO44', 'RIO80', 'BPS', 'BPI', 'RIO A', 'RIO D')	"RIO 44"
serial	array (all discovered serial number for the address and type)	["001234"]
outputs	array (of Output objects, only present for RIO44 and RIO08 that are on the queried controller)	[{"output":1,"value":true},{"output":2,"value":true},{"output":3,"value":true},{"output":4,"value":true}]
inputs	array (of Input objects, only present for RIO44 and RIO80 that are 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 (if the remote device is detected as being online)	true

The Output object has the following properties:

Property	Return Type	Return Example
output	integer	1
state	boolean (true means the output is on, false means it is off)	false

The Input object has the following properties:

Property	Return Type	Return Example
input	integer	1

type	string ('Analog', 'Digital', 'Contact Closure')	'Digital'
value	integer or bool (depends on type)	true

JavaScript

```
get_remote_device_info(callback)
```

Returns an array of all remote devices in the project with the same properties as in the HTTP call.

Example:

```
Query.get_remote_device_info( function(remote){
 var type = remote[0].type // type of the first remote device
}
```

Text Slots

Returns data about the text slots in the project. Show

Lua

```
get_text_slot(slotName)
```

- slotName is the name of the text slot

Returns the value of slotName

Example:

```
log(get_text_slot("test_slot"))
```

HTTP

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

slotNames can be used to filter which text slots are returned and can be a single name or array

The returned object has the following structure

text_slots array of Text Slot objects

Each Text Slot object contains the following properties:

Property	Return Type	Return Example
name	string	"text"
value	string	"example"

JavaScript

```
get_text_slot(callback[, filter])
```

filter may contain a names property which is used to filter which text slot values are returned

Returns an array of all text slots in the project with the same properties as in the HTTP call.

Example:

```
Query.get_text_slot( function(text){
 var value = text[0].value // value of the first text slot
```

```
}, {names: "test_slot1", "test_slot2"})
```

Get Log

Returns the log from the queried controller. Show

Lua

Not currently available.

HTTP

```
GET /api/log
```

The returned object has the following structure

Property	Return Type
log	string (containing the whole log of the controller)

JavaScript

Not currently available.

Protocol

Returns the protocols and universes being output from the queried controller. Show

Lua

Not currently available.

HTTP

```
GET /api/protocol
```

Returns all the universes on the queried controller

The returned object has the following structure

outputs	array of Protocol objects

Each Protocol object has the following properties:

Property	Return Type	Return Example
type	integer	1
name	string	"DMX"
disabled	boolean (whether the output has been disabled via an Action)	true
universes	array (Universe objects)	{"key":{"index":1},"name":"1"},{"key":{"index":2},"name":"2"}
dmx_proxy	DMX Proxy Object (where appropriate)	{ "ip_address": "192.168.1.17", "name": "Controller 1" }

Each Universe object has the following properties:

Property	Return Type	Return Example

name	string	"1"
key	Universe Key object	{"index":1}

Each DMX Proxy object has the following properties:

Property	Return Type	Return Example
name	string (name of the controller that is outputting this universe)	'Controller 1'
ip_address	string IP Address of the controller outputting this universe	'192.168.1.23'

The properties of the Universe Key object depends upon the type:

For DMX, Pathport, sACN and Art-Net:

Property	Return Type	Return Example
index	integer	1

For KiNET:

Property	Return Type	Return Example
kinet_port	integer	1
kinet_power_supply_num	integer	1

For RIO DMX:

Property	Return Type	Return Example
remote_device_num	integer	1
remote_device_type	integer (corresponding to values in query.js)	1

For EDN:

Property	Return Type	Return Example
remote_device_num	integer	1
port	integer	1

JavaScript

```
get_protocols(callback)
```

Returns all the universes on the queried controller, with the same structure as the HTTP response.

Output

Returns the levels being output from the queried controller. Show

Lua

```
get_dmx_universe(idx)
```

```
get_artnet_universe(idx)
```

```
get_pathport_universe(idx)
```

```
get_sacn_universe(idx)
```

- idx is the required universe number

```
get_kinet_universe(power_supply_num, port_num)
```

- power_supply_num is the power supply to return the output from
- port_num is the port to return the output from

These all return a Universe object, which has the following function

```
get_channel_value(chnl)
```

- chnl is the channel to get the value from

Example:

```
uni = get_dmx_universe(1) -- get DMX Universe 1

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

HTTP

```
GET /api/output?universe=universeKey
```

- universeKey is a string in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET, protocol:remoteDeviceType:remoteDeviceNum for RIO DMX and protocol:remoteDeviceNum:port for EDN.
- protocol can be dmx, pathport, sacn, art-net, kinet, rio-dmx or edn
- remoteDeviceType can be rio08, rio44 or rio80

Example:

GET /api/output?universe=dmx:1

GET /api/output?universe=rio-dmx:rio44:1

The returned object has the following structure

Property	Return Type	Return Example
channels	array (of channel values)	[0,0,0,0,0,0,0,0,0,255,255,255...255,0,255]
disabled	boolean (whether the output has been disabled via an Action)	true
proxied_tpc_name	string (only if controller is LPC, universe is DMX 2, DMX Proxy has been enabled and the TPC is offline)	'Controller 2'

JavaScript

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

Argument	Type	Example
universekey	string or an object containing protocol and either index, kinet_power_supply_num and kinet_port or remote_device_type and remote_device_num	dmx:1

- universeKey can be either a string, or an object containing protocol and either index, kinet_power_supply_num and kinet_port or remote_device_type and remote_device_num as received from get_protocols

Example:

```
Query.get_output({protocol:KINET, kinet_port:1, kinet_power_supply_num:1}, function(u){

   console.log(u)

})
```

```
Query.get_output({protocol:DMX, num:1}, function(u){
   console.log(u)
})

Query.get_output("dmx:1", function(u){
   console.log(u)
})
```

Returns an object with the same structure as in the HTTP call

Input

Returns the inputs on the queried controller. Show

Lua

`get_input(idx)`

Argument	Type	Example
idx	integer	1

Returns the value of the controllers input as a boolean or integer

Example:

```
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)
```

`get_dmx_input(chnl)`

Argument	Type	Example
chnl	integer	1

- chnl is the required channel number

Returns the value of the DMX input at channel chnl as an integer. If no DMX Input is detected, it will return nil.

HTTP

`GET /api/input`

The returned object has the following structure

Property	Return Type	Return Example

gpio	array (of Input objects, on LPC or TPC+EXT)	[{"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},{"input":5,"type":"Contact Closure","value":true},{"input":6,"-type":"Contact Closure","value":true},{"input":7,"type":"Contact Closure","value":true},{"input":8,"type":"Contact Closure","value":true}]
dmxIn	object (DMX Input object, if DMX Input is configured)	[0,0,0,0,0,0,0,0,0,255,255,255...255,0,255]

The Input object has the following properties:

Property	Return Type	Return Example
input	integer	1
type	string ('Analog', 'Digital', 'Contact Closure')	"Contact Closure"
value	integer or bool (depends on type)	true

The DMX Input object has the following properties:

Property	Return Type	Return Example
error	string (if DMX Input is configured but no DMX is received)	"No DMX received"
dmxInFrame	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"

JavaScript

Not currently available.

Trigger

Returns the triggers in the project. Show

Lua

Not currently available.

HTTP

```
GET /api/trigger?[type=[triggerType]]
```

- triggerType is a string defining the trigger types to be returned.

The returned object has the following structure

triggers	array (of Trigger objects)

The Trigger object has the following properties:

Property	Return Type	Return Example
type	string	"Startup"
num	integer	1
name	string	"Startup"
description	string	""
trigger_text	string	"At startup"
conditions	array (of Condition objects)	[{"text":"Before 12:00:00 every day"}]
actions	array (of Action objects	[{"text":"Start Timeline 1"}]

The Condition and Action objects have the following properties:

Property	Return Type	Return Example
text	string	"Start Timeline 1"

JavaScript

Not currently available.

Lua Variable

Returns the current value of the specified Lua variable. Show

Lua

Not currently available.

HTTP

```
GET /api/lua?variables=luaVariables
```

Argument	Type	Example
luaVariables	string or comma separated list	'myVar' or 'myVar, myVar2, myVar3'

Returns an object containing all the Lua variables requested and their values.

JavaScript

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

Argument	Type	Example
luaVariables	string or array	'myVar' or 'myVar, myVar2, myVar3'

Returns an object containing all the Lua variables requested and their values.

Example:

```
Query.get_lua_variables("myVar", function(lua){
 var value = lua.myVar
}
```

Trigger Variable

Returns the value of a variables from the trigger that ran the script. Show

Lua

```
get_trigger_variable(idx)
```

Argument	Type	Example
idx	integer	1

Returns the trigger variable at idx as a Variant object.

Example:

```
-- Use with a TPC 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
```

HTTP

Not currently available.

JavaScript

Not currently available.

Resources

Use to locate resources in the controller's memory. Show

Lua

```
get_resource_path(filename)
```

Argument	Type	Example
filename	string	'settings.txt'

Returns a path to the resource filename.

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Content Target

Returns information about a Content Target in the project. Show

Lua

On a VLC

get_content_target(compositionNum)

On a VLC+

```
get_content_target(compositionNum, type)
```

- compositionNum is the usernumber of the composition to return
- type is the type of target within the composition to return (PRIMARY, SECONDARY, OVERLAY_1, OVERLAY_2)

Returns a Content Target object with the following properties:

master_intensity_level	Variant
rotation_offset (VLC+ only)	float
x_position_offset (VLC+ only)	float
y_position_offset (VLC+ only)	float

Example:

```
target = get_content_target(1)

current_level = target.master_intensity_level

target = get_content_target(1,PRIMARY)

current_angle = target.rotation_offset
```

HTTP

Not currently available.

JavaScript

Not currently available.

Config

Returns information about a controller's Configuration. Show

Lua

```
get_log_level()
```

Returns the current log level of the controller

```
get_syslog_enabled()
```

Returns true if Syslog is enabled, false otherwise

```
get_syslog_ip_address()
```

Returns the IP address of the Syslog server

```
get_ntp_enabled()
```

Returns true if NTP is enabled

```
get_ntp_ip_address()
```

Returns the IP address of the NTP server

Example:

HTTP

```
GET /api/config
```

Returns an object with the following properties:

Property	Return Type	Return Example
ip	string	"192.168.1.3"
subnet_mask	string	"255.255.255.0"
gateway	string	"192.168.1.1"
dhcp_enabled	boolean	true
name_server_1	string	"192.168.1.1"
name_server_2	string	"8.8.8.8"
http_port	integer	80
https_port	integer	443
year	integer	2019
month	integer (1-12)	4
day	integer (1-31)	25
hour	integer (0-23)	13
minute	integer (0-59)	21
second	integer (0-59)	46
watchdog_enabled	boolean	true
log_level	integer (1-5)	3
syslog_enabled	boolean	true
syslog_ip	string	"192.168.1.2"
ntp_enabled	boolean	true
ntp_ip	string	"192.168.1.1"

JavaScript

```
get_config(callback)
```

Returns the controller's configuration

The returned object has the same structure as in the HTTP call

API Actions

Below are the ways of changing properties or changing output on the controller. show

Start Timeline

Start a timeline in the project Show

Lua

```
get_timeline(timelineNum):start()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "start",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.start_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Start Scene

Start a scene in the project Show

Lua

```
get_scene(sceneNum):start()
```

Argument	Type	Example
sceneNum	integer	1

HTTP

```
POST /api/scene
{
   "action": "start",
   "num": sceneNum
}
```

Argument	Type	Example
sceneNum	integer	1

JavaScript

```
Query.start_scene({ "num": sceneNum}, callback)
```

Argument	Type	Example
sceneNum	integer	1

Release Timeline

Release a timeline in the project Show

Lua

```
get_timeline(timelineNum):release([fade])
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

HTTP

```
POST /api/timeline

{

  "action": "release",

  "num": timelineNum[,

  "fade": fade]

}
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

JavaScript

```
Query.release_timeline({ "num": timelineNum[, "fade": fade]}, callback)
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

Release Scene

Release a scene in the project Show

Lua

```
get_scene(sceneNum):release([fade])
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

HTTP

```
POST /api/scene

{

  "action": "release",

  "num": sceneNum[,

  "fade": fade]

}
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

JavaScript

```
Query.release_scene({ "num": sceneNum[, "fade": fade]}, callback)
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

Toggle Timeline

Toggle a timeline in the project (if it is running, stop it, and if it is not running, start it) Show

Lua

```
get_timeline(timelineNum):toggle([fade])
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

HTTP

```
POST /api/timeline
{
  "action": "toggle",
  "num": timelineNum[,
  "fade": fade]
}
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

JavaScript

```
Query.toggle_timeline({ "num": timelineNum[, "fade": fade]}, callback)
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

Toggle Scene

Toggle a scene in the project (if it is running, stop it, and if it is not running, start it) Show

Lua

```
get_scene(sceneNum):toggle([fade])
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

HTTP

```
POST /api/scene

{

  "action": "toggle",

  "num": sceneNum[,

  "fade": fade]

}
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

JavaScript

```
Query.toggle_scene({ "num": sceneNum[, "fade": fade]}, callback)
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

Pause Timeline

Pause a timeline in the project Show

Lua

```
get_timeline(timelineNum):pause()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline

{

  "action": "pause",

  "num": timelineNum

}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.pause_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Resume Timeline

Resume a timeline in the project Show

Lua

```
get_timeline(timelineNum):resume()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "resume",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.resume_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Pause All

Pause all timelines in the project Show

Lua

```
pause_all()
```

HTTP

```
POST /api/timeline
{
 "action": "pause"
}
```

JavaScript

```
Query.pause_all(callback)
```

Resume All

Resume all timelines in the project Show

Lua

```
resume_all()
```

HTTP

```
POST /api/timeline

{

 "action": "resume"

}
```

JavaScript

```
Query.resume_all(callback)
```

Release All

Release all timelines, scenes or timelines, scenes and overrides in the project Show

Lua

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

release_all_timelines([fade,] [group])

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

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

HTTP

```
POST /api/release_all

POST /api/timeline

POST /api/scene

{

  "action": "release"[, (not required for release all)

  "group": group][,

  "fade": fade]

}
```

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

JavaScript

```
release_all_timelines({["fade": fade]}, callback)

release_all_scenes({["fade": fade]}, callback)
```

```
release_all({ ["fade": fade,] ["group": group] }, callback)
```

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

Set Timeline Rate

Set the rate of a timeline in the project Show

Lua

```
get_timeline(timelineNum):set_rate(rate)
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

HTTP

```
POST /api/timeline

{

  "action": "set_rate",

  "num": timelineNum,

  "rate": rate

}
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

JavaScript

```
Query.set_timeline_rate({"num": timelineNum, "rate": rate }, callback)
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

Set Timeline Position

Set the position of a timeline in the project Show

Lua

```
get_timeline(timelineNum):set_position(position)
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

HTTP

```
POST /api/timeline

{

 "action": "set_position",

  "num": timelineNum,

  "position": position

}
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

JavaScript

```
Query.set_timeline_position({"num": timelineNum, "position": position }, call-
back)
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

Enqueue Trigger

Fire a trigger in the project Show

Lua

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

Argument	Type	Example
num - the trigger number to enqueue	integer	1
var... - 0 or more variables to pass to the trigger	comma separated variables	1,2,"string"

Example

```
enqueue_trigger(1,1,2,"string")
```

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

Enqueues a trigger without testing its conditions first (it will always be fired).

HTTP

```
POST /api/trigger

{
```

```
  "num": num[,
  "var": var...][,
  "conditions": test_conditions]
}
```

Argument	Type	Example
num	integer	1
var...	comma separated variables	1,2,"string"
test_conditions	boolean	true

- num - the trigger number to enqueue
- var... - 0 or more variables to pass to the trigger
- test_conditions - Should the conditions on the trigger be tested?

JavaScript

```
Query.fire_trigger({"num": num[, "var": var...][, "conditions": test_conditions]
}, callback)
```

Argument	Type	Example
num	integer	1
var...	comma separated variables	'1,2,"string"'
test_conditions	boolean	true

- var... - 0 or more variables to pass to the trigger. If passing multiple variables, they must be a single string surrounded by single quotes ('), string variables should be surrounded by double quotes (").
- num - the trigger number to enqueue
- test_conditions - Should the conditions on the trigger be tested?

Run Script

Run a script or parse into the command line on the controller Show

Lua

Not currently available.

HTTP

```
POST /api/cmdline
{
  "input": chunk,
}
```

NOTE: returns "Executed" if successful, or an error string if not

Argument	Type	Example
chunk - the script to parse or run	string	"tl = 1 get_timeline(tl):start()"

JavaScript

```
Query.run_command({ "input": chunk }, callback)
```

NOTE: returns "Executed" if successful, or an error string if not

Argument	Type	Example
chunk - the script to parse or run	string	"tl = 1 get_timeline(tl):start()"

Hardware Reset

Reset the controller (power reboot) Show

Lua

Not currently available.

HTTP

```
POST /api/reset
```

JavaScript

Not currently available.

Master Intensity

Master the intensity of a group or content target (applied as a multiplier to output levels) Show

Lua

Non-VLC

```
get_group(groupNum):set_master_intensity(level[, fade[, delay]])
```

VLC

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

VLC+

```
get_content_target(compositionNum, type):set_master_intensity(level, [fade,
[delay]])
```

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', 'overlay_1', 'overlay_2'
level	float or integer (0-255)	128 or 0.5
fade	float	2.0
delay	float	2.0

Example:

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

HTTP

Non-VLC

```
POST /api/group

{
```

```
 "action": "master_intensity",
    "num": groupNum,
    "level": level,
   ["fade": fade,]
   ["delay": delay]
}
```

VLC/VLC+

```
POST /api/content_target
{
"action": "master_intensity",
"level": level,
["fade": fade,]
["delay": delay,]
"type": type
}
```

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', 'overlay_1', 'overlay_2'
level	float or bounded integer	0.5 or "50:100"
fade	float	2.0
delay	float	2.0

JavaScript

Non-VLC

```
master_intensity({ "num": groupNum, "level": level, ["fade": fade,] ["delay":
delay] }, callback)
```

VLC/VLC+

master_content_target_intensity({ "type":type, "level": level, ["fade": fade,] ["delay": delay] }, callback)

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', 'overlay_1', 'overlay_2'
level	float or bounded integer	0.5 or "50:100"
fade	float	2.0
delay	float	2.0

Example:

```
Query.master_intensity({"num":1,"level":"50:100","fade":3) -- master group 1 to
```

```
50% (50/100 = 0.5) in 3 seconds)
```

NOTE: Group 0 will master the intensity of the 'All Fixtures' group. This can also be used on VLC family projects to master the intensity of the entire unit.

Set RGB

Set the Intensity, Red, Green, Blue levels for a fixture or group. Show

Lua

```
get_fixture_override(num)

get_group_override(num)

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

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

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

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

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

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

Argument	Type	Example
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated, "Damped, "Overshoot"

Example:

```
ov = get_fixture_override(1) -- Get fixture 1

ov:set_irgb(255, 255, 0, 0) -- Set the fixture to Red
```

HTTP

```
PUT /api/override

{

   "target": target,

   "num": num,

   ["intensity": intensity,]

   ["red": red,]
```

```
   ["green": green,]
   ["blue": blue,]
   ["temperature": temperature,]
   ["fade": fade,]
   ["path": path]
}
```

Argument	Type	Example
target	string (from options)	"group", "fixture"
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated, "Damped, "Overshoot"

Javascript

```
set_group_override({ "num": num, ["intensity": intensity,] ["red": red,]
["green": green,] ["blue": blue,] ["temperature": temperature,] ["fade": fade,]
["path": path] }, callback)
```

```
set_fixture_override({ "num": num, ["intensity": intensity,] ["red": red,]
["green": green,] ["blue": blue,] ["temperature": temperature,] ["fade": fade,]
["path": path] }, callback)
```

Argument	Type	Example
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated, "Damped, "Overshoot"

Example:

```
Query.set_fixture_override({ "num": 1, "intensity": 255, "red": 255, "green":
0, "blue": 0});
```

NOTE: Group 0 will set the levels of the 'All Fixtures' group

Clear RGB

Remove any overrides on fixtures or groups. Show

Lua

```
get_fixture_override(num)

get_group_override(num)
   :clear([fade])

clear_all_overrides([fade])
```

Argument	Type	Example
num - group or fixture	integer	1
fade	float	2.0

Example:

```
ov = get_fixture_override(1) -- Get fixture 1

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

HTTP

```
DELETE /api/override

{
   ["target": target,]
   ["num": objectNum,]
   ["fade": fade]
}
```

If num is not included, target is ignored and all overrides are cleared.

Argument	Type	Example
target	string (from options)	"group" or "fixture"
num - group or fixture	integer	1
fade	float	2.0

JavaScript

```
clear_group_overrides({ ["num" :num,] ["fade": fade] }, callback)

clear_fixture_overrides({ ["num" :num,] ["fade": fade] }, callback)

clear_overrides({ ["fade": fade] }, callback)
```

Argument	Type	Example
num	integer	1
fade	float	2.0

Example:

```
Query.clear_overrides({"fade":3})
```

Set Text Slot

Set the value of a text slot used in the project. Show

Lua

```
set_text_slot(name, value)
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

Example:

```
set_text_slot("myTextSlot", "Hello World!")
```

HTTP

```
PUT /api/text_slot
{
   "name": name,
   "value": value
}
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

JavaScript

```
set_text_slot({"name": name, "value": value}, callback)
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

Example:

```
Query.set_text_slot("name:"myTextSlot", "value":"Hello World!")
```

Set BPS Button LED

Set the effect and intensity on BPS button LEDS. Show

Lua

```
get_bps(num):set_led(button, effect, [intensity], [fade])
```

Argument	Type	Example
num	integer	1
button	integer	1
effect	OFF, ON, SLOW_FLASH, FAST_FLASH, DOUBLE_FLASH, BLINK, PULSE, SINGLE, RAMP_ON, RAMP_OFF	FAST_ FLASH

intensity	integer (1-255)	255
fade	float	0.0

Example:

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

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Touch Control Value

Set the value on a Touch Slider or Color Picker. Show

Lua

```
set_control_value(name, [index,] value[, emitChange])
```

Argument	Type	Example
name - control Key	string	"slider001"
index - axis of movement (slider has 1, colour picker has 3)	integer (1-3) (default 1)	1
value	integer (0-255)	128
emitChange	boolean (default false)	false

Example:

```
set_control_value("slider001", 1, 128) -- set slider001 to half and don't fire associated triggers
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Touch Control State

Set the state on a Touch control. Show

Lua

```
set_control_state(name, state)
```

Argument	Type	Example

name - control Key	string	"slider001"
state - the state name form Interface	string (from options in Interface)	"Green"

Example:

```
set_control_state("slider001", "Green") -- set slider001 to a state called "Green"
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Touch Control Caption

Set the caption on a Touch control. Show

Lua

```
set_control_caption(name, caption)
```

Argument	Type	Example
name - control Key	string	"button001"
caption - text to display	string	"On"

Example:

```
set_control_caption("button001", "On") -- set button001's caption to "On"
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Touch Page

Change the page on a Touch interface. Show

Lua

```
set_interface_page(number[, transition])
```

Argument	Type	Example
number	integer	4
transition	SNAP, PAN_LEFT, PAN_RIGHT	PAN_LEFT

Example:

```
set_interface_page(4) -- change the page on the TPC's interface to page 4
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Disable Page

Disable the touchscreen. Show

Lua

```
set_interface_enabled([enable])
```

Argument	Type	Example
enable	boolean (default true)	true

Example:

```
set_interface_enabled(false) -- disable the TPC's touch screen

set_interface_enabled() -- enable the TPC's touch screen
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Lock Touch Device

Lock the Touch Device (requires Lock code to be set within Interface). Show

Lua

```
set_interface_locked([lock])
```

Argument	Type	Example
lock	boolean (default true)	true

Example:

```
set_interface_enabled(false) -- disable the TPC's touch screen

set_interface_enabled() -- enable the TPC's touch screen
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Transition Content Target

Move or rotate a Content Target. Show

Lua

```
get_content_target(compositionNum, type)
  :transition_rotation([offset], [count], [period], [delay], [useShortestPath])
  :transition_x_position([offset], [count], [period], [delay])
  :transition_y_position([offset], [count], [period], [delay])
```

Argument	Type	Example
compositionNum	integer	1
type	PRIMARY, SECONDARY, OVERLAY_1, OVERLAY_2	PRIMARY
offset	integer	10
count	integer	1
period	integer	5
delay	integer	0
useShortestPath	boolean (default false)	false

Example:

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

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Transition Adjustment Target

Move or rotate a Adjustment Target. Show

Lua

```
get_adjustment(num)
  :transition_rotation([offset], [count], [period], [delay], [useShortestPath])
  :transition_x_position([offset], [count], [period], [delay])
  :transition_y_position([offset], [count], [period], [delay])
```

Argument	Type	Example
num	integer	1
offset	integer	10
count	integer	1
period	integer	5
delay	integer	0
useShortestPath	boolean (default false)	false

Example:

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

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Beacon Controller

Beacons the controller (flashes Status LEDs or screen). Show

Lua

Not currently available.

HTTP

```
POST /api/beacon
```

JavaScript

```
toggle_beacon(callback)
```

Example:

```
Query.toggle_beacon()
```

Output to Log

Writes a message to the controller's Log. Show

Lua

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

Argument	Type	Example
level	option (LOG_DEBUG, LOG_TERSE, LOG_NORMAL, LOG_EXTENDED, LOG_VERBOSE, LOG_CRITICAL, default LOG_NORMAL)	LOG_CRITICAL
message	string	"Some message to log."

Example:

log(LOG_CRITICAL, "This is a criticial message!") -- logs the message at Critical log level

log("This is a normal message.") -- logs the message at Normal log level.

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Send variables to Web Interface

Sends data to the web interface in a JSON Object. Show

Lua

```
push_to_web(name, value)
```

Argument	Type	Example
name	string	"myVar"
value	variable	"Some value"

Example:

```
myVar = 15

push_to_web("myVar", myVar) -- will push the object {"myVar": 15}
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Park a Channel

Parks an output channel at a specified level. Show

Lua

```
Universe:park(channel, value)
```

Argument	Type	Example
Universe	Universe object	get_dmx_universe(1)
channel	integer (1-512)	1
value	integer (0-255)	128

Example:

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

HTTP

```
POST /api/channel
{
    "universe": universeKey,
    "channels": channelList,
    "level": level
}
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"
level	integer (0-255)	128

JavaScript

```
park_channel({ "universe": universeKey, "channels": channelList, "level": level
}, callback)
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX)	"dmx:1"

	• protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	
channelList	comma separated list(1-512)	"1-3,5"
value	integer (0-255)	128

Example:

```
park_channel({ "universe": "dmx:1","channels": 1, "level":128}); // Park chan-
nel 1 of DMX Universe 1 at 128 (50%)
```

Unpark a Channel

Unparks an output channel. Show

Lua

```
Universe:unpark(channel)
```

Argument	Type	Example
Universe	Universe object	get_dmx_universe(1)
channel	integer (1-512)	1

Example:

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

HTTP

```
DELETE /api/channel

{

    "universe": universeKey,

    "channels": channelList

}
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"

JavaScript

```
park_channel({ "universe": universeKey, "channels": channelList }, callback)
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX)	"dmx:1"

- protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx
- remoteDeviceType can be rio08, rio44 or rio80

channelList	comma separated list(1-512)	"1-3,5"

Example:

```
park_channel({ "universe": "dmx:1","channels": 1}); //Unpark channel 1 of DMX Universe 1 (it will go back to normal output levels)
```

Disable an Output

Unparks an output channel. Show

Lua

```
disable_output(protocol)
```

```
enable_output(protocol)
```

Argument	Type	Example
protocol	option (DMX, PATHPORT, ARTNET, KINET, SACN, DVI, RIO_DMX)	DMX

Example:

disable_output(DMX) -- Disable the DMX output from the controller

HTTP

```
POST /api/output
```

```
{
    "protocol": protocol,
    "action": action
}
```

Argument	Type	Example
protocol	string ("dmx", "pathport", "art-net", "kinet", "sacn", "dvi", "rio-dmx")	"dmx"
action	string ("enable", "disable")	"disable"

JavaScript

```
disable_output({ "protocol": protocol }, callback)
```

```
enable_output({ "protocol": protocol }, callback)
```

Argument	Type	Example
protocol	string ("dmx", "pathport", "art-net", "kinet", "sacn", "dvi", "rio-dmx")	"dmx"

Example:

```
disbale_output({ "protocol": "dmx"}); // Disable the DMX output
enable_output({ "protocol": "art-net"}); // Enable the Art-Net Output
```

Set Timeline Source Bus

Set the time source for a timeline. Show

Lua

```
Timeline:set_default_source()
```

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

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

Argument	Type	Example
Timeline	Timeline Object	get_timeline(1)
timecodeBus	TCODE_1 ... TCODE_6	TCODE_1
audioBus	AUDIO_1 ... AUDIO_4	AUDIO_1
band	integer (0=volume)	0
channel	LEFT, RIGHT or COMBINED	LEFT
peak	boolean (default false)	false

Example:

get_timeline(1):set_timecode_source(TCODE_1) -- Set the timecode source of timeline 1 to timecode bus 1

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Enable Timecode Bus

Enables or disables a timecode bus. Show

Lua

```
set_timecode_bus_enabled(bus[, enable])
```

- bus is the timecode bus to enable or disable (TCODE_1 ... TCODE_6)
- enable determines whether the bus should be enabled or disabled (boolean, default true)

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Digital Output

Sets the output of a RIO to on or off. Show

Lua

```
get_rio(type, num):set_output(outputNum, state)
```

- type can be RIO44 or RIO08 (RIO80 has no outputs)
- num is the remote device number
- outputNum is the number of the output
- state is the state to set the output to and can be any of: 0, 1, true, false, ON, OFF

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Log Level

Sets the output of a RIO to on or off. Show

Lua

set_log_level(log_level)

Sets the log level of the controller to log_level

- log_level can be LOG_DEBUG, LOG_TERSE, LOG_NORMAL, LOG_EXTENDED, LOG_VERBOSE, LOG_CRITICAL or 0-5.

HTTP

Not currently available.

Use Edit Config

JavaScript

Not currently available.

Use Edit Config

Edit Config

Sets the output of a RIO to on or off. Show

Lua

Not currently available.

HTTP

```
POST /api/config

{

   "ip": ipAddress,
```

```
   "subnet_mask": subnetMask,
   "gateway": gateway,
   "dhcp_enabled": dhcpEnabled,
   "name_server_1": dnsServer1IPAddress,
   "name_server_2": dnsServer2IPAddress,
   "http_port": httpPort,
   "https_port": httpsPort,
   "year": year,
   "month": month,
   "day": day,
   "hour": hour,
   "minute": minute,
   "second": second,
   "watchdog_enabled": watchdogEnabled,
   "log_level": logLevel,
   "syslog_enabled": syslogEnabled,
   "syslog_ip": syslogIpAddress,
   "ntp_enabled": ntpEnabled,
   "ntp_ip": ntpIpAddress,
   "password": password
}
```

Argument	Type	Example
ipAddress	string	"192.168.1.2"
subnetMask	string	"255.255.255.0"
gateway	string	"192.168.1.1"
dhcpEnabled	boolean	true
dnsServer1IPAddress	string	"192.168.1.1"
dnsServer2IPAddress	string	"8.8.8.8"
httpPort	integer	80
httpsPort	integer	443
year	integer	2019
month	integer (0-11)	4
day	integer (1-31)	25
hour	integer (0-23)	13
minute	integer (0-59)	22
second	integer (0-59)	40
watchdogEnabled	boolean	true
logLevel	integer (0-5)	3

syslogEnabled	boolean	true
syslogIpAddress	string	"192.168.1.4"
ntpEnabled	boolean	true
ntpIpAddress	string	"192.168.1.1"
password	string	"myPassword"

If the response is 200 OK, the response body will be

```
{
   "restart": restart
}
```

`restart` is a boolean. If true, the controller will reset imminently in order to apply the changes

JavaScript

```
edit_config(params, callback)
```

Sends a request to change the controller's configuration

params is an object of the same format as in the HTTP request

The callback parameter will contain the same object as a response to the HTTP request

API Subscriptions

Subscriptions allow data to be pushed to the web interface whenever there is a change within the project. show

Subscribe Timeline Status

Subscribes to changes in the timeline status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_timeline_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	'running'
onstage	boolean	true
position	number (milliseconds)	5000

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_timeline_status(function(t){
```

```
        alert(t.num + ": " + t.state)
})
```

Subscribe Scene Status

Subscribes to changes in the scene status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_scene_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	'running'
onstage	boolean	true

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_scene_status(function(s){
        alert(s.num + ": " + s.state)
})
```

Subscribe Group Status

Subscribes to changes in group level, as set by the Master Intensity action (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_group_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
name	string	'Group 1'

level	integer (0-255)	128

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_group_status(function(g){
        alert(g.num + ": " + g.level)
})
```

Subscribe Remote Device Status

Subscribes to changes in Remote Device Online/Offline Status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_remote_device_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
type	string ('RIO 08', 'RIO 44', 'RIO 80','RIO D', 'RIO A', 'BPS')	'Group 1'
online	boolean	true
serial	string (of serial number)	'001001'

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_remote_device_status(function(r){
        alert(r.num + ": " + r.level)
})
```

Subscribe Beacon

Subscribes to Beacons (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_beacon(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
on	boolean	true

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_beacon(function(b){
        if (b.on){
                alert("Beacon Turned On")
        else {
                alert("Beacon Turned Off")
        }
})
```

Subscribe Lua

The receiver for the push_to_web() Lua function. Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_lua(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
key	as defined by push_to_web()	value

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_lua(function(l){
        key = Object.keys(l)[0]
        value = l.key
        alert(key + ": " + value)
})
```

API Objects

Below are the helper functions and objects in the project. show

Variant

A Lua object that allows a type and range to be associated with a variable. Show

Lua

See here.

HTTP

Not currently available.

JavaScript

Not currently available.

DateTime

A Lua object containing time data. Show

Lua

The DateTime object contains the following properties:

Property	Return Type	Return Example
.year	integer	2017
.month	integer (0-11)	5
.monthday	integer (0-30)	8
.weekday	integer(0-6)	1
.hour	integer(0-23)	13
.minute	integer (0-59)	21
.second	integer (0-59)	46
.utc_timestamp	integer	1494249706
.time_string	string	
.date_string	string	

HTTP

Not currently available.

JavaScript

Not currently available.

Printing an enum

Lua functions to convert integers returned from some functions as text. Show

Lua

```
digital_input_to_string()
```

```
button_state_to_string()
```

Examples:

```
log(digital_input_to_string(get_input(1)))

str = button_state_to_string(get_bps(1):get_state(1))
```

HTTP

Not currently available.

JavaScript

Not currently available.

API v3

The Pharos system includes multiple API options:

- Lua (used internally with Conditions and Actions)
- HTTP (used with external devices/software to communicate with a controller)
- JavaScript (used with Custom Web Interfaces)

These APIs have been unified to simplify their use as much as possible.

Glossary:

- Object - A collection of key value pairs e.g. "name" = "Controller 1" (syntax will differ between languages).
- String - A series of characters e.g."Th1s_is-4(string)"
- Number - Any whole or floating point(decimal) number e.g. 1,2,3,1.5,12.3456)
- Integer - A whole number
- Bounded integer - An integer with a range (e.g. 10:100 = 10%)
- Float/real/number - A decimal number (e.g. 3.2 or 1.0)
- JSON - (JavaScript Object Notation) a way of transferring information in the form of a JavaScript Object
- GET - A HTTP method to request data from a server
- POST - A HTTP method to request data in a more secure way
- PUT - A HTTP method to send data to a server
- Variant - See here
- [] - anything shown within square brackets is optional. The square brackets should be omitted if the optional section is used.
- callback - A function to run when the javascript function has been run, or a reply has been received.

HTTP Requests

Please note, when a HTTP POST request is sent, it must include a Content-Type header set to "application/json", otherwise it will be treated as invalid.

API Queries

Below are the ways of getting data from the controller. show

System

Returns data about the controller. show

Lua

The system namespace has the following properties:

Property	Return type	Return Example
.hardware_type	string	"lpc"
.channel_capacity	integer	512
.serial_number	string	"006321"
.memory_total	string	"12790Kb"
.memory_used	string	"24056Kb"
.memory_free	string	"103884Kb"
.storage_size	string	"1914MB"
.bootloader_version	string	"0.9.0"
.firmware_version	string	"2.14.0"

.reset_reason	string	"Software Reset"
.last_boot_time	DateTime object	
.ip_address	string	"192.168.1.3"
.subnet_mask	string	"255.255.255.0"
.default_gateway	string	"192.168.1.3"

Example:

```
capacity = system.channel_capacity

boot_time = system.last_boot_time.time_string
```

HTTP

```
GET /api/system
```

Returns an object with the following properties:

Property	Return type	Return Example
hardware_type	string	"LPC"
channel_capacity	integer	512
serial_number	string	"006321"
memory_total	string	"12790Kb"
memory_used	string	"24056Kb"
memory_free	string	"103884Kb"
storage_size	string	"1914MB"
bootloader_version	string	"0.9.0"
firmware_version	string	"2.14.0"
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"
default_gateway	string	"192.168.1.3"

JavaScript

```
get_system_info(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_system_info( function(system){

   var capacity = system.channel_capacity

}
```

Project

Returns data about the project. Show

Lua

get_current_project()

Returns an object with the following properties:

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

Example:

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

HTTP

```
GET /api/project
```

Returns an object with the following properties:

Property	Return type	Return Example
name	string	"Help Project"
author	string	"Pharos"
filename	string	"help_project_v1.pd2"
unique_id	string	"{6b48627a-1d5e-4b2f-81e2-481e092a6a79}"
upload_date	string	"2017-01-30T15:19:08"

JavaScript

```
get_project_info(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_project_info( function(project){
 var author = project.author
}
```

Replication

Returns data about the install replication. Show

Lua

get_current_replication()

Returns an object with the following properties:

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

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Location

Returns data about the install location. Show

Lua

get_location()

Returns an object with the following properties:

Property	Return type	Return Example
lat	float	51.512
long	float	-0.303

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Network 2

Returns data about the Network 2 (Protocol) Interface. Show

Lua

The protocol_interface namespace has the following properties:

Property	Return type	Return 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"

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Time

Returns data about the time stored in the controller. Show

Lua

The time namespace has the following functions which 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()

and the following properties

Property	Return Type	Return Example
is_dst	boolean	
gmt_offset	string	

Each function returns a DateTime object, with the following properties:

Property	Return Type	Return Example
.year	integer	2017
.month	integer (1-12)	5
.monthday	integer (1-31)	8
.weekday	integer (1-7)	1
.hour	integer (0-23)	13
.minute	integer (0-59)	21
.second	integer (0-59)	46
.utc_timestamp	integer	1494249706
.time_string	string	
.date_string	string	

Example:

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

HTTP

```
GET /api/time
```

Returns an object with the following properties:

Property	Return Type	Return Example
datetime	string	"01 Feb 2017 13:44:42"
local_time	integer (controller's local time in milliseconds)	1485956682
uptime	integer (time since last boot)	493347

JavaScript

```
get_current_time(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_current_time( function(time){
 var uptime = time.uptime
}
```

Timeline

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

Lua

```
get_timeline(timelineNum)
```

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

The returned object has the following properties

Property	Return Type	Return Example
name	string	"Timeline 1"
group	string ('A', 'B', 'C', 'D','')	'A'
length	integer	10000
source_bus	integer (equivalent to constants: DEFAULT, TCODE_1 ... TCODE_6, AUDIO_1 ... AUDIO_4)	1
timecode_format	string	"SMPTE30"
audio_band	integer (0 is equivalent to constant VOLUME)	0
audio_channel	integer (equivalent to constants: LEFT, RIGHT or COMBINED)	1
audio_peak	boolean	false
time_offset	integer	5000
state	integer (equivalent to constants: Timeline.NONE, Timeline.RUNNING, Timeline.PAUSED, Timeline.HOLDING_AT_END, Timeline.RELEASED)	1
onstage	boolean	true
position	integer	5000
priority	integer (equivalent to constants: HIGH_PRIORITY, ABOVE_NORMAL_PRIORITY, NORMAL_PRIORITY, BELOW_NORMAL_PRIORITY or LOW_PRIORITY)	0

Example:

```
tl = get_timeline(1)

name = tl.name

state = tl.state

if (tl.source_bus == TCODE_1) then

    -- do something

end
```

HTTP

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

- num can be used to filter which timelines are returned and can be a single number or a string representing the required timelines (e.g. "1,2,5-9")

Returns an object with the following properties:

timelines array of timeline objects

Each timeline object contains the following properties:

Property	Return Type	Return Example
num	integer	1
name	string	"Timeline 1"
group	string('A', 'B', 'C', 'D' or empty)	"A"
length	integer	10000
source_bus	string ('internal', 'timecode_1',...'timecode_6', 'audio_1',...'audio_4')	100
timecode_format	string	"SMPTE30"
audio_band	integer (0 is volume band)	1
audio_channel	string ('left', 'right', 'combined')	"combined"
audio_peak	boolean	false
time_offset	integer	5000
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	"running"
onstage	boolean	true
position	integer	10000
priority	string ('high', 'above_normal', 'normal', 'below_normal', 'low')	"normal"

JavaScript

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

- num can be used to filter which timelines are returned and is defined as a JSON object which can contain a single number or a string representing the required timelines (e.g. "1,2,5-9")

Returns an array of timelines in the same way as the HTTP call

Example:

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

Scene

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

Lua

```
get_scene(sceneNum)
```

Returns a single Scene object for the Scene with user number SceneNum.

The returned object has the following properties

Property	Return Type	Return Example
name	string	"Scene 1"
state	string ('none', 'started')	"none"
onstage	boolean	false

Example:

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

HTTP

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

num can be used to filter which scenes are returned and can be a single number or array

Returns an object with the following properties:

scenes array of scene objects

Each scene object contains the following properties:

Property	Return Type	Return Example
name	string	"Scene 1"
num	integer	1
state	string ('none', 'started')	"none"
onstage	boolean	false

JavaScript

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

filter may contain a num property which is used to filter which scenes are returned

Returns an array of scenes in the same way as the HTTP call

Example:

```
Query.get_scene_info( function(s){
 var name = s.scenes[0].name //name of the first timeline
}, {"num":"1-4"})
```

Group

Returns data about the groups in the project. Show

Lua

```
get_group(groupNum)
```

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

The returned object has the following properties:

Property	Return Type	Return Example
name	string	"Group 1"
master_intensity_level	Variant	

Example:

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

HTTP

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

num can be used to filter which groups are returned and can be a single number or array

Returns an object with the following properties:

groups array of group objects

Each group object contains the following properties:

Property	Return Type	Return Example
num	integer (only included for user created groups)	1
name	string	"Group 1"
level	integer (0-100)	100

JavaScript

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

filter may contain a num property which is used to filter which groups are returned

Returns an array of groups in the same way as the HTTP call

Example:

```
Query.get_group_info( function(g){
 var name = g.groups[0].name //name of the first timeline
}, {"num":"1-4"})
```

NOTE: Group 0 will return data about the 'All Fixtures' group

Controller

Returns data about the controller. Show

Lua

```
get_current_controller()
```

Returns an object for the containing the following properties:

Property	Return Type	Return Example
number	integer	1
name	string	"Controller 1"

Example:

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

```
is_controller_online(controllerNumber)
```

Returns true if the controller with user number controllerNum has been discovered, and false otherwise

Example:

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

HTTP

```
GET /api/controller
```

Returns an object with the following properties:

controllers array of controller objects (one for each controller in the project)

Each controller object contains the following properties:

Property	Return Type	Return Example
num	number	1
type	string	"LPC"
name	string	"Controller 1"
serial	string	"009060"
ip_address	string (if the controller is discovered)/empty (if the controller is not discovered or is the queried controller)	"192.168.1.3" or ""
online	boolean	true

JavaScript

```
get_controller_info(callback)
```

Returns an array of controllers in the same way as the HTTP call

Example:

```
Query.get_controller_info( function(controller){
 var name = controller[0].name // name of the first controller
}
```

Temperature

Returns data about the controller's temperature. Show

Lua

```
get_temperature()
```

Returns an object with the following properties

Property	Return Type	Return Example
sys_temp	number (only for LPC X and VLC/VLC+)	40
core_temp	number (only for LPC X and VLC/VLC+)	44
ambient_temp	number (only for TPC, LPC X rev 1)	36.900001525878906
cc_temp	number (only for LPC X rev 2 and VLC/VLC+)	44
gpu_temp	number (only for VLC/VLC+)	44

Example:

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

HTTP

```
GET /api/temperature
```

Returns an object with the following properties:

Property	Return Type	Return Example
sys_temp	number (only for LPC X and VLC/VLC+)	40
core1_temp	number (only for LPC X and VLC/VLC+)	44
core2_temp	number (only for LPC X rev 1	44
ambient_temp	number (only for TPC, LPC X rev 1)	36.900001525878906
cc_temp	number (only for LPC X rev 2 and VLC/VLC+)	44
gpu_temp	number (only for VLC/VLC+)	44

JavaScript

```
get_temperature(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_temperature( function(temp){
 var ambient = temp.ambient_temp // ambient temperature of the controller
}
```

Remote Device

Returns data about the Remote Device/s in the project. Show

Lua

```
get_rio(type, num):get_input(inputNum)
```

- type can be RIO80, RIO44 or RIO08
- num is the remote device number
- inputNum is the number of the input

Returns a boolean if the input is set to Digital or Contact Closure, or an integer if the input is set to Analog.

Example:

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

```
get_bps(num):get_state(buttonNum)
```

- num is the BPS number
- buttonNum is the number of the button

Returns the state of the button, which can be RELEASED, PRESSED, HELD or REPEAT

Example:

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

HTTP

```
GET /api/remote_device
```

Returns an array of all remote devices in the project.

The returned object has the following structure

remote_devices array of Remote Device objects

Each Remote Device object contains the following properties:

Property	Return Type	Return Example
num	integer	1
type	string ('RIO08', 'RIO44', 'RIO80', 'BPS', 'BPI', 'RIO A', 'RIO D')	"RIO 44"
serial	array (all discovered serial number for the address	["001234"]

	and type)	
outputs	array (of Output objects, only present for RIO44 and RIO08 that are on the queried controller)	[{"output":1,"value":true},{"output":2,"value":true},{"output":3,"value":true},{"output":4,"value":true}]
inputs	array (of Input objects, only present for RIO44 and RIO80 that are 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 (if the remote device is detected as being online)	true

The Output object has the following properties:

Property	Return Type	Return Example
output	integer	1
state	boolean (true means the output is on, false means it is off)	false

The Input object has the following properties:

Property	Return Type	Return Example
input	integer	1
type	string ('Analog', 'Digital', 'Contact Closure')	'Digital'
value	integer or bool (depends on type)	true

JavaScript

```
get_remote_device_info(callback)
```

Returns an array of all remote devices in the project with the same properties as in the HTTP call.

Example:

```
Query.get_remote_device_info( function(remote){
 var type = remote[0].type // type of the first remote device
}
```

Text Slots

Returns data about the text slots in the project. Show

Lua

```
get_text_slot(slotName)
```

- slotName is the name of the text slot

Returns the value of slotName

Example:

```
log(get_text_slot("test_slot"))
```

HTTP

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

slotNames can be used to filter which text slots are returned and can be a single name or array

The returned object has the following structure

text_slots	array of Text Slot objects

Each Text Slot object contains the following properties:

Property	Return Type	Return Example
name	string	"text"
value	string	"example"

JavaScript

```
get_text_slot(callback[, filter])
```

filter may contain a names property which is used to filter which text slot values are returned

Returns an array of all text slots in the project with the same properties as in the HTTP call.

Example:

```
Query.get_text_slot( function(text){
 var value = text[0].value // value of the first text slot
}, {names: "test_slot1", "test_slot2"})
```

Get Log

Returns the log from the queried controller. Show

Lua

Not currently available.

HTTP

```
GET /api/log
```

The returned object has the following structure

Property	Return Type
log	string (containing the whole log of the controller)

JavaScript

Not currently available.

Protocol

Returns the protocols and universes being output from the queried controller. Show

Lua

Not currently available.

HTTP

```
GET /api/protocol
```

Returns all the universes on the queried controller

The returned object has the following structure

outputs array of Protocol objects

Each Protocol object has the following properties:

Property	Return Type	Return Example
type	integer	1
name	string	"DMX"
disabled	boolean (whether the output has been disabled via an Action)	true
universes	array (Universe objects)	{"key":{"index":1},"name":"1"},{"key":{"index":2},"name":"2"}
dmx_proxy	DMX Proxy Object (where appropriate)	{ "ip_address": "192.168.1.17", "name": "Controller 1" }

Each Universe object has the following properties:

Property	Return Type	Return Example
name	string	"1"
key	Universe Key object	{"index":1}

Each DMX Proxy object has the following properties:

Property	Return Type	Return Example
name	string (name of the controller that is outputting this universe)	'Controller 1'
ip_address	string IP Address of the controller outputting this universe	'192.168.1.23'

The properties of the Universe Key object depends upon the type:

For DMX, Pathport, sACN and Art-Net:

Property	Return Type	Return Example
index	integer	1

For KiNET:

Property	Return Type	Return Example
kinet_port	integer	1
kinet_power_supply_num	integer	1

JavaScript

```
get_protocols(callback)
```

Returns all the universes on the queried controller

The returned object has the following structure

outputs array of Protocol objects

Each Protocol object has the following properties:

Property	Return Type	Return Example
type	integer	1
name	string	"DMX"
disabled	boolean (whether the output has been disabled via an Action)	true
universes	array (Universe objects)	[{"key":{"index":1},"name":"1"},{"key":{"index":2},"name":"2"}]
dmx_proxy	DMX Proxy Object (where appropriate)	

Each Universe object has the following properties:

Property	Return Type	Return Example
name	string	"1"
key	Universe Key object	{"index":1}

Each DMX Proxy object has the following properties:

Property	Return Type	Return Example
name	string (name of the controller that is outputting this universe)	'Controller 1'
ip_address	string IP Address of the controller outputting this universe	'192.168.1.23'

The properties of the Universe Key object depends upon the type:

For DMX, Pathport, sACN and Art-Net:

Property	Return Type	Return Example
index	integer	1

For KiNET:

Property	Return Type	Return Example
kinet_port	integer	1
kinet_power_supply_num	integer	1

Output

Returns the levels being output from the queried controller. Show

Lua

```
get_dmx_universe(idx)
```

```
get_artnet_universe(idx)
```

```
get_pathport_universe(idx)
```

```
get_sacn_universe(idx)
```

- idx is the required universe number

```
get_kinet_universe(power_supply_num, port_num)
```

- power_supply_num is the power supply to return the output from
- port_num is the port to return the output from

These all return a Universe object, which has the following function

```
get_channel_value(chnl)
```

- chnl is the channel to get the value from

Example:

```
uni = get_dmx_universe(1) -- get DMX Universe 1

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

HTTP

```
GET /api/output?universe=universeKey
```

- universeKey is a string in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX.
- protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx
- remoteDeviceType can be rio08, rio44 or rio80

Example:

GET /api/output?universe=dmx:1

GET /api/output?universe=rio-dmx:rio44:1

The returned object has the following structure

Property	Return Type	Return Example
channels	array (of channel values)	[0,0,0,0,0,0,0,0,0,255,255,255...255,0,255]
disabled	boolean (whether the output has been disabled via an Action)	true
proxied_tpc_name	string (only if controller is LPC, universe is DMX 2, DMX Proxy has been enabled and the TPC is offline)	'Controller 2'

JavaScript

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

Argument	Type	Example
universekey	string or an object containing protocol and either index, kinet_power_supply_num and kinet_port or remote_device_type and remote_device_num	dmx:1

- universeKey can be either a string, or an object containing protocol and either index, kinet_power_supply_num and kinet_port or remote_device_type and remote_device_num as received from get_protocols

Returns an object with the same structure as in the HTTP call

Input

Returns the inputs on the queried controller. Show

Lua

```
get_input(idx)
```

Argument	Type	Example
idx	integer	1

Returns the value of the controllers input as a boolean or integer

Example:

```
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)
```

`get_dmx_input(chnl)`

Argument	Type	Example
chnl	integer	1

- chnl is the required channel number

Returns the value of the DMX input at channel chnl as an integer

HTTP

`GET /api/input`

The returned object has the following structure

Property	Return Type	Return Example
gpio	array (of Input objects, on LPC or TPC+EXT)	[{"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},{"input":5,"type":"Contact Closure","value":true},{"input":6,"-type":"Contact Closure","value":true},{"input":7,"type":"Contact Closure","value":true},{"input":8,"type":"Contact Closure","value":true}]
dmxIn	object (DMX Input object, if DMX Input is configured)	[0,0,0,0,0,0,0,0,0,255,255,255...255,0,255]

The Input object has the following properties:

Property	Return Type	Return Example
input	integer	1
type	string ('Analog', 'Digital', 'Contact Closure')	"Contact Closure"
value	integer or bool (depends on type)	true

The DMX Input object has the following properties:

Property	Return Type	Return Example

error	string (if DMX Input is configured but no DMX is received)	"No DMX received"
dmxInFrame	array (of channel values)	[0,0,0,0,0,0,0,0,255,255,255...255,0,255]

JavaScript

Not currently available.

Trigger

Returns the triggers in the project. Show

Lua

Not currently available.

HTTP

```
GET /api/trigger
```

The returned object has the following structure

triggers array (of Trigger objects)

The Trigger object has the following properties:

Property	Return Type	Return Example
type	string	"Startup"
num	integer	1
name	string	"Startup"
trigger_text	string	"At startup"
conditions	array (of Condition objects)	[{"text":"Before 12:00:00 every day"}]
actions	array (of Action objects	[{"text":"Start Timeline 1"}]

The Condition and Action objects have the following properties:

Property	Return Type	Return Example
text	string	"Start Timeline 1"

JavaScript

Not currently available.

Lua Variable

Returns the current value of the specified Lua variable. Show

Lua

Not currently available.

HTTP

```
GET /api/lua?variables=luaVariables
```

Argument	Type	Example
luaVariables	string or comma separated list	'myVar' or 'myVar, myVar2, myVar3'

Returns an object containing all the Lua variables requested and their values.

JavaScript

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

Argument	Type	Example
luaVariables	string or array	'myVar' or 'myVar, myVar2, myVar3'

Returns an object containing all the Lua variables requested and their values.

Example:

```
Query.get_lua_variables("myVar", function(lua){
 var value = lua.myVar
}
```

Trigger Variable

Returns the value of a variables from the trigger that ran the script. Show

Lua

```
get_trigger_variable(idx)
```

Argument	Type	Example
idx	integer	1

Returns the trigger variable at idx as a Variant object.

Example:

```
-- Use with a TPC 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
```

HTTP

Not currently available.

JavaScript

Not currently available.

Resources

Use to locate resources in the controller's memory. Show

Lua

```
get_resource_path(filename)
```

Argument	Type	Example
filename	string	'settings.txt'

Returns a path to the resource filename.

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Content Target

Returns information about a Content Target in the project. Show

Lua

On a VLC

get_content_target(compositionNum)

On a VLC+

```
get_content_target(compositionNum, type)
```

- compositionNum is the usernumber of the composition to return
- type is the type of target within the composition to return (PRIMARY, SECONDARY, OVERLAY_1, OVERLAY_2)

Returns a Content Target object with the following properties:

master_intensity_level	Variant
rotation_offset (VLC+ only)	float
x_position_offset (VLC+ only)	float
y_position_offset (VLC+ only)	float

Example:

```
target = get_content_target(1)

current_level = target.master_intensity_level

target = get_content_target(1,PRIMARY)

current_angle = target.rotation_offset
```

HTTP

Not currently available.

JavaScript

Not currently available.

API Actions

Below are the ways of changing properties or changing output on the controller. show

Start Timeline

Start a timeline in the project Show

Lua

```
get_timeline(timelineNum):start()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "start",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.start_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Start Scene

Start a scene in the project Show

Lua

```
get_scene(sceneNum):start()
```

Argument	Type	Example
sceneNum	integer	1

HTTP

```
POST /api/scene
{
   "action": "start",
   "num": sceneNum
}
```

Argument	Type	Example
sceneNum	integer	1

JavaScript

```
Query.start_scene({ "num": sceneNum}, callback)
```

Argument	Type	Example
sceneNum	integer	1

Release Timeline

Release a timeline in the project Show

Lua

```
get_timeline(timelineNum):release([fade])
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

HTTP

```
POST /api/timeline
{
  "action": "release",
  "num": timelineNum[,
  "fade": fade]
}
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

JavaScript

```
Query.release_timeline({ "num": timelineNum[, "fade": fade]}, callback)
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

Release Scene

Release a scene in the project Show

Lua

```
get_scene(sceneNum):release([fade])
```

Argument	Type	Example
sceneNum	integer	1

fade	float	2.0

HTTP

```
POST /api/scene
{
  "action": "release",
  "num": sceneNum[,
  "fade": fade]
}
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

JavaScript

```
Query.release_scene({ "num": sceneNum[, "fade": fade]}, callback)
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

Toggle Timeline

Toggle a timeline in the project (if it is running, stop it, and if it is not running, start it) Show

Lua

```
get_timeline(timelineNum):toggle([fade])
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

HTTP

```
POST /api/timeline
{
  "action": "toggle",
  "num": timelineNum[,
  "fade": fade]
}
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

JavaScript

```
Query.toggle_timeline({ "num": timelineNum[, "fade": fade]}, callback)
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

Toggle Scene

Toggle a scene in the project (if it is running, stop it, and if it is not running, start it) Show

Lua

```
get_scene(sceneNum):toggle([fade])
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

HTTP

```
POST /api/scene
{
  "action": "toggle",
  "num": sceneNum[,
  "fade": fade]
}
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

JavaScript

```
Query.toggle_scene({ "num": sceneNum[, "fade": fade]}, callback)
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

Pause Timeline

Pause a timeline in the project Show

Lua

```
get_timeline(timelineNum):pause()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "pause",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.pause_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Resume Timeline

Resume a timeline in the project Show

Lua

```
get_timeline(timelineNum):resume()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "resume",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.resume_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Pause All

Pause all timelines in the project Show

Lua

```
pause_all()
```

HTTP

```
POST /api/timeline

{

 "action": "pause"

}
```

JavaScript

```
Query.pause_all(callback)
```

Resume All

Resume all timelines in the project Show

Lua

```
resume_all()
```

HTTP

```
POST /api/timeline

{

 "action": "resume"

}
```

JavaScript

```
Query.resume_all(callback)
```

Release All

Release all timelines, scenes or timelines, scenes and overrides in the project Show

Lua

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

release_all_timelines([fade,] [group])

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

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

HTTP

```
POST /api/release_all

POST /api/timeline

POST /api/scene
```

```
{
  "action": "release"[, (not required for release all)
  "group": group][,
  "fade": fade]
}
```

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

JavaScript

```
release_all_timelines({["fade": fade]}, callback)
release_all_scenes({["fade": fade]}, callback)
release_all({ ["fade": fade,] ["group": group] }, callback)
```

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

Set Timeline Rate

Set the rate of a timeline in the project Show

Lua

```
get_timeline(timelineNum):set_rate(rate)
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

HTTP

```
POST /api/timeline
{
  "action": "set_rate",
  "num": timelineNum,
  "rate": rate
}
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

JavaScript

```
Query.set_timeline_rate({"num": timelineNum, "rate": rate }, callback)
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

Set Timeline Position

Set the position of a timeline in the project Show

Lua

```
get_timeline(timelineNum):set_position(position)
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

HTTP

```
POST /api/timeline

{

 "action": "set_position",

  "num": timelineNum,

  "position": position

}
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

JavaScript

```
Query.set_timeline_position({"num": timelineNum, "position": position }, call-
back)
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

Enqueue Trigger

Fire a trigger in the project Show

Lua

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

Argument	Type	Example
num - the trigger number to enqueue	integer	1
var... - 0 or more variables to pass	comma separated variables	1,2,"string"

to the trigger

Example

```
enqueue_trigger(1,1,2,"string")
```

HTTP

```
POST /api/trigger

{

  "num": num[,

  "var": var...][,

  "conditions": test_conditions]

}
```

Argument	Type	Example
num	integer	1
var...	comma separated variables	1,2,"string"
test_conditions	boolean	true

- num - the trigger number to enqueue
- var... - 0 or more variables to pass to the trigger
- test_conditions - Should the conditions on the trigger be tested?

JavaScript

```
Query.fire_trigger({"num": num[, "var": var...][, "conditions": test_conditions]
}, callback)
```

Argument	Type	Example
num	integer	1
var...	comma separated variables	'1,2,"string"'
test_conditions	boolean	true

- num - the trigger number to enqueue
- var... - 0 or more variables to pass to the trigger. If passing multiple variables, they must be a single string surrounded by single quotes ('), string variables should be surrounded by double quotes (").
- test_conditions - Should the conditions on the trigger be tested?

Run Script

Run a script or parse into the command line on the controller Show

Lua

Not currently available.

HTTP

```
POST /api/cmdline

{

  "input": chunk,

}
```

NOTE: returns "Executed" if successful, or an error string if not

Argument	Type	Example
chunk - the script to parse or run	string	"tl = 1 get_timeline(tl):start()"

JavaScript

```
Query.run_command({ "input": chunk }, callback)
```

NOTE: returns "Executed" if successful, or an error string if not

Argument	Type	Example
chunk - the script to parse or run	string	"tl = 1 get_timeline(tl):start()"

Hardware Reset

Reset the controller (power reboot) Show

Lua

Not currently available.

HTTP

```
POST /api/reset
```

JavaScript

Not currently available.

Master Intensity

Master the intensity of a group or content target (applied as a multiplier to output levels) Show

Lua

Non-VLC

```
get_group(groupNum):set_master_intensity(level[, fade[, delay]])
```

VLC

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

VLC+

```
get_content_target(compositionNum, type):set_master_intensity(level, [fade,
[delay]])
```

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	PRIMARY, SECONDARY, TARGET_3 ... TARGET_8
level	float or integer (0-255)	128 or 0.5
fade	float	2.0
delay	float	2.0

Example:

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

HTTP

Non-VLC

```
POST /api/group

{

 "action": "master_intensity",

    "num": groupNum,

    "level": level,

   ["fade": fade,]

   ["delay": delay]

}
```

VLC/VLC+

```
POST /api/content_target

{

"action": "master_intensity",

"level": level,

["fade": fade,]

["delay": delay,]

"type": type

}
```

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', target_3' ... 'target_8'
level	float or bounded integer	0.5 or "50:100"
fade	float	2.0
delay	float	2.0

JavaScript

Non-VLC

```
master_intensity({ "num": groupNum, "level": level, ["fade": fade,] ["delay":
delay] }, callback)
```

VLC/VLC+

master_content_target_intensity({ "type":type, "level": level, ["fade": fade,] ["delay": delay] }, callback)

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', target_3' ... 'target_8'
level	float or bounded integer	0.5 or "50:100"
fade	float	2.0
delay	float	2.0

Example:

```
Query.master_intensity({"num":1,"level":"50:100","fade":3) -- master group 1 to 50% (50/100 = 0.5) in 3 seconds)
```

NOTE: Group 0 will master the intensity of the 'All Fixtures' group

Set RGB

Set the Intensity, Red, Green, Blue levels for a fixture or group. Show

Lua

```
get_fixture_override(num)

get_group_override(num)

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

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

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

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

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

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

Argument	Type	Example
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated, "Damped, "Overshoot"

Example:

```
ov = get_fixture_override(1) -- Get fixture 1

ov:set_irgb(255, 255, 0, 0) -- Set the fixture to Red
```

HTTP

```
PUT /api/override

{

   "target": target,

   "num": num,

   ["intensity": intensity,]

   ["red": red,]

   ["green": green,]

   ["blue": blue,]

   ["temperature": temperature,]

   ["fade": fade,]

   ["path": path]

}
```

Argument	Type	Example
target	string (from options)	"group", "fixture"
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated, "Damped, "Overshoot"

Javascript

```
set_group_override({ "num": num, ["intensity": intensity,] ["red": red,]
["green": green,] ["blue": blue,] ["temperature": temperature,] ["fade": fade,]
["path": path] }, callback)

set_fixture_override({ "num": num, ["intensity": intensity,] ["red": red,]
["green": green,] ["blue": blue,] ["temperature": temperature,] ["fade": fade,]
["path": path] }, callback)
```

Argument	Type	Example
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255

temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated, "Damped, "Overshoot"

Example:

```
Query.set_fixture_override({ "num": 1, "intensity": 255, "red": 255, "green": 0, "blue": 0});
```

NOTE: Group 0 will set the levels of the 'All Fixtures' group

Clear RGB

Remove any overrides on fixtures or groups. Show

Lua

```
get_fixture_override(num)

get_group_override(num)
   :clear([fade])

clear_all_overrides([fade])
```

Argument	Type	Example
num - group or fixture	integer	1
fade	float	2.0

Example:

```
ov = get_fixture_override(1) -- Get fixture 1

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

HTTP

```
DELETE /api/override

{
   ["target": target,]
   ["num": objectNum,]
   ["fade": fade]
}
```

If num is not included, target is ignored and all overrides are cleared.

Argument	Type	Example
target	string (from options)	"group" or "fixture"
num - group or fixture	integer	1
fade	float	2.0

JavaScript

```
clear_group_overrides({ ["num" :num,] ["fade": fade] }, callback)
```

```
clear_fixture_overrides({ ["num" :num,] ["fade": fade] }, callback)
clear_overrides({ ["fade": fade] }, callback)
```

Argument	Type	Example
num	integer	1
fade	float	2.0

Example:

```
Query.clear_overrides({"fade":3})
```

Set Text Slot

Set the value of a text slot used in the project. Show

Lua

```
set_text_slot(name, value)
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

Example:

```
set_text_slot("myTextSlot", "Hello World!")
```

HTTP

```
PUT /api/text_slot
{
   "name": name,
   "value": value
}
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

JavaScript

```
set_text_slot({"name": name, "value": value}, callback)
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

Example:

```
Query.set_text_slot("name:"myTextSlot", "value":"Hello World!")
```

Set BPS Button LED

Set the effect and intensity on BPS button LEDS. Show

Lua

```
get_bps(num):set_led(button, effect, [intensity], [fade])
```

Argument	Type	Example
num	integer	1
button	integer	1
effect	OFF, ON, SLOW_FLASH, FAST_FLASH, DOUBLE_FLASH, BLINK, PULSE, SINGLE, RAMP_ON, RAMP_OFF	FAST_FLASH
intensity	integer (1-255)	255
fade	float	0.0

Example:

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

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Touch Control Value

Set the value on a Touch Slider or Color Picker. Show

Lua

```
set_control_value(name, [index,] value[, emitChange])
```

Argument	Type	Example
name - control Key	string	"slider001"
index - axis of movement (slider has 1, colour picker has 3)	integer (1-3) (default 1)	1
value	integer (0-255)	128
emitChange	boolean (default false)	false

Example:

```
set_control_value("slider001", 1, 128) -- set slider001 to half and don't fire associated triggers
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Touch Control State

Set the state on a Touch control. Show

Lua

```
set_control_state(name, state)
```

Argument	Type	Example
name - control Key	string	"slider001"
state - the state name form Interface	string (from options in Interface)	"Green"

Example:

```
set_control_state("slider001", "Green") -- set slider001 to a state called "Green"
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Touch Control Caption

Set the caption on a Touch control. Show

Lua

```
set_control_caption(name, caption)
```

Argument	Type	Example
name - control Key	string	"button001"
caption - text to display	string	"On"

Example:

```
set_control_caption("button001", "On") -- set button001's caption to "On"
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set Touch Page

Change the page on a Touch interface. Show

Lua

```
set_interface_page(number[, transition])
```

Argument	Type	Example
number	integer	4
transition	SNAP, PAN_LEFT, PAN_RIGHT	PAN_LEFT

Example:

```
set_interface_page(4) -- change the page on the TPC's interface to page 4
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Disable Page

Disable the touchscreen. Show

Lua

```
set_interface_enabled([enable])
```

Argument	Type	Example
enable	boolean (default true)	true

Example:

```
set_interface_enabled(false) -- disable the TPC's touch screen

set_interface_enabled() -- enable the TPC's touch screen
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Lock Touch Device

Lock the Touch Device (requires Lock code to be set within Interface). Show

Lua

```
set_interface_locked([lock])
```

Argument	Type	Example
lock	boolean (default true)	true

Example:

```
set_interface_enabled(false) -- disable the TPC's touch screen

set_interface_enabled() -- enable the TPC's touch screen
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Transition Content Target

Move or rotate a Content Target. Show

Lua

```
get_content_target(compositionNum, type)
  :transition_rotation([offset], [count], [period], [delay], [useShortestPath])
  :transition_x_position([offset], [count], [period], [delay])
  :transition_y_position([offset], [count], [period], [delay])
```

Argument	Type	Example
compositionNum	integer	1
type	PRIMARY, SECONDARY, TARGET_3 ... TARGET_8	PRIMARY
offset	integer	10
count	integer	1
period	integer	5
delay	integer	0
useShortestPath	boolean (default false)	false

Example:

```
tar = get_composition_target(1,PRIMARY)

tar:transition_x_position(10,1,5) -- Move 10 pixels right in 5 seconds

tar:transition_y_position(10,1,5) -- Move 10 pixels down in 5 seconds

tar:transition_rotation(90,1,5) -- Rotate by 90 degrees in 5 seconds
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Transition Adjustment Target

Move or rotate a Adjustment Target. Show

Lua

```
get_adjustment(num)
  :transition_rotation([offset], [count], [period], [delay], [useShortestPath])
  :transition_x_position([offset], [count], [period], [delay])
  :transition_y_position([offset], [count], [period], [delay])
```

Argument	Type	Example
num	integer	1
offset	integer	10
count	integer	1
period	integer	5
delay	integer	0
useShortestPath	boolean (default false)	false

Example:

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

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Beacon Controller

Beacons the controller (flashes Status LEDs or screen). Show

Lua

Not currently available.

HTTP

```
POST /api/beacon
```

JavaScript

```
toggle_beacon(callback)
```

Example:

```
Query.toggle_beacon()
```

Output to Log

Writes a message to the controller's Log. Show

Lua

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

Argument	Type	Example
level	option (LOG_DEBUG, LOG_TERSE, LOG_NORMAL, LOG_EXTENDED, LOG_VERBOSE, LOG_CRITICAL, default LOG_NORMAL)	LOG_CRITICAL
message	string	"Some message to log."

Example:

log(LOG_CRITICAL, "This is a criticial message!") -- logs the message at Critical log level

log("This is a normal message.") -- logs the message at Normal log level.

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Send variables to Web Interface

Sends data to the web interface in a JSON Object. Show

Lua

```
push_to_web(name, value)
```

Argument	Type	Example
name	string	"myVar"
value	variable	"Some value"

Example:

```
myVar = 15

push_to_web("myVar", myVar) -- will push the object {"myVar": 15}
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Park a Channel

Parks an output channel at a specified level. Show

Lua

```
Universe:park(channel, value)
```

Argument	Type	Example
Universe	Universe object	get_dmx_universe(1)
channel	integer (1-512)	1
value	integer (0-255)	128

Example:

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

HTTP

```
POST /api/channel

{

    "universe": universeKey,

    "channels": channelList,

    "level": level

}
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"
level	integer (0-255)	128

JavaScript

```
park_channel({ "universe": universeKey, "channels": channelList, "level": level
}, callback)
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"
value	integer (0-255)	128

Example:

```
park_channel({ "universe": "dmx:1","channels": 1, "level":128}); // Park chan-
nel 1 of DMX Universe 1 at 128 (50%)
```

Unpark a Channel

Unparks an output channel. Show

Lua

```
Universe:unpark(channel)
```

Argument	Type	Example
Universe	Universe object	get_dmx_universe(1)
channel	integer (1-512)	1

Example:

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

HTTP

```
DELETE /api/channel

{

    "universe": universeKey,

    "channels": channelList

}
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"

JavaScript

```
park_channel({ "universe": universeKey, "channels": channelList }, callback)
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"

Example:

```
park_channel({ "universe": "dmx:1","channels": 1}); //Unpark channel 1 of DMX Universe 1 (it will go back to normal output levels)
```

Disable an Output

Unparks an output channel. Show

Lua

```
disable_output(protocol)
```

```
enable_output(protocol)
```

Argument	Type	Example
protocol	option (DMX, PATHPORT, ARTNET, KINET, SACN, DVI, RIO_DMX)	DMX

Example:

disable_output(DMX) -- Disable the DMX output from the controller

HTTP

```
POST /api/output
```

```
{
    "protocol": protocol,
    "action": action
}
```

Argument	Type	Example
protocol	string ("dmx", "pathport", "art-net", "kinet", "sacn", "dvi", "rio-dmx")	"dmx"
action	string ("enable", "disable")	"disable"

JavaScript

```
disable_output({ "protocol": protocol }, callback)
```

```
enable_output({ "protocol": protocol }, callback)
```

Argument	Type	Example
protocol	string ("dmx", "pathport", "art-net", "kinet", "sacn", "dvi", "rio-dmx")	"dmx"

Example:

```
disbale_output({ "protocol": "dmx"}); // Disable the DMX output

enable_output({ "protocol": "art-net"}); // Enable the Art-Net Output
```

Set Timeline Source Bus

Set the time source for a timeline. Show

Lua

```
Timeline:set_default_source()

Timeline:set_timecode_source(timecodeBus[, offset])

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

Argument	Type	Example
Timeline	Timeline Object	get_timeline(1)
timecodeBus	TCODE_1 ... TCODE_6	TCODE_1
audioBus	AUDIO_1 ... AUDIO_4	AUDIO_1
band	integer (0=volume)	0
channel	LEFT, RIGHT or COMBINED	LEFT
peak	boolean (default false)	false

Example:

```
get_timeline(1):set_timecode_source(TCODE_1) -- Set the timecode source of timeline 1 to timecode bus 1
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Enable Timecode Bus

Enables or disables a timecode bus. Show

Lua

```
set_timecode_bus_enabled(bus[, enable])
```

- bus is the timecode bus to enable or disable (TCODE_1 ... TCODE_6)
- enable determines whether the bus should be enabled or disabled (boolean, default true)

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

API Subscriptions

Subscriptions allow data to be pushed to the web interface whenever there is a change within the project. show

Subscribe Timeline Status

Subscribes to changes in the timeline status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_timeline_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	'running'
onstage	boolean	true
position	number (milliseconds)	5000

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_timeline_status(function(t){
        alert(t.num + ": " + t.state)
})
```

Subscribe Scene Status

Subscribes to changes in the scene status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_scene_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	'running'
onstage	boolean	true

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_scene_status(function(s){
        alert(s.num + ": " + s.state)
})
```

Subscribe Group Status

Subscribes to changes in group level, as set by the Master Intensity action (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_group_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
name	string	'Group 1'
level	integer (0-255)	128

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_group_status(function(g){
        alert(g.num + ": " + g.level)
})
```

Subscribe Remote Device Status

Subscribes to changes in Remote Device Online/Offline Status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_remote_device_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
type	string ('RIO 08', 'RIO 44', 'RIO 80','RIO D', 'RIO A', 'BPS')	'Group 1'
online	boolean	true
serial	string (of serial number)	'001001'

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_remote_device_status(function(r){
        alert(r.num + ": " + r.level)
})
```

Subscribe Beacon

Subscribes to Beacons (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_beacon(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
on	boolean	true

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_beacon(function(b){
        if (b.on){
                alert("Beacon Turned On")
        else {
```

```
                alert("Beacon Turned Off")
        }
})
```

Subscribe Lua

The receiver for the push_to_web() Lua function. Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

`subscribe_lua(callback)`

Returns an object with the following properties:

Property	Return type	Return Example
key	as defined by push_to_web()	value

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_lua(function(l){
        key = Object.keys(l)[0]
        value = l.key
        alert(key + ": " + value)
})
```

API Objects

Below are the helper functions and objects in the project. show

Variant

A Lua object that allows a type and range to be associated with a variable. Show

Lua

See here.

HTTP

Not currently available.

JavaScript

Not currently available.

DateTime

A Lua object containing time data. Show

Lua

The DateTime object contains the following properties:

Property	Return Type	Return Example
.year	integer	2017
.month	integer (0-11)	5
.monthday	integer (0-30)	8
.weekday	integer(0-6)	1
.hour	integer(0-23)	13
.minute	integer (0-59)	21
.second	integer (0-59)	46
.utc_timestamp	integer	1494249706
.time_string	string	
.date_string	string	

HTTP

Not currently available.

JavaScript

Not currently available.

Printing an enum

Lua functions to convert integers returned from some functions as text. Show

Lua

```
digital_input_to_string()
```

```
button_state_to_string()
```

Examples:

```
log(digital_input_to_string(get_input(1)))
```

```
str = button_state_to_string(get_bps(1):get_state(1))
```

HTTP

Not currently available.

JavaScript

Not currently available.

API v2

The Pharos system includes multiple API options:

- Lua (used internally with Conditions and Actions)
- HTTP (used with external devices/software to communicate with a controller)
- JavaScript (used with Custom Web Interfaces)

These APIs have been unified to simplify their use as much as possible.

Glossary:

- Object - A collection of key value pairs e.g. "name" = "Controller 1" (syntax will differ between languages).
- String - A series of characters e.g."Th1s_is-4(string)"
- Number - Any whole or floating point(decimal) number e.g. 1,2,3,1.5,12.3456)
- Integer - A whole number
- Bounded integer - An integer with a range (e.g. 10:100 = 10%)
- Float/real/number - A decimal number (e.g. 3.2 or 1.0)
- JSON - (JavaScript Object Notation) a way of transferring information in the form of a JavaScript Object
- GET - A HTTP method to request data from a server
- POST - A HTTP method to request data in a more secure way
- PUT - A HTTP method to send data to a server
- Variant - See here
- [] - anything shown within square brackets is optional. The square brackets should be omitted if the optional section is used.
- callback - A function to run when the javascript function has been run, or a reply has been received.

HTTP Requests

Please note, when a HTTP POST request is sent, it must include a Content-Type header set to "application/json", otherwise it will be treated as invalid.

API Queries

Below are the ways of getting data from the controller. show

System

Returns data about the controller. show

Lua

The system namespace has the following properties:

Property	Return type	Return Example
.hardware_type	string	"lpc"
.channel_capacity	integer	512
.serial_number	string	"006321"
.memory_total	string	"12790Kb"
.memory_used	string	"24056Kb"
.memory_free	string	"103884Kb"
.storage_size	string	"1914MB"
.bootloader_version	string	"0.9.0"
.firmware_version	string	"2.14.0"

.reset_reason	string	"Software Reset"
.last_boot_time	DateTime object	
.ip_address	string	"192.168.1.3"
.subnet_mask	string	"255.255.255.0"
.default_gateway	string	"192.168.1.3"

Example:

```
capacity = system.channel_capacity

boot_time = system.last_boot_time.time_string
```

HTTP

```
GET /api/system
```

Returns an object with the following properties:

Property	Return type	Return Example
hardware_type	string	"LPC"
channel_capacity	integer	512
serial_number	string	"006321"
memory_total	string	"12790Kb"
memory_used	string	"24056Kb"
memory_free	string	"103884Kb"
storage_size	string	"1914MB"
bootloader_version	string	"0.9.0"
firmware_version	string	"2.14.0"
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"
default_gateway	string	"192.168.1.3"

JavaScript

```
get_system_info(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_system_info( function(system){
   var capacity = system.channel_capacity
}
```

Project

Returns data about the project. Show

Lua

get_current_project()

Returns an object with the following properties:

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

Example:

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

HTTP

```
GET /api/project
```

Returns an object with the following properties:

Property	Return type	Return Example
name	string	"Help Project"
author	string	"Pharos"
filename	string	"help_project_v1.pd2"
unique_id	string	"{6b48627a-1d5e-4b2f-81e2-481e092a6a79}"
upload_date	string	"2017-01-30T15:19:08"

JavaScript

```
get_project_info(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_project_info( function(project){
 var author = project.author
}
```

Replication

Returns data about the install replication. Show

Lua

get_current_replication()

Returns an object with the following properties:

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

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Time

Returns data about the time stored in the controller. Show

Lua

The time namespace has the following functions which 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()

and the following properties

Property	Return Type	Return Example
is_dst	boolean	
gmt_offset	string	

Each function returns a DateTime object, with the following properties:

Property	Return Type	Return Example
.year	integer	2017
.month	integer (1-12)	5
.monthday	integer (1-31)	8
.weekday	integer(1-7)	1
.hour	integer(0-23)	13
.minute	integer (0-59)	21
.second	integer (0-59)	46
.utc_timestamp	integer	1494249706
.time_string	string	
.date_string	string	

Example:

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

HTTP

```
GET /api/time
```

Returns an object with the following properties:

Property	Return Type	Return Example
datetime	string	"01 Feb 2017 13:44:42"
local_time	integer (controller's local time in milliseconds)	1485956682
uptime	integer (time since last boot)	493347

JavaScript

```
get_current_time(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_current_time( function(time){
 var uptime = time.uptime
}
```

Timeline

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

Lua

```
get_timeline(timelineNum)
```

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

The returned object has the following properties

Property	Return Type	Return Example
name	string	"Timeline 1"
group	string ('A', 'B', 'C', 'D','')	'A'
length	integer	10000
source_bus	integer (equivalent to constants: DEFAULT, TCODE_1 ... TCODE_6, AUDIO_1 ... AUDIO_4)	1
timecode_format	string	"SMPTE30"
audio_band	integer (0 is equivalent to constant VOLUME)	0
audio_channel	integer (equivalent to constants: LEFT, RIGHT or COMBINED)	1
audio_peak	boolean	false
time_offset	integer	5000
state	integer (equivalent to constants: Timeline.NONE, Timeline.RUNNING, Timeline.PAUSED, Timeline.HOLDING_AT_END, Timeline.RELEASED)	1
onstage	boolean	true
position	integer	5000
priority	integer (equivalent to constants: HIGH_PRIORITY, ABOVE_NORMAL_PRIORITY, NORMAL_PRIORITY, BELOW_NORMAL_PRIORITY or LOW_PRIORITY)	0

Example:

```
tl = get_timeline(1)

name = tl.name

state = tl.state

if (tl.source_bus == TCODE_1) then

    -- do something

end
```

HTTP

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

- num can be used to filter which timelines are returned and can be a single number or a string representing the required timelines (e.g. "1,2,5-9")

Returns an object with the following properties:

timelines array of timeline objects

Each timeline object contains the following properties:

Property	Return Type	Return Example
num	integer	1
name	string	"Timeline 1"
group	string('A', 'B', 'C', 'D' or empty)	"A"
length	integer	10000
source_bus	string ('internal', 'timecode_1',...'timecode_6', 'audio_1',...'audio_4')	100
timecode_format	string	"SMPTE30"
audio_band	integer (0 is volume band)	1
audio_channel	string ('left', 'right', 'combined')	"combined"
audio_peak	boolean	false
time_offset	integer	5000
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	"running"
onstage	boolean	true
position	integer	10000
priority	string ('high', 'above_normal', 'normal', 'below_normal', 'low')	"normal"

JavaScript

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

- num can be used to filter which timelines are returned and is defined as a JSON object which can contain a single number or a string representing the required timelines (e.g. "1,2,5-9")

Returns an array of timelines in the same way as the HTTP call

Example:

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

Scene

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

Lua

```
get_scene(sceneNum)
```

Returns a single Scene object for the Scene with user number SceneNum.

The returned object has the following properties

Property	Return Type	Return Example
name	string	"Scene 1"
state	string ('none', 'started')	"none"
onstage	boolean	false

Example:

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

HTTP

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

num can be used to filter which scenes are returned and can be a single number or array

Returns an object with the following properties:

scenes array of scene objects

Each scene object contains the following properties:

Property	Return Type	Return Example
name	string	"Scene 1"
num	integer	1
state	string ('none', 'started')	"none"
onstage	boolean	false

JavaScript

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

filter may contain a num property which is used to filter which scenes are returned

Returns an array of scenes in the same way as the HTTP call

Example:

```
Query.get_scene_info( function(s){
 var name = s.scenes[0].name //name of the first timeline
}, {"num":"1-4"})
```

Group

Returns data about the groups in the project. Show

Lua

```
get_group(groupNum)
```

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

The returned object has the following properties:

Property	Return Type	Return Example
name	string	"Group 1"
master_intensity_level	Variant	

Example:

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

HTTP

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

num can be used to filter which groups are returned and can be a single number or array

Returns an object with the following properties:

groups array of group objects

Each group object contains the following properties:

Property	Return Type	Return Example
num	integer (only included for user created groups)	1
name	string	"Group 1"
level	integer (0-100)	100

JavaScript

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

filter may contain a num property which is used to filter which groups are returned

Returns an array of groups in the same way as the HTTP call

Example:

```
Query.get_group_info( function(g){
 var name = g.groups[0].name //name of the first timeline
}, {"num":"1-4"})
```

NOTE: Group 0 will return data about the 'All Fixtures' group

Controller

Returns data about the controller. Show

Lua

```
get_current_controller()
```

Returns an object for the containing the following properties:

Property	Return Type	Return Example
number	integer	1
name	string	"Controller 1"

Example:

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

```
is_controller_online(controllerNumber)
```

Returns true if the controller with user number controllerNum has been discovered, and false otherwise

Example:

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

HTTP

```
GET /api/controller
```

Returns an object with the following properties:

controllers array of controller objects (one for each controller in the project)

Each controller object contains the following properties:

Property	Return Type	Return Example
num	number	1
type	string	"LPC"
name	string	"Controller 1"
serial	string	"009060"
ip_address	string (if the controller is discovered)/empty (if the controller is not discovered or is the queried controller)	"192.168.1.3" or ""
online	boolean	true

JavaScript

```
get_controller_info(callback)
```

Returns an array of controllers in the same way as the HTTP call

Example:

```
Query.get_controller_info( function(controller){
 var name = controller[0].name // name of the first controller
}
```

Temperature

Returns data about the controller's temperature. Show

Lua

```
get_temperature()
```

Returns an object with the following properties

Property	Return Type	Return Example
sys_temp	number (only for LPC X and VLC/VLC+)	40
core_temp	number (only for LPC X and VLC/VLC+)	44
ambient_temp	number (only for TPC, LPC X rev 1)	36.900001525878906
cc_temp	number (only for LPC X rev 2 and VLC/VLC+)	44
gpu_temp	number (only for VLC/VLC+)	44

Example:

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

HTTP

```
GET /api/temperature
```

Returns an object with the following properties:

Property	Return Type	Return Example
sys_temp	number (only for LPC X and VLC/VLC+)	40
core1_temp	number (only for LPC X and VLC/VLC+)	44
core2_temp	number (only for LPC X rev 1	44
ambient_temp	number (only for TPC, LPC X rev 1)	36.900001525878906
cc_temp	number (only for LPC X rev 2 and VLC/VLC+)	44
gpu_temp	number (only for VLC/VLC+)	44

JavaScript

```
get_temperature(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_temperature( function(temp){
 var ambient = temp.ambient_temp // ambient temperature of the controller
}
```

Remote Device

Returns data about the Remote Device/s in the project. Show

Lua

```
get_rio(type, num):get_input(inputNum)
```

- type can be RIO80, RIO44 or RIO08
- num is the remote device number
- inputNum is the number of the input

Returns a boolean if the input is set to Digital or Contact Closure, or an integer if the input is set to Analog.

Example:

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

```
get_bps(num):get_state(buttonNum)
```

- num is the BPS number
- buttonNum is the number of the button

Returns the state of the button, which can be RELEASED, PRESSED, HELD or REPEAT

Example:

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

HTTP

```
GET /api/remote_device
```

Returns an array of all remote devices in the project.

The returned object has the following structure

remote_devices array of Remote Device objects

Each Remote Device object contains the following properties:

Property	Return Type	Return Example
num	integer	1
type	string ('RIO08', 'RIO44', 'RIO80', 'BPS', 'BPI', 'RIO A', 'RIO D')	"RIO 44"
serial	array (all discovered serial number for the address	["001234"]

	and type)	
outputs	array (of Output objects, only present for RIO44 and RIO08 that are on the queried controller)	[{"output":1,"value":true},{"output":2,"value":true},{"output":3,"value":true},{"output":4,"value":true}]
inputs	array (of Input objects, only present for RIO44 and RIO80 that are 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 (if the remote device is detected as being online)	true

The Output object has the following properties:

Property	Return Type	Return Example
output	integer	1
state	boolean (true means the output is on, false means it is off)	false

The Input object has the following properties:

Property	Return Type	Return Example
input	integer	1
type	string ('Analog', 'Digital', 'Contact Closure')	'Digital'
value	integer or bool (depends on type)	true

JavaScript

```
get_remote_device_info(callback)
```

Returns an array of all remote devices in the project with the same properties as in the HTTP call.

Example:

```
Query.get_remote_device_info( function(remote){
 var type = remote[0].type // type of the first remote device
}
```

Text Slots

Returns data about the text slots in the project. Show

Lua

```
get_text_slot(slotName)
```

- slotName is the name of the text slot

Returns the value of slotName

Example:

```
log(get_text_slot("test_slot"))
```

HTTP

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

slotNames can be used to filter which text slots are returned and can be a single name or array

The returned object has the following structure

text_slots	array of Text Slot objects

Each Text Slot object contains the following properties:

Property	Return Type	Return Example
name	string	"text"
value	string	"example"

JavaScript

```
get_text_slot(callback[, filter])
```

filter may contain a names property which is used to filter which text slot values are returned

Returns an array of all text slots in the project with the same properties as in the HTTP call.

Example:

```
Query.get_text_slot( function(text){
 var value = text[0].value // value of the first text slot
}, {names: "test_slot1", "test_slot2"})
```

Get Log

Returns the log from the queried controller. Show

Lua

Not currently available.

HTTP

```
GET /api/log
```

The returned object has the following structure

Property	Return Type
log	string (containing the whole log of the controller)

JavaScript

Not currently available.

Protocol

Returns the protocols and universes being output from the queried controller. Show

Lua

Not currently available.

HTTP

```
GET /api/protocol
```

Returns all the universes on the queried controller

The returned object has the following structure

outputs array of Protocol objects

Each Protocol object has the following properties:

Property	Return Type	Return Example
type	integer	1
name	string	"DMX"
disabled	boolean (whether the output has been disabled via an Action)	true
universes	array (Universe objects)	{"key":{"index":1},"name":"1"},{"key":{"index":2},"name":"2"}
dmx_proxy	DMX Proxy Object (where appropriate)	{ "ip_address": "192.168.1.17", "name": "Controller 1" }

Each Universe object has the following properties:

Property	Return Type	Return Example
name	string	"1"
key	Universe Key object	{"index":1}

Each DMX Proxy object has the following properties:

Property	Return Type	Return Example
name	string (name of the controller that is outputting this universe)	'Controller 1'
ip_address	string IP Address of the controller outputting this universe	'192.168.1.23'

The properties of the Universe Key object depends upon the type:

For DMX, Pathport, sACN and Art-Net:

Property	Return Type	Return Example
index	integer	1

For KiNET:

Property	Return Type	Return Example
kinet_port	integer	1
kinet_power_supply_num	integer	1

JavaScript

```
get_protocols(callback)
```

Returns all the universes on the queried controller

The returned object has the following structure

outputs array of Protocol objects

Each Protocol object has the following properties:

Property	Return Type	Return Example
type	integer	1
name	string	"DMX"
disabled	boolean (whether the output has been disabled via an Action)	true
universes	array (Universe objects)	[{"key":{"index":1},"name":"1"},{"key":{"index":2},"name":"2"}]
dmx_proxy	DMX Proxy Object (where appropriate)	

Each Universe object has the following properties:

Property	Return Type	Return Example
name	string	"1"
key	Universe Key object	{"index":1}

Each DMX Proxy object has the following properties:

Property	Return Type	Return Example
name	string (name of the controller that is outputting this universe)	'Controller 1'
ip_address	string IP Address of the controller outputting this universe	'192.168.1.23'

The properties of the Universe Key object depends upon the type:

For DMX, Pathport, sACN and Art-Net:

Property	Return Type	Return Example
index	integer	1

For KiNET:

Property	Return Type	Return Example
kinet_port	integer	1
kinet_power_supply_num	integer	1

Output

Returns the levels being output from the queried controller. Show

Lua

```
get_dmx_universe(idx)
```

```
get_artnet_universe(idx)
```

```
get_pathport_universe(idx)
```

```
get_sacn_universe(idx)
```

- idx is the required universe number

```
get_kinet_universe(power_supply_num, port_num)
```

- power_supply_num is the power supply to return the output from
- port_num is the port to return the output from

These all return a Universe object, which has the following function

```
get_channel_value(chnl)
```

- chnl is the channel to get the value from

Example:

```
uni = get_dmx_universe(1) -- get DMX Universe 1

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

HTTP

```
GET /api/output?universe=universeKey
```

- universeKey is a string in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX.
- protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx
- remoteDeviceType can be rio08, rio44 or rio80

Example:

GET /api/output?universe=dmx:1

GET /api/output?universe=rio-dmx:rio44:1

The returned object has the following structure

Property	Return Type	Return Example
channels	array (of channel values)	[0,0,0,0,0,0,0,0,0,255,255,255...255,0,255]
disabled	boolean (whether the output has been disabled via an Action)	true
proxied_tpc_name	string (only if controller is LPC, universe is DMX 2, DMX Proxy has been enabled and the TPC is offline)	'Controller 2'

JavaScript

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

Argument	Type	Example
universekey	string or an object containing protocol and either index, kinet_power_supply_num and kinet_port or remote_device_type and remote_device_num	dmx:1

- universeKey can be either a string, or an object containing protocol and either index, kinet_power_supply_num and kinet_port or remote_device_type and remote_device_num as received from get_protocols

Returns an object with the same structure as in the HTTP call

Input

Returns the inputs on the queried controller. Show

Lua

```
get_input(idx)
```

Argument	Type	Example
idx	integer	1

Returns the value of the controllers input as a boolean or integer

Example:

```
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)
```

`get_dmx_input(chnl)`

Argument	Type	Example
chnl	integer	1

- chnl is the required channel number

Returns the value of the DMX input at channel chnl as an integer

HTTP

`GET /api/input`

The returned object has the following structure

Property	Return Type	Return Example
gpio	array (of Input objects, on LPC or TPC+EXT)	[{"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},{"input":5,"type":"Contact Closure","value":true},{"input":6,"-type":"Contact Closure","value":true},{"input":7,"type":"Contact Closure","value":true},{"input":8,"type":"Contact Closure","value":true}]
dmxIn	object (DMX Input object, if DMX Input is configured)	[0,0,0,0,0,0,0,0,0,255,255,255...255,0,255]

The Input object has the following properties:

Property	Return Type	Return Example
input	integer	1
type	string ('Analog', 'Digital', 'Contact Closure')	"Contact Closure"
value	integer or bool (depends on type)	true

The DMX Input object has the following properties:

Property	Return Type	Return Example

error	string (if DMX Input is configured but no DMX is received)	"No DMX received"
dmxInFrame	array (of channel values)	[0,0,0,0,0,0,0,0,255,255,255...255,0,255]

JavaScript

Not currently available.

Trigger

Returns the triggers in the project. Show

Lua

Not currently available.

HTTP

```
GET /api/trigger
```

The returned object has the following structure

triggers array (of Trigger objects)

The Trigger object has the following properties:

Property	Return Type	Return Example
type	string	"Startup"
num	integer	1
name	string	"Startup"
trigger_text	string	"At startup"
conditions	array (of Condition objects)	[{"text":"Before 12:00:00 every day"}]
actions	array (of Action objects	[{"text":"Start Timeline 1"}]

The Condition and Action objects have the following properties:

Property	Return Type	Return Example
text	string	"Start Timeline 1"

JavaScript

Not currently available.

Lua Variable

Returns the current value of the specified Lua variable. Show

Lua

Not currently available.

HTTP

```
GET /api/lua?variables=luaVariables
```

Argument	Type	Example
luaVariables	string or comma separated list	'myVar' or 'myVar, myVar2, myVar3'

Returns an object containing all the Lua variables requested and their values.

JavaScript

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

Argument	Type	Example
luaVariables	string or array	'myVar' or 'myVar, myVar2, myVar3'

Returns an object containing all the Lua variables requested and their values.

Example:

```
Query.get_lua_variables("myVar", function(lua){
 var value = lua.myVar
}
```

Trigger Variable

Returns the value of a variables from the trigger that ran the script. Show

Lua

```
get_trigger_variable(idx)
```

Argument	Type	Example
idx	integer	1

Returns the trigger variable at idx as a Variant object.

Example:

```
-- Use with a TPC 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
```

HTTP

Not currently available.

JavaScript

Not currently available.

Resources

Use to locate resources in the controller's memory. Show

Lua

```
get_resource_path(filename)
```

Argument	Type	Example
filename	string	'settings.txt'

Returns a path to the resource filename.

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Content Target

Returns information about a Content Target in the project. Show

Lua

On a VLC

get_content_target(compositionNum)

On a VLC+

```
get_content_target(compositionNum, type)
```

- compositionNum is the usernumber of the composition to return
- type is the type of target within the composition to return (PRIMARY, SECONDARY, OVERLAY_1, OVERLAY_2)

Returns a Content Target object with the following properties:

master_intensity_level	Variant
rotation_offset (VLC+ only)	float
x_position_offset (VLC+ only)	float
y_position_offset (VLC+ only)	float

Example:

```
target = get_content_target(1)

current_level = target.master_intensity_level

target = get_content_target(1,PRIMARY)

current_angle = target.rotation_offset
```

HTTP

Not currently available.

JavaScript

Not currently available.

API Actions

Below are the ways of changing properties or changing output on the controller. show

Start Timeline

Start a timeline in the project Show

Lua

```
get_timeline(timelineNum):start()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "start",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.start_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Start Scene

Start a scene in the project Show

Lua

```
get_scene(sceneNum):start()
```

Argument	Type	Example
sceneNum	integer	1

HTTP

```
POST /api/scene
{
   "action": "start",
   "num": sceneNum
}
```

Argument	Type	Example
sceneNum	integer	1

JavaScript

```
Query.start_scene({ "num": sceneNum}, callback)
```

Argument	Type	Example
sceneNum	integer	1

Release Timeline

Release a timeline in the project Show

Lua

```
get_timeline(timelineNum):release([fade])
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

HTTP

```
POST /api/timeline
{
  "action": "release",
  "num": timelineNum[,
  "fade": fade]
}
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

JavaScript

```
Query.release_timeline({ "num": timelineNum[, "fade": fade]}, callback)
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

Release Scene

Release a scene in the project Show

Lua

```
get_scene(sceneNum):release([fade])
```

Argument	Type	Example
sceneNum	integer	1

fade	float	2.0

HTTP

```
POST /api/scene
{
  "action": "release",
  "num": sceneNum[,
  "fade": fade]
}
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

JavaScript

```
Query.release_scene({ "num": sceneNum[, "fade": fade]}, callback)
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

Toggle Timeline

Toggle a timeline in the project (if it is running, stop it, and if it is not running, start it) Show

Lua

```
get_timeline(timelineNum):toggle([fade])
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

HTTP

```
POST /api/timeline
{
  "action": "toggle",
  "num": timelineNum[,
  "fade": fade]
}
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

JavaScript

```
Query.toggle_timeline({ "num": timelineNum[, "fade": fade]}, callback)
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

Toggle Scene

Toggle a scene in the project (if it is running, stop it, and if it is not running, start it) Show

Lua

```
get_scene(sceneNum):toggle([fade])
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

HTTP

```
POST /api/scene
{
  "action": "toggle",
  "num": sceneNum[,
  "fade": fade]
}
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

JavaScript

```
Query.toggle_scene({ "num": sceneNum[, "fade": fade]}, callback)
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

Pause Timeline

Pause a timeline in the project Show

Lua

```
get_timeline(timelineNum):pause()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "pause",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.pause_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Resume Timeline

Resume a timeline in the project Show

Lua

```
get_timeline(timelineNum):resume()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "resume",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.resume_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Pause All

Pause all timelines in the project Show

Lua

```
pause_all()
```

HTTP

```
POST /api/timeline

{

 "action": "pause"

}
```

JavaScript

```
Query.pause_all(callback)
```

Resume All

Resume all timelines in the project Show

Lua

```
resume_all()
```

HTTP

```
POST /api/timeline

{

 "action": "resume"

}
```

JavaScript

```
Query.resume_all(callback)
```

Release All

Release all timelines, scenes or timelines, scenes and overrides in the project Show

Lua

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

release_all_timelines([fade,] [group])

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

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

HTTP

```
POST /api/release_all

POST /api/timeline

POST /api/scene
```

```
{
  "action": "release"[, (not required for release all)
  "group": group][,
  "fade": fade]
}
```

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

JavaScript

```
release_all_timelines({["fade": fade]}, callback)
release_all_scenes({["fade": fade]}, callback)
release_all({ ["fade": fade,] ["group": group] }, callback)
```

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

Set Timeline Rate

Set the rate of a timeline in the project Show

Lua

```
get_timeline(timelineNum):set_rate(rate)
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

HTTP

```
POST /api/timeline
{
  "action": "set_rate",
  "num": timelineNum,
  "rate": rate
}
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

JavaScript

```
Query.set_timeline_rate({"num": timelineNum, "rate": rate }, callback)
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

Set Timeline Position

Set the position of a timeline in the project Show

Lua

```
get_timeline(timelineNum):set_position(position)
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

HTTP

```
POST /api/timeline

{

 "action": "set_position",

  "num": timelineNum,

  "position": position

}
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

JavaScript

```
Query.set_timeline_position({"num": timelineNum, "position": position }, call-
back)
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

Enqueue Trigger

Fire a trigger in the project Show

Lua

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

Argument	Type	Example
num - the trigger number to enqueue	integer	1
var... - 0 or more variables to pass	comma separated variables	1,2,"string"

to the trigger

Example

```
enqueue_trigger(1,1,2,"string")
```

HTTP

```
POST /api/trigger

{

  "num": num[,

  "var": var...][,

  "conditions": test_conditions]

}
```

Argument	Type	Example
num	integer	1
var...	comma separated variables	1,2,"string"
test_conditions	boolean	true

- num - the trigger number to enqueue
- var... - 0 or more variables to pass to the trigger
- test_conditions - Should the conditions on the trigger be tested?

JavaScript

```
Query.fire_trigger({"num": num[, "var": var...][, "conditions": test_conditions]
}, callback)
```

Argument	Type	Example
num	integer	1
var...	comma separated variables	'1,2,"string"'
test_conditions	boolean	true

- num - the trigger number to enqueue
- var... - 0 or more variables to pass to the trigger. If passing multiple variables, they must be a single string surrounded by single quotes ('), string variables should be surrounded by double quotes (").
- test_conditions - Should the conditions on the trigger be tested?

Run Script

Run a script or parse into the command line on the controller Show

Lua

Not currently available.

HTTP

```
POST /api/cmdline

{

  "input": chunk,

}
```

NOTE: returns "Executed" if successful, or an error string if not

Argument	Type	Example
chunk - the script to parse or run	string	"tl = 1 get_timeline(tl):start()"

JavaScript

```
Query.run_command({ "input": chunk }, callback)
```

NOTE: returns "Executed" if successful, or an error string if not

Argument	Type	Example
chunk - the script to parse or run	string	"tl = 1 get_timeline(tl):start()"

Hardware Reset

Reset the controller (power reboot) Show

Lua

Not currently available.

HTTP

```
POST /api/reset
```

JavaScript

Not currently available.

Master Intensity

Master the intensity of a group or content target (applied as a multiplier to output levels) Show

Lua

Non-VLC

```
get_group(groupNum):set_master_intensity(level[, fade[, delay]])
```

VLC

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

VLC+

```
get_content_target(compositionNum, type):set_master_intensity(level, [fade,
[delay]])
```

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', 'overlay_1', 'overlay_2'
level	float or integer (0-255)	128 or 0.5
fade	float	2.0
delay	float	2.0

Example:

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

HTTP

Non-VLC

```
POST /api/group

{

 "action": "master_intensity",

    "num": groupNum,

    "level": level,

   ["fade": fade,]

   ["delay": delay]

}
```

VLC/VLC+

```
POST /api/content_target

{

"action": "master_intensity",

"level": level,

["fade": fade,]

["delay": delay,]

"type": type

}
```

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', 'overlay_1', 'overlay_2'
level	float or bounded integer	0.5 or "50:100"
fade	float	2.0
delay	float	2.0

JavaScript

Non-VLC

```
master_intensity({ "num": groupNum, "level": level, ["fade": fade,] ["delay":
delay] }, callback)
```

VLC/VLC+

master_content_target_intensity({ "type":type, "level": level, ["fade": fade,] ["delay": delay] }, callback)

Argument	Type	Example

groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', 'overlay_1', 'overlay_2'
level	float or bounded integer	0.5 or "50:100"
fade	float	2.0
delay	float	2.0

Example:

```
Query.master_intensity({"num":1,"level":"50:100","fade":3) -- master group 1 to 50% (50/100 = 0.5) in 3 seconds)
```

NOTE: Group 0 will master the intensity of the 'All Fixtures' group

Set RGB

Set the Intensity, Red, Green, Blue levels for a fixture or group. Show

Lua

```
get_fixture_override(num)

get_group_override(num)

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

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

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

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

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

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

Argument	Type	Example
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated, "Damped, "Overshoot"

Example:

```
ov = get_fixture_override(1) -- Get fixture 1

ov:set_irgb(255, 255, 0, 0) -- Set the fixture to Red
```

HTTP

```
PUT /api/override
```

```
{
   "target": target,
   "num": num,
   ["intensity": intensity,]
   ["red": red,]
   ["green": green,]
   ["blue": blue,]
   ["temperature": temperature,]
   ["fade": fade,]
   ["path": path]
}
```

Argument	Type	Example
target	string (from options)	"group", "fixture"
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated, "Damped, "Overshoot"

Javascript

```
set_group_override({ "num": num, ["intensity": intensity,] ["red": red,]
["green": green,] ["blue": blue,] ["temperature": temperature,] ["fade": fade,]
["path": path] }, callback)
```

```
set_fixture_override({ "num": num, ["intensity": intensity,] ["red": red,]
["green": green,] ["blue": blue,] ["temperature": temperature,] ["fade": fade,]
["path": path] }, callback)
```

Argument	Type	Example
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from	"Default", "Linear", "Start", "End", "Braked", "Accelerated,

	options)	"Damped, "Overshoot"

Example:

```
Query.set_fixture_override({ "num": 1, "intensity": 255, "red": 255, "green": 0, "blue": 0});
```

NOTE: Group 0 will set the levels of the 'All Fixtures' group

Clear RGB

Remove any overrides on fixtures or groups. Show

Lua

```
get_fixture_override(num)

get_group_override(num)
   :clear([fade])

clear_all_overrides([fade])
```

Argument	Type	Example
num - group or fixture	integer	1
fade	float	2.0

Example:

```
ov = get_fixture_override(1) -- Get fixture 1

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

HTTP

```
DELETE /api/override

{
   ["target": target,]
   ["num": objectNum,]
   ["fade": fade]
}
```

If num is not included, target is ignored and all overrides are cleared.

Argument	Type	Example
target	string (from options)	"group" or "fixture"
num - group or fixture	integer	1
fade	float	2.0

JavaScript

```
clear_group_overrides({ ["num" :num,] ["fade": fade] }, callback)

clear_fixture_overrides({ ["num" :num,] ["fade": fade] }, callback)

clear_overrides({ ["fade": fade] }, callback)
```

Argument	Type	Example
num	integer	1
fade	float	2.0

Example:

```
Query.clear_overrides({"fade":3})
```

Set Text Slot

Set the value of a text slot used in the project. Show

Lua

```
set_text_slot(name, value)
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

Example:

```
set_text_slot("myTextSlot", "Hello World!")
```

HTTP

```
PUT /api/text_slot
{
   "name": name,
   "value": value
}
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

JavaScript

```
set_text_slot({"name": name, "value": value}, callback)
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

Example:

```
Query.set_text_slot("name:"myTextSlot", "value":"Hello World!")
```

Set BPS Button LED

Set the effect and intensity on BPS button LEDS. Show

Lua

```
get_bps(num):set_led(button, effect, [intensity], [fade])
```

Argument	Type	Example
num	integer	1
button	integer	1
effect	OFF, ON, SLOW_FLASH, FAST_FLASH, DOUBLE_FLASH, BLINK, PULSE, SINGLE, RAMP_ON, RAMP_OFF	FAST_FLASH
intensity	integer (1-255)	255
fade	float	0.0

Example:

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

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set TPC Control Value

Set the value on a TPC Slider or Color Picker. Show

Lua

```
set_control_value(name, [index,] value[, emitChange])
```

Argument	Type	Example
name - control Key	string	"slider001"
index - axis of movement (slider has 1, colour picker has 3)	integer (1-3) (default 1)	1
value	integer (0-255)	128
emitChange	boolean (default false)	false

Example:

```
set_control_value("slider001", 1, 128) -- set slider001 to half and don't fire associated triggers
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set TPC Control State

Set the state on a TPC control. Show

Lua

```
set_control_state(name, state)
```

Argument	Type	Example
name - control Key	string	"slider001"
state - the state name form Interface	string (from options in Interface)	"Green"

Example:

```
set_control_state("slider001", "Green") -- set slider001 to a state called
"Green"
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set TPC Control Caption

Set the caption on a TPC control. Show

Lua

```
set_control_caption(name, caption)
```

Argument	Type	Example
name - control Key	string	"button001"
caption - text to display	string	"On"

Example:

```
set_control_caption("button001", "On") -- set button001's caption to "On"
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set TPC Page

Change the page on a TPC interface. Show

Lua

```
set_interface_page(number[, transition])
```

Argument	Type	Example
number	integer	4
transition	SNAP, PAN_LEFT, PAN_RIGHT	PAN_LEFT

Example:

```
set_interface_page(4) -- change the page on the TPC's interface to page 4
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Disable Page

Disable the TPC touchscreen. Show

Lua

```
set_interface_enabled([enable])
```

Argument	Type	Example
enable	boolean (default true)	true

Example:

```
set_interface_enabled(false) -- disable the TPC's touch screen

set_interface_enabled() -- enable the TPC's touch screen
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Lock TPC

Lock the TPC (requires Lock code to be set within Interface). Show

Lua

```
set_interface_locked([lock])
```

Argument	Type	Example
lock	boolean (default true)	true

Example:

```
set_interface_enabled(false) -- disable the TPC's touch screen
set_interface_enabled() -- enable the TPC's touch screen
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Transition Content Target

Move or rotate a Content Target. Show

Lua

```
get_content_target(compositionNum, type)
  :transition_rotation([offset], [count], [period], [delay], [useShortestPath])
  :transition_x_position([offset], [count], [period], [delay])
  :transition_y_position([offset], [count], [period], [delay])
```

Argument	Type	Example
compositionNum	integer	1
type	PRIMARY, SECONDARY, OVERLAY_1, OVERLAY_2	PRIMARY
offset	integer	10
count	integer	1
period	integer	5
delay	integer	0
useShortestPath	boolean (default false)	false

Example:

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

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Beacon Controller

Beacons the controller (flashes Status LEDs or screen). Show

Lua

Not currently available.

HTTP

```
POST /api/beacon
```

JavaScript

```
toggle_beacon(callback)
```

Example:

```
Query.toggle_beacon()
```

Output to Log

Writes a message to the controller's Log. Show

Lua

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

Argument	Type	Example
level	option (LOG_DEBUG, LOG_TERSE, LOG_NORMAL, LOG_EXTENDED, LOG_VERBOSE, LOG_CRITICAL, default LOG_NORMAL)	LOG_CRITICAL
message	string	"Some message to log."

Example:

log(LOG_CRITICAL, "This is a criticial message!") -- logs the message at Critical log level

log("This is a normal message.") -- logs the message at Normal log level.

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Send variables to Web Interface

Sends data to the web interface in a JSON Object. Show

Lua

```
push_to_web(name, value)
```

Argument	Type	Example
name	string	"myVar"
value	variable	"Some value"

Example:

```
myVar = 15

push_to_web("myVar", myVar) -- will push the object {"myVar": 15}
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Park a Channel

Parks an output channel at a specified level. Show

Lua

```
Universe:park(channel, value)
```

Argument	Type	Example
Universe	Universe object	get_dmx_universe(1)
channel	integer (1-512)	1
value	integer (0-255)	128

Example:

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

HTTP

```
POST /api/channel

{

    "universe": universeKey,

    "channels": channelList,

    "level": level

}
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-	"dmx:1"

	Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	
channelList	comma separated list(1-512)	"1-3,5"
level	integer (0-255)	128

JavaScript

```
park_channel({ "universe": universeKey, "channels": channelList, "level": level }, callback)
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"
value	integer (0-255)	128

Example:

```
park_channel({ "universe": "dmx:1","channels": 1, "level":128}); // Park channel 1 of DMX Universe 1 at 128 (50%)
```

Unpark a Channel

Unparks an output channel. Show

Lua

```
Universe:unpark(channel)
```

Argument	Type	Example
Universe	Universe object	get_dmx_universe(1)
channel	integer (1-512)	1

Example:

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

HTTP

```
DELETE /api/channel

{

    "universe": universeKey,

    "channels": channelList

}
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"

JavaScript

```
park_channel({ "universe": universeKey, "channels": channelList }, callback)
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"

Example:

```
park_channel({ "universe": "dmx:1","channels": 1}); //Unpark channel 1 of DMX Universe 1 (it will go back to normal output levels)
```

Disable an Output

Unparks an output channel. Show

Lua

```
disable_output(protocol)
```

```
enable_output(protocol)
```

Argument	Type	Example
protocol	option (DMX, PATHPORT, ARTNET, KINET, SACN, DVI, RIO_DMX)	DMX

Example:

disable_output(DMX) -- Disable the DMX output from the controller

HTTP

```
POST /api/output
```

```
{
```

```
    "protocol": protocol,
```

```
    "action": action
```

```
}
```

Argument	Type	Example

protocol	string ("dmx", "pathport", "art-net", "kinet", "sacn", "dvi", "rio-dmx")	"dmx"
action	string ("enable", "disable")	"disable"

JavaScript

```
disable_output({ "protocol": protocol }, callback)
```

```
enable_output({ "protocol": protocol }, callback)
```

Argument	Type	Example
protocol	string ("dmx", "pathport", "art-net", "kinet", "sacn", "dvi", "rio-dmx")	"dmx"

Example:

```
disbale_output({ "protocol": "dmx"}); // Disable the DMX output
```

```
enable_output({ "protocol": "art-net"}); // Enable the Art-Net Output
```

Set Timeline Source Bus

Set the time source for a timeline. Show

Lua

```
Timeline:set_default_source()
```

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

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

Argument	Type	Example
Timeline	Timeline Object	get_timeline(1)
timecodeBus	TCODE_1 ... TCODE_6	TCODE_1
audioBus	AUDIO_1 ... AUDIO_4	AUDIO_1
band	integer (0=volume)	0
channel	LEFT, RIGHT or COMBINED	LEFT
peak	boolean (default false)	false

Example:

```
get_timeline(1):set_timecode_source(TCODE_1) -- Set the timecode source of timeline 1 to timecode bus 1
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Enable Timecode Bus

Enables or disables a timecode bus. Show

Lua

```
set_timecode_bus_enabled(bus[, enable])
```

- bus is the timecode bus to enable or disable (TCODE_1 ... TCODE_6)
- enable determines whether the bus should be enabled or disabled (boolean, default true)

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

API Subscriptions

Subscriptions allow data to be pushed to the web interface whenever there is a change within the project. show

Subscribe Timeline Status

Subscribes to changes in the timeline status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_timeline_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	'running'
onstage	boolean	true
position	number (milliseconds)	5000

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_timeline_status(function(t){
        alert(t.num + ": " + t.state)
})
```

Subscribe Scene Status

Subscribes to changes in the scene status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_scene_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	'running'
onstage	boolean	true

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_scene_status(function(s){
        alert(s.num + ": " + s.state)
})
```

Subscribe Group Status

Subscribes to changes in group level, as set by the Master Intensity action (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_group_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
name	string	'Group 1'
level	integer (0-255)	128

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_group_status(function(g){
        alert(g.num + ": " + g.level)
```

```
})
```

Subscribe Remote Device Status

Subscribes to changes in Remote Device Online/Offline Status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_remote_device_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
type	string ('RIO 08', 'RIO 44', 'RIO 80','RIO D', 'RIO A', 'BPS')	'Group 1'
online	boolean	true
serial	string (of serial number)	'001001'

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_remote_device_status(function(r){
        alert(r.num + ": " + r.level)
})
```

Subscribe Beacon

Subscribes to Beacons (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_beacon(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
on	boolean	true

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_beacon(function(b){
        if (b.on){
                alert("Beacon Turned On")
        else {
                alert("Beacon Turned Off")
        }
})
```

Subscribe Lua

The receiver for the push_to_web() Lua function. Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_lua(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
key	as defined by push_to_web()	value

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_lua(function(l){
        key = Object.keys(l)[0]
        value = l.key
        alert(key + ": " + value)
})
```

API Objects

Below are the helper functions and objects in the project. show

Variant

A Lua object that allows a type and range to be associated with a variable. Show

Lua

See here.

HTTP

Not currently available.

JavaScript

Not currently available.

DateTime

A Lua object containing time data. Show

Lua

The DateTime object contains the following properties:

Property	Return Type	Return Example
.year	integer	2017
.month	integer (0-11)	5
.monthday	integer (0-30)	8
.weekday	integer(0-6)	1
.hour	integer(0-23)	13
.minute	integer (0-59)	21
.second	integer (0-59)	46
.utc_timestamp	integer	1494249706
.time_string	string	
.date_string	string	

HTTP

Not currently available.

JavaScript

Not currently available.

Printing an enum

Lua functions to convert integers returned from some functions as text. Show

Lua

```
digital_input_to_string()

button_state_to_string()
```

Examples:

```
log(digital_input_to_string(get_input(1)))

str = button_state_to_string(get_bps(1):get_state(1))
```

HTTP

Not currently available.

JavaScript

Not currently available.

API v1

The Pharos system includes multiple API options:

- Lua (used internally with Conditions and Actions)
- HTTP (used with external devices/software to communicate with a controller)
- JavaScript (used with Custom Web Interfaces)

These APIs have been unified to simplify their use as much as possible.

Glossary:

- Object - A collection of key value pairs e.g. "name" = "Controller 1" (syntax will differ between languages).
- String - A series of characters e.g."Th1s_is-4(string)"
- Number - Any whole or floating point(decimal) number e.g. 1,2,3,1.5,12.3456)
- Integer - A whole number
- Bounded integer - An integer with a range (e.g. 10:100 = 10%)
- Float/real/number - A decimal number (e.g. 3.2 or 1.0)
- JSON - (JavaScript Object Notation) a way of transferring information in the form of a JavaScript Object
- GET - A HTTP method to request data from a server
- POST - A HTTP method to request data in a more secure way
- PUT - A HTTP method to send data to a server
- Variant - See here
- [] - anything shown within square brackets is optional. The square brackets should be omitted if the optional section is used.
- callback - A function to run when the javascript function has been run, or a reply has been received.

HTTP Requests

Please note, when a HTTP POST request is sent, it must include a Content-Type header set to "application/json", otherwise it will be treated as invalid.

API Queries

Below are the ways of getting data from the controller. show

System

Returns data about the controller. show

Lua

The system namespace has the following properties:

Property	Return type	Return Example
.hardware_type	string	"lpc"
.channel_capacity	integer	512
.serial_number	string	"006321"
.memory_total	string	"12790Kb"
.memory_used	string	"24056Kb"
.memory_free	string	"103884Kb"
.storage_size	string	"1914MB"
.bootloader_version	string	"0.9.0"
.firmware_version	string	"2.14.0"

.reset_reason	string	"Software Reset"
.last_boot_time	DateTime object	
.ip_address	string	"192.168.1.3"
.subnet_mask	string	"255.255.255.0"
.default_gateway	string	"192.168.1.3"

Example:

```
capacity = system.channel_capacity

boot_time = system.last_boot_time.time_string
```

HTTP

```
GET /api/system
```

Returns an object with the following properties:

Property	Return type	Return Example
hardware_type	string	"LPC"
channel_capacity	integer	512
serial_number	string	"006321"
memory_total	string	"12790Kb"
memory_used	string	"24056Kb"
memory_free	string	"103884Kb"
storage_size	string	"1914MB"
bootloader_version	string	"0.9.0"
firmware_version	string	"2.14.0"
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"
default_gateway	string	"192.168.1.3"

JavaScript

```
get_system_info(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_system_info( function(system){

   var capacity = system.channel_capacity

}
```

Project

Returns data about the project. Show

Lua

get_current_project()

Returns an object with the following properties:

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

Example:

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

HTTP

```
GET /api/project
```

Returns an object with the following properties:

Property	Return type	Return Example
name	string	"Help Project"
author	string	"Pharos"
filename	string	"help_project_v1.pd2"
unique_id	string	"{6b48627a-1d5e-4b2f-81e2-481e092a6a79}"
upload_date	string	"2017-01-30T15:19:08"

JavaScript

```
get_project_info(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_project_info( function(project){
 var author = project.author
}
```

Replication

Returns data about the install replication. Show

Lua

get_current_replication()

Returns an object with the following properties:

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

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Time

Returns data about the time stored in the controller. Show

Lua

The time namespace has the following functions which 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()

and the following properties

Property	Return Type	Return Example
is_dst	boolean	
gmt_offset	string	

Each function returns a DateTime object, with the following properties:

Property	Return Type	Return Example
.year	integer	2017
.month	integer (1-12)	5
.monthday	integer (1-31)	8
.weekday	integer(0-6) (Sunday = 0)	1
.hour	integer(0-23)	13
.minute	integer (0-59)	21
.second	integer (0-59)	46
.utc_timestamp	integer	1494249706
.time_string	string	
.date_string	string	

Example:

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

HTTP

```
GET /api/time
```

Returns an object with the following properties:

Property	Return Type	Return Example
datetime	string	"01 Feb 2017 13:44:42"
utc	integer (controller's local time in milliseconds	1485956682
uptime	integer (time since last boot)	493347

JavaScript

```
get_current_time(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_current_time( function(time){
 var uptime = time.uptime
}
```

Timeline

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

Lua

```
get_timeline(timelineNum)
```

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

The returned object has the following properties

Property	Return Type	Return Example
name	string	"Timeline 1"
group	string ('A', 'B', 'C', 'D','')	'A'
length	integer	10000
source_bus	integer (equivalent to constants: DEFAULT, TCODE_1 ... TCODE_6, AUDIO_1 ... AUDIO_4)	1
timecode_format	string	"SMPTE30"
audio_band	integer (0 is equivalent to constant VOLUME)	0
audio_channel	integer (equivalent to constants: LEFT, RIGHT or COMBINED)	1
audio_peak	boolean	false
time_offset	integer	5000
state	integer (equivalent to constants: Timeline.NONE, Timeline.RUNNING, Timeline.PAUSED, Timeline.HOLDING_AT_END, Timeline.RELEASED)	1
onstage	boolean	true
position	integer	5000
priority	integer (equivalent to constants: HIGH_PRIORITY, ABOVE_NORMAL_PRIORITY, NORMAL_PRIORITY, BELOW_NORMAL_PRIORITY or LOW_PRIORITY)	0

Example:

```
tl = get_timeline(1)

name = tl.name

state = tl.state

if (tl.source_bus == TCODE_1) then

    -- do something

end
```

HTTP

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

- num can be used to filter which timelines are returned and can be a single number or a string representing the required timelines (e.g. "1,2,5-9")

Returns an object with the following properties:

timelines array of timeline objects

Each timeline object contains the following properties:

Property	Return Type	Return Example
num	integer	1
name	string	"Timeline 1"
group	string('A', 'B', 'C', 'D' or empty)	"A"
length	integer	10000
source_bus	string ('internal', 'timecode_1',...'timecode_6', 'audio_1',...'audio_4')	100
timecode_format	string	"SMPTE30"
audio_band	integer (0 is volume band)	1
audio_channel	string ('left', 'right', 'combined')	"combined"
audio_peak	boolean	false
time_offset	integer	5000
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	"running"
onstage	boolean	true
position	integer	10000
priority	string ('high', 'above_normal', 'normal', 'below_normal', 'low')	"normal"

JavaScript

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

- num can be used to filter which timelines are returned and is defined as a JSON object which can contain a single number or a string representing the required timelines (e.g. "1,2,5-9")

Returns an array of timelines in the same way as the HTTP call

Example:

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

Scene

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

Lua

```
get_scene(sceneNum)
```

Returns a single Scene object for the Scene with user number SceneNum.

The returned object has the following properties

Property	Return Type	Return Example
name	string	"Scene 1"
state	string ('none', 'started')	"none"
onstage	boolean	false

Example:

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

HTTP

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

num can be used to filter which scenes are returned and can be a single number or array

Returns an object with the following properties:

scenes array of scene objects

Each scene object contains the following properties:

Property	Return Type	Return Example
name	string	"Scene 1"
num	integer	1
state	string ('none', 'started')	"none"
onstage	boolean	false

JavaScript

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

filter may contain a num property which is used to filter which scenes are returned

Returns an array of scenes in the same way as the HTTP call

Example:

```
Query.get_scene_info( function(s){
 var name = s.scenes[0].name //name of the first timeline
}, {"num":"1-4"})
```

Group

Returns data about the groups in the project. Show

Lua

```
get_group(groupNum)
```

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

The returned object has the following properties:

Property	Return Type	Return Example
name	string	"Group 1"
master_intensity_level	Variant	

Example:

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

HTTP

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

num can be used to filter which groups are returned and can be a single number or array

Returns an object with the following properties:

groups array of group objects

Each group object contains the following properties:

Property	Return Type	Return Example
num	integer	1
name	string	"Group 1"
level	integer (0-100)	100

JavaScript

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

filter may contain a num property which is used to filter which groups are returned

Returns an array of groups in the same way as the HTTP call

Example:

```
Query.get_group_info( function(g){
 var name = g.groups[0].name //name of the first timeline
}, {"num":"1-4"})
```

NOTE: Group 0 will return data about the 'All Fixtures' group

Controller

Returns data about the controller. Show

Lua

```
get_current_controller()
```

Returns an object for the containing the following properties:

Property	Return Type	Return Example
number	integer	1
name	string	"Controller 1"

Example:

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

```
is_controller_online(controllerNumber)
```

Returns true if the controller with user number controllerNum has been discovered, and false otherwise

Example:

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

HTTP

```
GET /api/controller
```

Returns an object with the following properties:

controllers array of controller objects (one for each controller in the project)

Each controller object contains the following properties:

Property	Return Type	Return Example
num	number	1
type	string	"LPC"
name	string	"Controller 1"
serial	string	"009060"
ip_address	string (if the controller is discovered)/empty (if the controller is not discovered or is the queried controller)	"192.168.1.3" or ""
online	boolean	true

JavaScript

```
get_controller_info(callback)
```

Returns an array of controllers in the same way as the HTTP call

Example:

```
Query.get_controller_info( function(controller){
 var name = controller[0].name // name of the first controller
}
```

Temperature

Returns data about the controller's temperature. Show

Lua

```
get_temperature()
```

Returns an object with the following properties

Property	Return Type	Return Example
sys_temp	number (only for LPC X and VLC/VLC+)	40
core1_temp	number (only for LPC X and VLC/VLC+)	44
core2_temp	number (only for LPC X rev 1	44
ambient_temp	number (only for TPC, LPC X rev 1)	36.900001525878906
cc_temp	number (only for LPC X rev 2 and VLC/VLC+)	44
gpu_temp	number (only for VLC/VLC+)	44

Example:

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

HTTP

```
GET /api/temperature
```

Returns an object with the following properties:

Property	Return Type	Return Example
sys_temp	number (only for LPC X and VLC/VLC+)	40
core1_temp	number (only for LPC X and VLC/VLC+)	44
core2_temp	number (only for LPC X rev 1	44
ambient_temp	number (only for TPC, LPC X rev 1)	36.900001525878906
cc_temp	number (only for LPC X rev 2 and VLC/VLC+)	44
gpu_temp	number (only for VLC/VLC+)	44

JavaScript

```
get_temperature(callback)
```

Returns an object with the same properties as in the HTTP call

Example:

```
Query.get_temperature( function(temp){
 var ambient = temp.ambient_temp // ambient temperature of the controller
}
```

Remote Device

Returns data about the Remote Device/s in the project. Show

Lua

```
get_rio(type, num):get_input(inputNum)
```

- type can be RIO80, RIO44 or RIO08
- num is the remote device number
- inputNum is the number of the input

Returns a boolean if the input is set to Digital or Contact Closure, or an integer if the input is set to Analog.

Example:

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

```
get_bps(num):get_state(buttonNum)
```

- num is the BPS number
- buttonNum is the number of the button

Returns the state of the button, which can be RELEASED, PRESSED, HELD or REPEAT

Example:

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

HTTP

```
GET /api/remote_device
```

Returns an array of all remote devices in the project.

The returned object has the following structure

remote_devices array of Remote Device objects

Each Remote Device object contains the following properties:

Property	Return Type	Return Example
num	integer	1
type	string ('RIO08', 'RIO44', 'RIO80', 'BPS', 'BPI', 'RIO A', 'RIO D')	"RIO 44"
serial	array (all discovered serial number for the address	["001234"]

	and type)	
outputs	array (of Output objects, only present for RIO44 and RIO08 that are on the queried controller)	[{"output":1,"value":true},{"output":2,"value":true},{"output":3,"value":true},{"output":4,"value":true}]
inputs	array (of Input objects, only present for RIO44 and RIO80 that are 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 (if the remote device is detected as being online)	true

The Output object has the following properties:

Property	Return Type	Return Example
output	integer	1
state	boolean (true means the output is on, false means it is off)	false

The Input object has the following properties:

Property	Return Type	Return Example
input	integer	1
type	string ('Analog', 'Digital', 'Contact Closure')	'Digital'
value	integer or bool (depends on type)	true

JavaScript

```
get_remote_device_info(callback)
```

Returns an array of all remote devices in the project with the same properties as in the HTTP call.

Example:

```
Query.get_remote_device_info( function(remote){
 var type = remote[0].type // type of the first remote device
}
```

Text Slots

Returns data about the text slots in the project. Show

Lua

```
get_text_slot(slotName)
```

- slotName is the name of the text slot

Returns the value of slotName

Example:

```
log(get_text_slot("test_slot"))
```

HTTP

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

slotNames can be used to filter which text slots are returned and can be a single name or array

The returned object has the following structure

text_slots	array of Text Slot objects

Each Text Slot object contains the following properties:

Property	Return Type	Return Example
name	string	"text"
value	string	"example"

JavaScript

```
get_text_slot(callback[, filter])
```

filter may contain a names property which is used to filter which text slot values are returned

Returns an array of all text slots in the project with the same properties as in the HTTP call.

Example:

```
Query.get_text_slot( function(text){
 var value = text[0].value // value of the first text slot
}, {names: "test_slot1", "test_slot2"})
```

Get Log

Returns the log from the queried controller. Show

Lua

Not currently available.

HTTP

```
GET /api/log
```

The returned object has the following structure

Property	Return Type
log	string (containing the whole log of the controller)

JavaScript

Not currently available.

Protocol

Returns the protocols and universes being output from the queried controller. Show

Lua

Not currently available.

HTTP

```
GET /api/protocol
```

Returns all the universes on the queried controller

The returned object has the following structure

outputs array of Protocol objects

Each Protocol object has the following properties:

Property	Return Type	Return Example
type	integer	1
name	string	"DMX"
universes	array (Universe objects)	{"key":{"index":1},"name":"1"},{"key":{"index":2},"-name":"2"}
dmx_proxy	DMX Proxy Object (where appropriate)	{ "ip_address": "192.168.1.17", "name": "Controller 1" }

Each Universe object has the following properties:

Property	Return Type	Return Example
name	string	"1"
key	Universe Key object	{"index":1}

Each DMX Proxy object has the following properties:

Property	Return Type	Return Example
name	string (name of the controller that is outputting this universe)	'Controller 1'
ip_address	string IP Address of the controller outputting this universe	'192.168.1.23'

The properties of the Universe Key object depends upon the type:

For DMX, Pathport, sACN and Art-Net:

Property	Return Type	Return Example
index	integer	1

For KiNET:

Property	Return Type	Return Example
kinet_port	integer	1
kinet_power_supply_num	integer	1

JavaScript

```
get_protocols(callback)
```

Returns all the universes on the queried controller

The returned object has the following structure

outputs array of Protocol objects

Each Protocol object has the following properties:

Property	Return Type	Return Example
type	integer	1
name	string	"DMX"
universes	array (Universe objects)	[{"key":{"index":1},"name":"1"},{"key":{"index":2},"-name":"2"}]
dmx_proxy	DMX Proxy Object (where appropriate)	

Each Universe object has the following properties:

Property	Return Type	Return Example
name	string	"1"
key	Universe Key object	{"index":1}

Each DMX Proxy object has the following properties:

Property	Return Type	Return Example
name	string (name of the controller that is outputting this universe)	'Controller 1'
ip_address	string IP Address of the controller outputting this universe	'192.168.1.23'

The properties of the Universe Key object depends upon the type:

For DMX, Pathport, sACN and Art-Net:

Property	Return Type	Return Example
index	integer	1

For KiNET:

Property	Return Type	Return Example
kinet_port	integer	1
kinet_power_supply_num	integer	1

Output

Returns the levels being output from the queried controller. Show

Lua

```
get_dmx_universe(idx)
```

```
get_artnet_universe(idx)
```

```
get_pathport_universe(idx)
```

```
get_sacn_universe(idx)
```

- idx is the required universe number

```
get_kinet_universe(power_supply_num, port_num)
```

- power_supply_num is the power supply to return the output from
- port_num is the port to return the output from

These all return a Universe object, which has the following function

```
get_channel_value(chnl)
```

- chnl is the channel to get the value from

Example:

```
uni = get_dmx_universe(1) -- get DMX Universe 1

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

HTTP

```
GET /api/output?universe=universeKey
```

- universeKey is a string in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX.
- protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx
- remoteDeviceType can be rio08, rio44 or rio80

Example:

GET /api/output?universe=dmx:1

GET /api/output?universe=rio-dmx:rio44:1

The returned object has the following structure

Property	Return Type	Return Example
channels	array (of channel values)	[0,0,0,0,0,0,0,0,0,255,255,255...255,0,255]
proxied_tpc_name	string (only if controller is LPC, universe is DMX 2, DMX Proxy has been enabled and the TPC is offline)	'Controller 2'

JavaScript

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

Argument	Type	Example
universekey	string or an object containing protocol and either index, kinet_power_supply_num and kinet_port or remote_device_type and remote_device_num	dmx:1

- universeKey can be either a string, or an object containing protocol and either index, kinet_power_supply_num and kinet_port or remote_device_type and remote_device_num as received from get_protocols

Returns an object with the same structure as in the HTTP call

Input

Returns the inputs on the queried controller. Show

Lua

```
get_input(idx)
```

Argument	Type	Example
idx	integer	1

Returns the value of the controllers input as a boolean or integer

Example:

```
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)
```

`get_dmx_input(chnl)`

Argument	Type	Example
chnl	integer	1

- chnl is the required channel number

Returns the value of the DMX input at channel chnl as an integer

HTTP

`GET /api/input`

The returned object has the following structure

Property	Return Type	Return Example
gpio	array (of Input objects, on LPC or TPC+EXT)	[{"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},{"input":5,"type":"Contact Closure","value":true},{"input":6,"-type":"Contact Closure","value":true},{"input":7,"type":"Contact Closure","value":true},{"input":8,"type":"Contact Closure","value":true}]
dmxIn	object (DMX Input object, if DMX Input is configured)	[0,0,0,0,0,0,0,0,0,255,255,255...255,0,255]

The Input object has the following properties:

Property	Return Type	Return Example
input	integer	1
type	string ('Analog', 'Digital', 'Contact Closure')	"Contact Closure"
value	integer or bool (depends on type)	true

The DMX Input object has the following properties:

Property	Return Type	Return Example
error	string (if DMX Input is configured but no DMX is received)	"No DMX received"
dmxInFrame	array (of channel values)	[0,0,0,0,0,0,0,0,0,255,255,255...255,0,255]

JavaScript

Not currently available.

Trigger

Returns the triggers in the project. Show

Lua

Not currently available.

HTTP

```
GET /api/trigger
```

The returned object has the following structure

triggers array (of Trigger objects)

The Trigger object has the following properties:

Property	Return Type	Return Example
type	string	"Startup"
num	integer	1
name	string	"Startup"
trigger_text	string	"At startup"
conditions	array (of Condition objects)	[{"text":"Before 12:00:00 every day"}]
actions	array (of Action objects	[{"text":"Start Timeline 1"}]

The Condition and Action objects have the following properties:

Property	Return Type	Return Example
text	string	"Start Timeline 1"

JavaScript

Not currently available.

Lua Variable

Returns the current value of the specified Lua variable. Show

Lua

Not currently available.

HTTP

```
GET /api/lua?variables=luaVariables
```

Argument	Type	Example
luaVariables	string or comma separated list	'myVar' or 'myVar, myVar2, myVar3'

Returns an object containing all the Lua variables requested and their values.

JavaScript

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

Argument	Type	Example
luaVariables	string or array	'myVar' or 'myVar, myVar2, myVar3'

Returns an object containing all the Lua variables requested and their values.

Example:

```
Query.get_lua_variables("myVar", function(lua){
 var value = lua.myVar
}
```

Trigger Variable

Returns the value of a variables from the trigger that ran the script. Show

Lua

```
get_trigger_variable(idx)
```

Argument	Type	Example
idx	integer	1

Returns the trigger variable at idx as a Variant object.

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Resources

Use to locate resources in the controller's memory. Show

Lua

```
get_resource_path(filename)
```

Argument	Type	Example
filename	string	'settings.txt'

Returns a path to the resource filename.

Example:

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

HTTP

Not currently available.

JavaScript

Not currently available.

Content Target

Returns information about a Content Target in the project. Show

Lua

On a VLC

get_content_target(compositionNum)

On a VLC+

```
get_content_target(compositionNum, type)
```

- compositionNum is the usernumber of the composition to return
- type is the type of target within the composition to return (PRIMARY, SECONDARY, OVERLAY_1, OVERLAY_2)

Returns a Content Target object with the following properties:

master_intensity_level	Variant
rotation_offset (VLC+ only)	float
x_position_offset (VLC+ only)	float
y_position_offset (VLC+ only)	float

Example:

```
target = get_content_target(1)

current_level = target.master_intensity_level

target = get_content_target(1,PRIMARY)

current_angle = target.rotation_offset
```

HTTP

Not currently available.

JavaScript

Not currently available.

API Actions

Below are the ways of changing properties or changing output on the controller. show

Start Timeline

Start a timeline in the project Show

Lua

```
get_timeline(timelineNum):start()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "start",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.start_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Start Scene

Start a scene in the project Show

Lua

```
get_scene(sceneNum):start()
```

Argument	Type	Example
sceneNum	integer	1

HTTP

```
POST /api/scene
{
   "action": "start",
   "num": sceneNum
}
```

Argument	Type	Example
sceneNum	integer	1

JavaScript

```
Query.start_scene({ "num": sceneNum}, callback)
```

Argument	Type	Example
sceneNum	integer	1

Release Timeline

Release a timeline in the project Show

Lua

```
get_timeline(timelineNum):release([fade])
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

HTTP

```
POST /api/timeline

{

  "action": "release",

  "num": timelineNum[,

  "fade": fade]

}
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

JavaScript

```
Query.release_timeline({ "num": timelineNum[, "fade": fade]}, callback)
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

Release Scene

Release a scene in the project Show

Lua

```
get_scene(sceneNum):release([fade])
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

HTTP

```
POST /api/scene

{

  "action": "release",

  "num": sceneNum[,
```

```
  "fade": fade]
}
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

JavaScript

```
Query.release_scene({ "num": sceneNum[, "fade": fade]}, callback)
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

Toggle Timeline

Toggle a timeline in the project (if it is running, stop it, and if it is not running, start it) Show

Lua

```
get_timeline(timelineNum):toggle([fade])
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

HTTP

```
POST /api/timeline
{
  "action": "toggle",
  "num": timelineNum[,
  "fade": fade]
}
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

JavaScript

```
Query.toggle_timeline({ "num": timelineNum[, "fade": fade]}, callback)
```

Argument	Type	Example
timelineNum	integer	1
fade	float	2.0

Toggle Scene

Toggle a scene in the project (if it is running, stop it, and if it is not running, start it) Show

Lua

```
get_scene(sceneNum):toggle([fade])
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

HTTP

```
POST /api/scene
{
  "action": "toggle",
  "num": sceneNum[,
  "fade": fade]
}
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

JavaScript

```
Query.toggle_scene({ "num": sceneNum[, "fade": fade]}, callback)
```

Argument	Type	Example
sceneNum	integer	1
fade	float	2.0

Pause Timeline

Pause a timeline in the project Show

Lua

```
get_timeline(timelineNum):pause()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "pause",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.pause_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Resume Timeline

Resume a timeline in the project Show

Lua

```
get_timeline(timelineNum):resume()
```

Argument	Type	Example
timelineNum	integer	1

HTTP

```
POST /api/timeline
{
  "action": "resume",
  "num": timelineNum
}
```

Argument	Type	Example
timelineNum	integer	1

JavaScript

```
Query.resume_timeline({ "num": timelineNum}, callback)
```

Argument	Type	Example
timelineNum	integer	1

Pause All

Pause all timelines in the project Show

Lua

```
pause_all()
```

HTTP

```
POST /api/timeline
{
 "action": "pause"
}
```

JavaScript

```
Query.pause_all(callback)
```

Resume All

Resume all timelines in the project Show

Lua

```
resume_all()
```

HTTP

```
POST /api/timeline

{

 "action": "resume"

}
```

JavaScript

```
Query.resume_all(callback)
```

Release All

Release all timelines, scenes or timelines, scenes and overrides in the project Show

Lua

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

release_all_timelines([fade,] [group])

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

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

HTTP

```
POST /api/release_all

POST /api/timeline

POST /api/scene

{

  "action": "release"[, (not required for release all)

  "group": group][,

  "fade": fade]

}
```

Argument	Type	Example

fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

JavaScript

```
release_all_timelines({["fade": fade]}, callback)
release_all_scenes({["fade": fade]}, callback)
release_all({ ["fade": fade,] ["group": group] }, callback)
```

Argument	Type	Example
fade	float	2.0
group	string ("A","B","C","D") prepend with ! for except (e.g. "!A")	"A"

Set Timeline Rate

Set the rate of a timeline in the project Show

Lua

```
get_timeline(timelineNum):set_rate(rate)
```

Argument	Type	Example
timelineNum	integer	1
rate	float or integer (0-255)	0.1 or 25

HTTP

```
POST /api/timeline
{
  "action": "set_rate",
  "num": timelineNum,
  "rate": rate
}
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

JavaScript

```
Query.set_timeline_rate({"num": timelineNum, "rate": rate }, callback)
```

Argument	Type	Example
timelineNum	integer	1
rate	float or bounded integer	10:100 or 0.1

Set Timeline Position

Set the position of a timeline in the project Show

Lua

```
get_timeline(timelineNum):set_position(position)
```

Argument	Type	Example
timelineNum	integer	1
position	float or integer (0-255)	0.1 or 25

HTTP

```
POST /api/timeline

{

 "action": "set_position",

  "num": timelineNum,

  "position": position

}
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

JavaScript

```
Query.set_timeline_position({"num": timelineNum, "position": position }, call-
back)
```

Argument	Type	Example
timelineNum	integer	1
position	float or bounded integer	10:100 or 0.1

Enqueue Trigger

Fire a trigger in the project Show

Lua

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

Argument	Type	Example
num - the trigger number to enqueue	integer	1
var... - 0 or more variables to pass to the trigger	comma separated variables	1,2,"string"

Example

```
enqueue_trigger(1,1,2,"string")
```

HTTP

```
POST /api/trigger
```

```
{
  "num": num[,
  "var": var...][,
  "conditions": test_conditions]
}
```

Argument	Type	Example
num	integer	1
var...	comma separated variables	1,2,"string"
test_conditions	boolean	true

- num - the trigger number to enqueue
- var... - 0 or more variables to pass to the trigger
- test_conditions - Should the conditions on the trigger be tested?

JavaScript

```
Query.fire_trigger({"num": num[, "var": var...][, "conditions": test_conditions]
}, callback)
```

Argument	Type	Example
num	integer	1
var...	comma separated variables	'1,2,"string"'
test_conditions	boolean	true

- num - the trigger number to enqueue
- var... - 0 or more variables to pass to the trigger. If passing multiple variables, they must be a single string surrounded by single quotes ('), string variables should be surrounded by double quotes (").
- test_conditions - Should the conditions on the trigger be tested?

Run Script

Run a script or parse into the command line on the controller Show

Lua

Not currently available.

HTTP

```
POST /api/cmdline
{
  "input": chunk,
}
```

NOTE: returns "Executed" if successful, or an error string if not

Argument	Type	Example
chunk - the script to parse or run	string	"tl = 1 get_timeline(tl):start()"

JavaScript

```
Query.run_command({ "input": chunk }, callback)
```

NOTE: returns "Executed" if successful, or an error string if not

Argument	Type	Example
chunk - the script to parse or run	string	"tl = 1 get_timeline(tl):start()"

Hardware Reset

Reset the controller (power reboot) Show

Lua

Not currently available.

HTTP

```
POST /api/reset
```

JavaScript

Not currently available.

Master Intensity

Master the intensity of a group or content target (applied as a multiplier to output levels) Show

Lua

Non-VLC

```
get_group(groupNum):set_master_intensity(level[, fade[, delay]])
```

VLC

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

VLC+

```
get_content_target(compositionNum, type):set_master_intensity(level, [fade,
[delay]])
```

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', 'overlay_1', 'overlay_2'
level	float or integer (0-255)	128 or 0.5
fade	float	2.0
delay	float	2.0

Example:

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

HTTP

Non-VLC

```
POST /api/group
```

```
{
 "action": "master_intensity",
    "num": groupNum,
    "level": level,
   ["fade": fade,]
   ["delay": delay]
}
```

VLC/VLC+

```
POST /api/content_target
{
"action": "master_intensity",
"level": level,
["fade": fade,]
["delay": delay,]
"type": type
}
```

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', 'overlay_1', 'overlay_2'
level	float or bounded integer	0.5 or "50:100"
fade	float	2.0
delay	float	2.0

JavaScript

Non-VLC

```
master_intensity({ "num": groupNum, "level": level, ["fade": fade,] ["delay":
delay] }, callback)
```

VLC/VLC+

master_content_target_intensity({ "type":type, "level": level, ["fade": fade,] ["delay": delay] }, callback)

Argument	Type	Example
groupNum	integer	1
compositionNum	Not currently used	
type - of content target	string (from options)	'primary', 'secondary', 'overlay_1', 'overlay_2'
level	float or bounded integer	0.5 or "50:100"
fade	float	2.0
delay	float	2.0

Example:

```
Query.master_intensity({"num":1,"level":"50:100","fade":3) -- master group 1 to 50% (50/100 = 0.5) in 3 seconds)
```

NOTE: Group 0 will master the intensity of the 'All Fixtures' group

Set RGB

Set the Intensity, Red, Green, Blue levels for a fixture or group. Show

Lua

```
get_fixture_override(num)

get_group_override(num)

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

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

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

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

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

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

Argument	Type	Example
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated, "Damped, "Overshoot"

Example:

```
ov = get_fixture_override(1) -- Get fixture 1

ov:set_irgb(255, 255, 0, 0) -- Set the fixture to Red
```

HTTP

```
PUT /api/override

{

   "target": target,

   "num": num,

   ["intensity": intensity,]

   ["red": red,]
```

```
   ["green": green,]
   ["blue": blue,]
   ["temperature": temperature,]
   ["fade": fade,]
   ["path": path]
}
```

Argument	Type	Example
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated, "Damped, "Overshoot"

Javascript

```
set_group_override({ "num": num, ["intensity": intensity,] ["red": red,]
["green": green,] ["blue": blue,] ["temperature": temperature,] ["fade": fade,]
["path": path] }, callback)
```

```
set_fixture_override({ "num": num, ["intensity": intensity,] ["red": red,]
["green": green,] ["blue": blue,] ["temperature": temperature,] ["fade": fade,]
["path": path] }, callback)
```

Argument	Type	Example
num - group or fixture	integer	1
intensity	integer (0-255)	255
red	integer (0-255)	255
green	integer (0-255)	255
blue	integer (0-255)	255
temperature	integer (0-255)	255
fade	float	2.0
path	string (from options)	"Default", "Linear", "Start", "End", "Braked", "Accelerated, "Damped, "Overshoot"

Example:

```
Query.set_fixture_override({ "num": 1, "intensity": 255, "red": 255, "green":
0, "blue": 0});
```

NOTE: Group 0 will set the levels of the 'All Fixtures' group

Clear RGB

Remove any overrides on fixtures or groups. Show

Lua

```
get_fixture_override(num)

get_group_override(num)

   :clear([fade])

clear_all_overrides([fade])
```

Argument	Type	Example
num - group or fixture	integer	1
fade	float	2.0

Example:

```
ov = get_fixture_override(1) -- Get fixture 1

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

HTTP

```
DELETE /api/override

{

   ["target": target,]

   ["num": objectNum,]

   ["fade": fade]

}
```

If num is not included, target is ignored and all overrides are cleared.

Argument	Type	Example
target	string (from options)	"group" or "fixture"
num - group or fixture	integer	1
fade	float	2.0

JavaScript

```
clear_group_overrides({ ["num" :num,] ["fade": fade] }, callback)

clear_fixture_overrides({ ["num" :num,] ["fade": fade] }, callback)

clear_overrides({ ["fade": fade] }, callback)
```

Argument	Type	Example
num	integer	1
fade	float	2.0

Example:

```
Query.clear_overrides({"fade":3})
```

Set Text Slot

Set the value of a text slot used in the project. Show

Lua

```
set_text_slot(name, value)
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

Example:

```
set_text_slot("myTextSlot", "Hello World!")
```

HTTP

```
PUT /api/text_slot

{

   "name": name,

   "value": value

}
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

JavaScript

```
set_text_slot({"name": name, "value": value}, callback)
```

Argument	Type	Example
name	string (matching text slot name)	"myTextSlot"
value	string	"Hello World!"

Example:

```
Query.set_text_slot("name:"myTextSlot", "value":"Hello World!")
```

Set BPS Button LED

Set the effect and intensity on BPS button LEDS. Show

Lua

```
get_bps(num):set_led(button, effect, [intensity], [fade])
```

Argument	Type	Example
num	integer	1
button	integer	1
effect	OFF, ON, SLOW_FLASH, FAST_FLASH, DOUBLE_FLASH, BLINK, PULSE, SINGLE, RAMP_ON, RAMP_OFF	FAST_ FLASH
intensity	integer (1-255)	255
fade	float	0.0

Example:

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

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set TPC Control Value

Set the value on a TPC Slider or Color Picker. Show

Lua

```
set_control_value(name, [index,] value[, emitChange])
```

Argument	Type	Example
name - control Key	string	"slider001"
index - axis of movement (slider has 1, colour picker has 3)	integer (1-3) (default 1)	1
value	integer (0-255)	128
emitChange	boolean (default false)	false

Example:

```
set_control_value("slider001", 1, 128) -- set slider001 to half and don't fire
associated triggers
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set TPC Control State

Set the state on a TPC control. Show

Lua

```
set_control_state(name, state)
```

Argument	Type	Example
name - control Key	string	"slider001"
state - the state name form Interface	string (from options in Interface)	"Green"

Example:

```
set_control_state("slider001", "Green") -- set slider001 to a state called
"Green"
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set TPC Control Caption

Set the caption on a TPC control. Show

Lua

```
set_control_caption(name, caption)
```

Argument	Type	Example
name - control Key	string	"button001"
caption - text to display	string	"On"

Example:

```
set_control_caption("button001", "On") -- set button001's caption to "On"
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Set TPC Page

Change the page on a TPC interface. Show

Lua

```
set_interface_page(number)
```

Argument	Type	Example
number	integer	4

Example:

```
set_interface_page(4) -- change the page on the TPC's interface to page 4
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Disable Page

Disable the TPC touchscreen. Show

Lua

```
set_interface_enabled([enable])
```

Argument	Type	Example
enable	boolean (default true)	true

Example:

```
set_interface_enabled(false) -- disable the TPC's touch screen

set_interface_enabled() -- enable the TPC's touch screen
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Lock TPC

Lock the TPC (requires Lock code to be set within Interface). Show

Lua

```
set_interface_locked([lock])
```

Argument	Type	Example
lock	boolean (default true)	true

Example:

```
set_interface_enabled(false) -- disable the TPC's touch screen

set_interface_enabled() -- enable the TPC's touch screen
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Transition Content Target

Move or rotate a Content Target. Show

Lua

```
get_content_target(compositionNum, type)
  :transition_rotation([offset], [count], [period], [delay], [useShortestPath])
  :transition_x_position([offset], [count], [period], [delay])
  :transition_y_position([offset], [count], [period], [delay])
```

Argument	Type	Example
compositionNum	integer	1
type	PRIMARY, SECONDARY, OVERLAY_1, OVERLAY_2	PRIMARY
offset	integer	10
count	integer	1
period	integer	5
delay	integer	0
useShortestPath	boolean (default false)	false

Example:

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

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Beacon Controller

Beacons the controller (flashes Status LEDs or screen). Show

Lua

Not currently available.

HTTP

```
POST /api/beacon
```

JavaScript

```
toggle_beacon(callback)
```

Example:

```
Query.toggle_beacon()
```

Output to Log

Writes a message to the controller's Log. Show

Lua

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

Argument	Type	Example
level	option (LOG_DEBUG, LOG_TERSE, LOG_NORMAL, LOG_EXTENDED, LOG_VERBOSE, LOG_CRITICAL, default LOG_NORMAL)	LOG_CRITICAL
message	string	"Some message to log."

Example:

log(LOG_CRITICAL, "This is a criticial message!") -- logs the message at Critical log level

log("This is a normal message.") -- logs the message at Normal log level.

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Send variables to Web Interface

Sends data to the web interface in a JSON Object. Show

Lua

```
push_to_web(name, value)
```

Argument	Type	Example

name	string	"myVar"
value	variable	"Some value"

Example:

```
myVar = 15

push_to_web("myVar", myVar) -- will push the object {"myVar": 15}
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Park a Channel

Parks an output channel at a specified level. Show

Lua

```
Universe:park(channel, value)
```

Argument	Type	Example
Universe	Universe object	get_dmx_universe(1)
channel	integer (1-512)	1
value	integer (0-255)	128

Example:

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

HTTP

```
POST /api/channel

{

    "universe": universeKey,

    "channels": channelList,

    "level": level

}
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx	"dmx:1"

	• remoteDeviceType can be rio08, rio44 or rio80	
channelList	comma separated list(1-512)	"1-3,5"
level	integer (0-255)	128

JavaScript

```
park_channel({ "universe": universeKey, "channels": channelList, "level": level
}, callback)
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"
value	integer (0-255)	128

Example:

```
park_channel({ "universe": "dmx:1","channels": 1, "level":128}); // Park chan-
nel 1 of DMX Universe 1 at 128 (50%)
```

Unpark a Channel

Unparks an output channel. Show

Lua

```
Universe:unpark(channel)
```

Argument	Type	Example
Universe	Universe object	get_dmx_universe(1)
channel	integer (1-512)	1

Example:

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

HTTP

```
DELETE /api/channel

{

    "universe": universeKey,

    "channels": channelList

}
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX)	"dmx:1"

	• protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	
channelList	comma separated list(1-512)	"1-3,5"

JavaScript

```
park_channel({ "universe": universeKey, "channels": channelList }, callback)
```

Argument	Type	Example
universeKey	string (in the form protocol:index for DMX, Pathport, sACN and Art-Net, protocol:kinetPowerSupplyNum:kinetPort for KiNET and protocol:remoteDeviceType:remoteDeviceNum for RIO DMX) • protocol can be dmx, pathport, sacn, art-net, kinet or rio-dmx • remoteDeviceType can be rio08, rio44 or rio80	"dmx:1"
channelList	comma separated list(1-512)	"1-3,5"

Example:

```
park_channel({ "universe": "dmx:1","channels": 1}); //Unpark channel 1 of DMX Universe 1 (it will go back to normal output levels)
```

Disable an Output

Unparks an output channel. Show

Lua

```
disable_output(protocol)
```

```
enable_output(protocol)
```

Argument	Type	Example
protocol	option (DMX, PATHPORT, ARTNET, KINET, SACN, DVI, RIO_DMX)	DMX

Example:

disable_output(DMX) -- Disable the DMX output from the controller

HTTP

```
POST /api/output
```

```
{
```

```
    "protocol": protocol,
```

```
    "action": action
```

```
}
```

Argument	Type	Example
protocol	string ("dmx", "pathport", "art-net", "kinet", "sacn", "dvi", "rio-dmx")	"dmx"
action	string ("enable", "disable")	"disable"

JavaScript

```
disable_output({ "protocol": protocol }, callback)
```

```
enable_output({ "protocol": protocol }, callback)
```

Argument	Type	Example
protocol	string ("dmx", "pathport", "art-net", "kinet", "sacn", "dvi", "rio-dmx")	"dmx"

Example:

```
disbale_output({ "protocol": "dmx"}); // Disable the DMX output

enable_output({ "protocol": "art-net"}); // Enable the Art-Net Output
```

Set Timeline Source Bus

Set the time source for a timeline. Show

Lua

```
Timeline:set_default_source()

Timeline:set_timecode_source(timecodeBus[, offset])

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

Argument	Type	Example
Timeline	Timeline Object	get_timeline(1)
timecodeBus	TCODE_1 ... TCODE_6	TCODE_1
audioBus	AUDIO_1 ... AUDIO_4	AUDIO_1
band	integer (0=volume)	0
channel	LEFT, RIGHT or COMBINED	LEFT
peak	boolean (default false)	false

Example:

```
get_timeline(1):set_timecode_source(TCODE_1) -- Set the timecode source of timeline 1 to timecode bus 1
```

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

Enable Timecode Bus

Enables or disables a timecode bus. Show

Lua

```
set_timecode_bus_enabled(bus[, enable])
```

- bus is the timecode bus to enable or disable (TCODE_1 ... TCODE_6)
- enable determines whether the bus should be enabled or disabled (boolean, default true)

HTTP

Not currently available.

Use Run Script or Enqueue Trigger

JavaScript

Not currently available.

Use Run Script or Enqueue Trigger

API Subscriptions

Subscriptions allow data to be pushed to the web interface whenever there is a change within the project. show

Subscribe Timeline Status

Subscribes to changes in the timeline status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_timeline_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	'running'
onstage	boolean	true
position	number (milliseconds)	5000

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_timeline_status(function(t){
        alert(t.num + ": " + t.state)
})
```

Subscribe Scene Status

Subscribes to changes in the scene status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_scene_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
state	string ('none', 'running', 'paused', 'holding_at_end', 'released')	'running'
onstage	boolean	true

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_scene_status(function(s){
        alert(s.num + ": " + s.state)
})
```

Subscribe Group Status

Subscribes to changes in group level, as set by the Master Intensity action (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_group_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
name	string	'Group 1'
level	integer (0-255)	128

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_group_status(function(g){
        alert(g.num + ": " + g.level)
})
```

Subscribe Remote Device Status

Subscribes to changes in Remote Device Online/Offline Status (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_remote_device_status(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
num	number	1
type	string ('RIO 08', 'RIO 44', 'RIO 80','RIO D', 'RIO A', 'BPS')	'Group 1'
online	boolean	true
serial	string (of serial number)	'001001'

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_remote_device_status(function(r){
        alert(r.num + ": " + r.level)
})
```

Subscribe Beacon

Subscribes to Beacons (any change is pushed to the interface). Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_beacon(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
on	boolean	true

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_beacon(function(b){
        if (b.on){
                alert("Beacon Turned On")
        else {
                alert("Beacon Turned Off")
        }
})
```

Subscribe Lua

The receiver for the push_to_web() Lua function. Show

Lua

Not currently available.

HTTP

Not currently available.

JavaScript

```
subscribe_lua(callback)
```

Returns an object with the following properties:

Property	Return type	Return Example
key	as defined by push_to_web()	value

Callback is used to define a function that should be called whenever the data is received

Example:

```
subscribe_lua(function(l){
        key = Object.keys(l)[0]
        value = l.key
        alert(key + ": " + value)
})
```

API Objects

Below are the helper functions and objects in the project. show

Variant

A Lua object that allows a type and range to be associated with a variable. Show

Lua

See here.

HTTP

Not currently available.

JavaScript

Not currently available.

DateTime

A Lua object containing time data. Show

Lua

The DateTime object contains the following properties:

Property	Return Type	Return Example
.year	integer	2017
.month	integer (0-11)	5
.monthday	integer (0-30)	8
.weekday	integer(0-6)	1
.hour	integer(0-23)	13
.minute	integer (0-59)	21
.second	integer (0-59)	46
.utc_timestamp	integer	1494249706
.time_string	string	
.date_string	string	

HTTP

Not currently available.

JavaScript

Not currently available.

Printing an enum

Lua functions to convert integers returned from some functions as text. Show

Lua

```
digital_input_to_string()

button_state_to_string()
```

Examples:

```
log(digital_input_to_string(get_input(1)))

str = button_state_to_string(get_bps(1):get_state(1))
```

HTTP

Not currently available.

JavaScript

Not currently available.

API Authentication

If there is a controller password or Web Interface Access Users setup in the project, then Authorisation will be required to use the HTTP or JavaScript API.

There are two types of Authorisation available, which can be used in different situations:

1. Cookie Authentication
2. Token Authentication

NOTE: Both these authentication methods will expire after 5 mins of inactivity

Cookie Authentication

Cookie Authentication is typically used by the web interface (either Default or Custom), and stores a small file on your computer, which lets the controller know that you are currently signed in.

This authentication is provided through the Default Login page, when using the Default Web Interface, or a Custom Web Interface, without a Custom login page defined.

Custom Login Page

If a Custom Web Interface is being used, then a Custom Login page can be configured. Typically this will be a HTML based page with a form element containing a username and password entry field. The code below can be used to generate these fields, and send the data to the controller's web server to authenticate the user and store the Cookie:

```
<form action="/authorise"
method="POST">

   <input type="text" name="user">

   <input type="password"
   name="password">

   <button
   type="submit">Submit</button>

</form>
```

Token Authentication

Token Authentication is typically used by the HTTP API, where there isn't necessarily a method to enter the username and password manually.

Token Authentication works by the user requesting a Bearer Token from the controller, with the username and password, and this token is then used in future requests.

To request a Bearer Token:

1. Send a HTTP request to http://[ip address]/token in the following format:

```
Method: POST

Headers:

   Content-Type: application/json
```

```
Body:
   {"user":[username], "password":[password]}
```

2. If successful you will receive a response containing the following JSON Object:

```
{
   "access_token": "[some_access_token]",
   "expires_in": 300,
   "token_type": "Bearer"
}
```

3. Subsequent requests will require the following header

```
Authorization: Bearer [some_access_token]
```

Legacy API

The Legacy API documentation is available here.

These APIs can be used if the Controller API Setting is set to Legacy.

Legacy HTTP API

Using the Legacy API, the following interaction between Custom web interfaces and the controller's project can be setup.

Firing Triggers

Custom web pages can trigger the Controller by creating a hyperlink to "/trigger/xyz", where "xyz" is the trigger number, as set in Triggers. Clicking on this hyperlink will fire the numbered trigger, if it exists, but will not cause the page to refresh so there is no need to use Java Script tricks to prevent the page from flickering.

By default, conditions are tested when firing a trigger in this way, but this can be disabled by specifying "conditions=0" in the URL query string. For example, "/trigger/1?conditions=0" will fire trigger 1 regardless of whether its conditions are satisfied.

Firing Triggers with variables

You can capture variables and inject them into the numbered trigger by specifying a "var" field in the query string. The value of this field is expecting a comma-separated list of values, with each value in the format "abc:def" where "abc" is the captured value and "def" is the range of the value (optional). If the value of "abc" is not a number, it is treated as 0. If value of "def" is not a number, it is treated as 255.

For example, the URL "/trigger/1?var=14" will fire trigger number 1 and will inject the value 14. If this trigger had a Start Timeline action that expected a variable to select the timeline, then it would start timeline 14.

In another example, the URL "/trigger/2?var=50:100" will fire trigger number 2 and will inject the value 50. If the variable is used by an action that is expecting a value within a range, for example Set intensity expecting a level between 'off' and 'full', then the injected variable will be treated as 50% since 50:100 = 50%.

You can also inject multiple variables. For example, "/trigger/2?var=50:100,3,4:16" will inject 50 (50%), 3 and 4 (25%).

If you want to inject a string to a variable, for example if you want to set the contents of a text slot, the characters must be bound by double quotes (" or %22 if using character escaping). To include a " in the injected string you must prefix it with a backslash. To include a backslash, write two backslashes. Here are some examples for injecting strings:

/trigger/1?var="hello" -> (hello)

/trigger/1?var=%22hello%22 -> (hello)

/trigger/1?var="hello \"world\"" -> (hello "world")

/trigger/1?var="hello\\world" -> (hello\world)

Anything after the closing " and before the next comma is ignored:

/trigger/1?var="hell"o,"world" -> (hell),(world)

If a closing quote is missing, the remaining string is used:

/trigger/1?var=1,"hello,2 -> (1),(hello,2)

Directly Controlling Outputs

Custom web pages can also directly control Timelines and Scenes without having to create triggers.

The options available are:

Start Timeline, Release Timeline, Pause Timeline, Resume Timeline, Start Scene and Release Scene

To perform these actions you can either type the URL directly into the address bar or set up a hyperlink with the relevant link:

Timelines

URL is of the form <ip_address>/timeline/<action>/<usernumber>

Possible actions are start, release, pause and resume. The release action also has an optional 'fade' field in the query string

Scenes

URL is of the form <ip_address>/scene/<action>/<usernumber>

Possible actions are start and release. The release action also has an optional 'fade' field in the query string

Examples

192.168,0.2/timeline/start/1

192.168.0.2/scene/release/2?fade=3

JavaScript Query Interface

The aim of the javascript library is to provide an abstraction to the web technologies used to communicate with the controller and to provide an api which will not change. This allows us to change the way we communicate with the controller without braking compatibility with existing custom web pages.

Usage

To use the javascript library include this line in the head of the html file.

```
<script src="/default/js/query.js"></script>
```

The Query object will then exist and queries can then be made by calling the approriate method eg

```
Query.get_current_time(callback)
```

callback is a function that is called when the server responds to the request. This is required as all the requests made to the controller are asynchronous. The response sent by the controller is given as the first argument of the callback function and is in JSON format.

Authentication

If an api call is made that the current user does not have sufficient privileges to make, the user will be redirected to the page defined by the AuthFormLoginRequiredLocation directive in the main .htaccess file.

API

`get_current_time(callback)`	Returns the current time and the number of milliseconds that the controller has been on.
`get_timeline_info(callback)`	Returns the name, length, time source and offset for all the timelines in the project.
`get_timeline_status(callback)`	Returns the position and status of all the timelines.
`subscribe_timeline_status(callback)`	Callback is called every time the status of a timeline changes.
`get_scene_status(callback)`	Returns the status of all the scenes in the project.
`subscribe_scene_status(callback)`	Callback is called every time the status of a scene changes.
`get_group_status(callback)`	Returns the status of all the groups in the project.
`subscribe_group_status(callback)`	Callback is called every time the level of a group changes.
`get_remote_device_status(callback)`	Returns the number and type of the remote devices registered to this controller.
`subscribe_remote_device_status (callback)`	A remote device message is pushed every time the status of a remote device changes.
`get_text_slot(callback)`	Returns the name and value for all the text slots in the project.
`set_text_slot(name, value)`	Sets the text slot with name to value
`get_temperature(callback)`	Returns the temperature readings from the controller (TPC or LPC X)
`get_protocols(callback)`	Returns the protocols, universe names and universe ids used by this controller.
`get_output(type, key, callback)`	Returns the current output from a universe. Specify the universe using the type and key
`park_channel(type, key, num, value)`	Parks the channels given by num in the universe specifed by

	type and key to value, the channel can be a single channel, a range of channels ("1-5") or a set of channels ("1,3,5,7") or any combination of those e.g ("1,3,5-10")
`unpark_channel(type, key, num)`	Unparks the channels given by num in the universe specifed by type and key, the channel can be a single channel, a range of channels ("1-5") or a set of channels ("1,3,5,7") or any combination of those e.g ("1,3,5-10")
`fire_trigger(num, variables, test_conditions)`	Fires the trigger given by num and passes any variables
`run_command(cmd)`	Runs the command given by cmd. This can either be parsed by the command line parser, or be a lua command, depending on the settings in designer.
`subscribe_beacon(callback)`	Web interface receives a message whenever the beacon changes.
`toggle_beacon()`	Toggles whether the beacon is on or not.
`subscribe_lua(callback)`	Allows you to pass data to the web interface using the `push_to_web()` Lua script. Example below
`get_system_info(callback)`	Returns the system information
`get_lua_variables(var,callback)`	Returns the variable/s defined in var. This can be a single string (e.g. "running") or an array of strings (e.g. ["running", "active", "myVar"]

Universe Types:

The type is entered as one of the constants below e.g. get_output(DMX,1,callback)

- DMX
- PATHPORT
- ARTNET
- KINET
- SACN
- DVI
- RIO_DMX

Universe Keys:

The Universe key is a JSON object containing the following data:

- DMX: {index : 1} or {index : 2}
- PATHPORT: {index : num} where num = the universe number
- ARTNET: {index : num} where num = the universe number
- KINET: {kinet_port : port, kinet_power_supply_num : num} where port is the port number on the power supply and num is the ID number of the power supply
- SACN: {index : num} where num = the universe number
- RIO_DMX: {remote_device_num : num, remote_device_type : type} where num is the address of the RIO, as set on the address wheel on the front of the RIO and type is one of the the following types:
  - RIO80
  - RIO44
  - RIO08

Examples

Using push_to_web()

The Lua Script has the function push_to_web(name,value).

This allows you to send some data to the web interface as a JSON packet

push_to_web(name,value) sends {name:value} to the interface.

Within the web interface, use the following Javascript to get the information:

```
Query.subscribe_lua(function(x){
  var NAME = Object.keys(x)[0] // Gets the name from the JSON packet and
  stores it as a variable
  var VALUE = x[NAME] // Gets the value from the JSON packet and stores it
  as a variable
  // You will then need to do something with the stored information
})
```

Fire a Trigger with multiple variables

If you have a trigger which is expecting multiple variables, you can add them to the fire_trigger() function as a string:

```
Query.fire_trigger(1, '"var1", 2, 3, "var4"')
```

JavaScript Query Examples

In the examples below, we will query the controller for the required information and display the returned object in the browser's console before displaying the relevant information in a popup.

In the examples, the callback functions have been defined at the time that the query is made, but these can be defined separately, as below:

```
function getTimeRespone(t){
        alert(datetime);
}
Query.get_current_time(getTimeResponse)
```

Function	Example Response	JavaScript

get_current_time(call-back)	{"data":{
	"datetime":"03 Feb 2016 22:41:20",
	"uptime":208171
	},"request":"current_time"}
		Query.get_current_time(
		function(t){
		console.log(t);
		alert(t.datetime);
		alert(t.uptime);
		});

get_timeline_info(call-back)	{"data":{
	"response":[
	{
	"length":"P00H01M00.00s",
	"name":"Timeline 1",
	"num":1,
	"timeOffset":"P00H00M00.00s",
	"timeSource":{
	"type":"internal"
	}
	},
	{
	"length":"P00H00M10.00s",
	"name":"Timeline 2",
	"num":2,
	"timeOffset":"P00H00M00.00s",
	"timeSource":{
	"bus":0,
	"timeFormat":"Unknown",
	"type":"Timecode"
	}
	}
	]
		Query.get_timeline_info(
		function(t){
		console.log(t); // Logging the incoming object
		var output = ""; // Creating a variable to put information in
		for (var i = 0; i < Object.keys (t.response).length; i++) { // Iterate over the object to extract information
		output += t.response[i].num + " - " + t.response[i] ["name"] + "\n"; // Add number and name to the output variable
		};
		alert(output); // Display the output variable
		}
		);

	},"request":"timeline_info"}	

get_timeline_status(call-back)	{"data":{
	"timelines":[
	{
	"active":false,
	"group":"",
	"name":"Timeline 1",
	"num":1,
	"position":0,
	"released":false
	},
	{
	"active":false,
	"group":"",
	"name":"Timeline 2",
	"num":2,
	"position":0,
	"released":false
	},
	{
	"active":false,
	"group":"",
	"name":"Timeline 3",
	"num":3,
	"position":0,
	"released":false
	}
	]
	},"request":"timeline"}
		Query.get_timeline_status(
		function(t){
		console.log(t);
		var output = ""; //
		Creating a variable to
		put information in
		for (var i = 0; i <
		Object.keys
		(t.timelines).length;
		i++) { // Iterate over
		the object to extract
		information
		if (t.timelines
		[i].active) { //
		only add active
		timelines
		output +=
		t.timelines
		[i].num + " - " +
		t.timelines[i]
		["name"] + "\n";
		// Add number and
		name to the
		output variable
		};
		};
		alert(output); //
		Display the output
		variable
		});

get_timeline_status(timeline,callback)	{"data":{
	"timelines":[
	{
	"active":false,
	"group":"",
	"name":"Timeline 1",
	"num":1,
	"position":0,
	"released":false
	}
		Query.get_timeline_status
		(1, function(t){
		console.log(t);
		var output = ""; //
		Creating a variable to
		put information in
		for (var i = 0; i <
		Object.keys
		(t.timelines).length;
		i++) { // Iterate over
		the object to extract
		information
		if (t.timelines
		[i].active) { //
		only add active

	]
	},"request":"timeline"}
		timelines
		output += t.timelines [i].num + " - " + t.timelines[i] ["name"] + "\n"; // Add number and name to the output variable
		};
		};
		alert(output); // Display the output variable
		});

subscribe_ timeline_ status(call-back)	{"broadcast":"timeline",
	"data":{
	"active":{
	"force_poll":false,
	"halted":false,
	"onstage":true,
	"running":true
	},
	"num":1,
	"position":60,
	"released":false
	}
	}
		Query.subscribe_timeline_ status(function(r){
		console.log(r);
		alert("TL" + r.num + " changed");
		})

get_scene_ status(call-back)	{"data":{
	"scenes":[
	{
	"active": true,
	"name":"Scene 1",
	"num":1,
	"onstage":true,
	"released": false
	},
	{
	"active": true,
	"name":"Scene 2",
	"num":2,
	"onstage": false,
	"released": false
		Query.get_scene_status(function(t){
		console.log(t)
		var output = "" // Creating a variable to put information in
		for (var i = 0; i < Object.keys (t.scenes).length; i++) { // Iterate over the object to extract information
		output += t.scenes [i].num + " - " + t.scenes[i]["name"] + "\n" // Add number and name to the output variable
		};
		alert(output) // Display the output

	},
	{
	"active": false,
	"name":"Scene 3",
	"num":3,
	"onstage": false,
	"released": false
	}
	]
	},"request":"scene"}
		variable
		});

subscribe_
scene_status
(callback)
	{"broadcast":"scene",
	"data":{
	"active": true,
	"num":1,
	"onstage": true,
	"released": false
	}
	}
		Query.subscribe_scene_
		status(function(r){
		console.log(r)
		if (r.onstage){
		alert("Scene " +
		r.num + " started")
		} else {
		alert("Scene " +
		r.num + " released")
		}
		})

get_group_
status(call-
back)
	{"data":{
	"groups":[
	{
	"level":100,
	"name":"All Fixtures",
	"num":0
	},
	{
	"level":100,
	"name":"All LED - RGB 8 bit",
	"num":0
	}
	]
	},"request":"group"}
		Query.get_group_status(
		function(t){
		console.log(t)
		var output = "" //
		Creating a variable to
		put information in
		for (var i = 0; i <
		Object.keys
		(t.groups).length; i++)
		{ // Iterate over the
		object to extract
		information
		output += t.groups
		[i].num + " - " +
		t.groups[i]["name"]+
		" - " + t.groups
		[i].level + "\n" //
		Add number and name
		to the output
		variable
		};
		alert(output) //
		Display the output
		variable
		});

subscribe_ group_status	{"broadcast":"group",	Query.subscribe_group_ status(function(r){

(callback)	"data":{
	"level":40,
	"name":"All LED - RGB 8 bit",
	"num":0
	}
	}
		console.log(r);
		alert(r["name"] + " set to " + r.level + "%");
		})

get_remote_device_status (callback)	{"data":{
	"remote_devices":[
	{
	"num":1,
	"online":false,
	"serial":"",
	"type":"RIO 80"
	},
	{
	"num":1,
	"online":true,
	"serial":"",
	"type":"BPS"
	},
	{
	"num":2,
	"online":true,
	"serial":"",
	"type":"BPS"
	}
	]
	},"request":"remote_device"}
		Query.get_remote_device_status(function(t){
		console.log(t)
		var output = "Remote Devices: \n"
		for (var i = 0; i < Object.keys(t.remote_devices).length; i++) {
		output += t.remote_devices[i]["type"] + " " + t.remote_devices[i]["num"] + ":" + t.remote_devices[i]["serial"] + "\n"
		};
		alert(output);
		});

subscribe_remote_device_status (callback)	{"broadcast":"remote_device",
	"data":{
	"num":1,
	"online":true,
	"serial":"011050",
	"type":"RIO 80"
	}
	}
		Query.subscribe_remote_device_status(function(r){
		console.log(r)
		if(r.online){
		alert(r["type"] + " " + r.num + "online")
		} else {
		alert(r["type"] + " " + r.num + "offline")
		}
		})

get_text_slot (callback)	{"data":{
	"text_slots":[
	{
	"name":"text1","value":"Hello"
	},
	{
	"name":"text2","value":"World"
	}
	]
	},"request":"text_slot"}
		Query.get_text_slot(function(t){
		console.log(t);
		var output = ""; // Creating a variable to put information in
		for (var i = 0; i < Object.keys(t.text_ slots).length; i++) { // Iterate over the object to extract information
		output +=t.text_ slots[i]["name"]+ " - " + t.text_slots [i]["value"] + "\n"; // Add number and name to the output variable
		};
		alert(output); // Display the output variable
		});

set_text_slot (name, value)	No Response	Query.set_text_slot("tex- t1","Test") // Sets the value of the text slot "Text1" to "Test"

get_tem- perature (callback)	{"data":{
	"temp":{
	"ambient_temp":24.75
	}
	},"request":"temperature"}
		Query.get_temperature(function(h){
		console.log(h);
		alert(h.temp.ambient_ temp);
		});

get_protocols (callback)	{"data":{
	"outputs":[
	{
	"name":"Art-Net",
	"type":4,
	"universes":[
	{
	"id":1,
	"name":"1"
	},
	{
	"id":2,
	"name":"2"
	},
		Query.get_protocols(function(t){
		console.log(t)
		var output =""
		for (var i = 0; i < Object.keys (t.outputs).length; i++) {
		output += t.outputs [i]["name"] + " in use\n"
		};
		alert(output);
		});

	] }] },"request":"protocol"}	

get_output (id, callback)	{"data":{
	"channels":[
	"0",
	"128",
	"255",
	"50",
	" ", -- This is an unpatched channel
	... 512 entries in total
	" "
	]
	},"request":"output"}
		Query.get_output(2, function(t){
		console.log(t);
		alert("Channel 1 = " + t.channels[0]);
		});

park_channel (type, key, num, value)	No Response	Query.park_channel(DMX, {index:1},1,128) // Set channel 12 on DMX Universe 1 to 128.
		Query.park_channel(DMX, {index:1},"1-10,15",50) // Set channels 1-10 and 15 to 50.

unpark_channel(type, key, num)	No Response	Query.unpark_channel(DMX, {index:1},12) // Release park of channel 12 on DMX universe 1.

fire_trigger (num, variables, test_conditions)	No Response	Query.fire_trigger(1) // Fire trigger 1
		Query.fire_trigger (2,"255,255,128,0",true) // Fire trigger 2, injecting 4 variables (255,255,128,0) and testing the conditions of the trigger.

run_command (cmd)
To use this function, go Project > Web Interface in Designer and check "Parse command line submissions as Lua commands"
	No Response	Query.run_command('get_timeline(1):start()') // Run a Lua script on the controller.
		NOTE: The Script must be contained within quotation marks (' or ").

subscribe_beacon(callback)	{"broadcast":"beacon", "data":{ "on":true	Query.subscribe_beacon (function(r){ console.log(r);

	}
	}
		var output = "";
		if (r.on){
		alert("Beacon On")
		}else{
		alert("Beacon Off")
		}
		})

toggle_beacon()	No Response	Query.toggle_beacon() // The beacon on the controller will be toggled

subscribe_lua()	{name:value} -- As specified in push_to_web() function	Query.subscribe_lua (function(x){
		console.log(x);
		alert(Object.keys(x)[0] + " = " + x[0]);
		})

get_system_info()	{"data":{
	"cf_card_size_kb":1957112,
	"controller_number":1,
	"date":"11 Apr 2016",
	"firmware_version": {"major":2,"minor":1,"point":0},
	"hardware_type":"LPC",
	"network_interface":{
	"gateway":"0.0.0.0",
	"ip_address":"172.30.2.243",
	"subnet_mask":"255.255.0.0"
	},
	"project":{
	"author":"",
	"name":"",
	"upload_date":"2016-04-11T15:12:53",
	"uuid":"{f21f3181-7b71-4f45-bdf3-d96db0e9e7b4}"
	},
	"serial_number":"0100992",
	"sunrise":"06:58:00",
	"sunset":"19:05:00",
	"time":"15:12:55"
	},"request":"system"}
		Query.get_system_info (function(x){
		console.log(x);
		alert(Object.keys(x)[0] + " = " + x[Object.keys (x)[0]]);
		})

get_lua_variables(var, callback)	{"data":{ "count":96, "count2":64	Query.get_lua_variables (["count","count2"], function(x){

	},"request":"lua"}	console.log(x) })

Trigger Programming Guide

Introduction

The Pharos Controllers offer many useful show control capabilities. Frequently it is the ability to cope with the particular show control needs of a project that is the critical factor in selecting a control system.

Show control broadly consists of two tasks. First we need to be able to interface with other devices, which may either be triggering us or be under our control. The Pharos Controller supports most of the core interfaces typically used for show control, either directly on the unit (contact closures, RS232, MIDI, TCP/IP, time and date) or Remote Devices. Within the Triggers screen of the Designer software we can configure the Controller to detect particular triggers and how to respond to them.

Second we need to be able to make decisions. These could be simple choices between two alternatives - perhaps a contact closure needs to trigger a different timeline depending on whether it is during the day or during the night. Within the Triggers screen we support a range of conditions that can be used to quickly implement this sort of logical decision making. We also provide a facility to treat values received on an input as a variable that can be used to alter the behaviour of actions - such as using a number received via RS232 to select a particular timeline.

The standard capabilities offered in the Triggers screen are extensive, but a good show control system has the ability to cope with situations that are anything but standard. Within the Pharos system when things get non-standard then we can use scripting.

Script is a simple programming language that allows users to extend the functionality of the Pharos system themselves. We use a freely available programming language called Lua. Anyone who has ever worked with a programming language will find all the typical tools are available, and it should be straightforward to pick up for those who have not. On top of the core Lua syntax we have added some dedicated Pharos functions that allow scripts to work directly with the capabilities of a Controller.

Not every problem requires script, but there are few show control problems that can't be solved using script where necessary. A few examples of situations where you might want to use script include:

- Making a single contact closure start a different timeline each time
- Make a timeline loop a set number of times and then release
- Track motion sensor activity over a period of time
- Inverting a DMX input before it is used with a Set Intensity action
- Interpreting data from a wind direction sensor
- Using a table of times for high and low tide to control bridge lighting
- Implementing an interactive game for a science museum

We will use some of the situations as examples below.

The Basics

There are a few basic things you need to know straight away. If any of them are not immediately clear then don't worry - there are lots of examples of how to apply them in the following section.

Lua scripts are written as simple text files using any text editor. It is standard practice to use a .lua filename extension though this is not required. These text files can be loaded directly into the Script Editor within Designer.

Comments

It is good practice to include readable comments in your scripts so that you (or anyone else) will be able to easily tell what you were aiming to achieve. In Lua everything after two dashes on a line is treated as a comment.

```
-- This is a comment
This = is + not - a * comment -- but this is!
```

The whole point of comments is that they have no effect on the behaviour of the script. But I am introducing them first so that I can use them within the examples that follow.

Variables

If you want to store a piece of data - whether it is a number, some text or just true or false - then you use a variable. You create a variable simply by giving it a name and using it in your script. A variable can store any type of data just by assigning it.

```
firstVariable = 10 -- assign a number
anotherVariable = "Some text" -- assign a string
```

When you next use these names then they will have the values that you assigned to them:

```
nextVariable = firstVariable + 5 -- value of nextVariable will be 15
```

Note that names are case-sensitive (i.e. capitals matter!), and once you have named a variable once then any time you use the same name you will be referring to the same variable - in programming terms it is *global*. This even applies across different scripts - so you can assign a number to a variable called `bob` in one script and then use the number in another script by referencing `bob`.

One of the most common errors when writing scripts is trying to use a named variable before it has been assigned a value - this will result in an error when the script is run. It is also very easy to use the same name in two different places and not realise that you are actually reusing a single variable. (There is a way of dealing with this for names you want to reuse that we will touch on later.)

Arithmetic

Scripts will often need to do some arithmetic - even if it is something very basic like keeping a counter of how many times it is run:

```
myCount = myCount + 1
```

All of the standard arithmetic operations are available. There is also a library of mathematical functions available should it be required, which includes things like random number generators.

Flow of Control

In most scripts there will be one or more points where you want to make choices. Lua provides four useful structures for this. The most common is `if`, where you can choose which path to take through the script by performing tests.

```
if myNumber < 5 then
    -- first choice
elseif myNumber < 15 and myNumber > 10 then
    -- second choice
else
    -- third choice
end
```

The other control structures all involve blocks of script that need to be repeated a certain number of times. The most straightforward is the `while` loop, which will repeat the enclosed block of script as long as the test at the start is true:

```
myNumber = 10
while myNumber > 0 do
    -- some useful script
    myNumber = myNumber - 1 -- myNumber counts down
end
```

The `repeat until` loop is really exactly the same, but here the test is done at the end of each loop and it will repeat while the test is false.

```
myNumber = 1
maxNumber = 4096
repeat
    -- some useful script
    myNumber = myNumber * 2
until myNumber == maxNumber
```

Here it is worth noting the use of two equal signs == to mean 'is equal to' in a test. This is different from a single equal sign, which is used for assigning values. It is another very common mistake to assign a value when you meant to test if it was equal, and it can be hard to spot because it is valid syntax that will not generate an error. The opposite of == meaning 'is equal to' is ~= meaning 'is not equal to'.

The other control structure is the `for` loop, which has a number of powerful options beyond the scope of what we need here. But it is worth seeing how it can be used to do basic loops in a slightly neater way:

```
for i = 1,10 do
    -- some useful script where i has value 1 to 10
end
```

A final word of caution regarding loops: be careful that you do not write a loop that will never exit! This is all too easy to do by forgetting to increment a counter value that you are using in the test for the loop. If your script has one of these 'infinite loops' then the Controller will get stuck when it runs the script and be reset by the watchdog feature (provided this is enabled). Script is a tool for the grown-ups and it will not protect you from doing silly things - so make sure you test your scripts carefully before leaving them to run.

Tables

Often you will need to store a set of values within a script - these might be a list of timeline numbers or the current states of all the contact closure inputs. Lua allows us to store multiple values within a single named variable and this is called a Table.

A table has to be created before it can be used:

```
firstTable = {} -- creates an empty table
secondTable = { 5,3,9,7 } -- a table with 4 entries
```

You can then access entries within the table by indexing into it - signified by square brackets. The number within the square brackets identified which entry within the table you want to use or modify.

```
x = secondTable[3] -- x now equals 9 (3rd entry)
firstTable[1] = 5 -- entry 1 now has value 5
firstTable[7] = 3 -- entry 7 now has value 3
x = firstTable[1] + firstTable[7] -- x now equals 5 + 3
```

Note that we are allowed to assign values to entries within the table without doing anything special to change the size of the table. We can keep adding elements to the table as needed and Lua will take care of it for us. This makes it possible to write scripts using tables that will work regardless of how many entries there are in the table (e.g. a list of 4 timeline numbers or of 40).

Tables are particularly powerful when used together with the loops we looked at in the previous section. For example if I have a table of numbers and I wanted to find the smallest then I could use the following script:

```
numbers = { 71,93,22,45,16,33,84 }

smallest = 10000 -- initialise with large number
i = 1 -- use to count loops
while numbers[i] do
    if numbers[i] < smallest then
        smallest = numbers[i]
    end
    i = i+1
end
```

This is our first really functional piece of script and there are a couple of things worth noting.

- The first entry in a table is accessed using the number one (i.e. `myTable[1]`). This may seem obvious - but some other programming languages start counting from zero.
- As we increment the variable `i` each time around the loop this means we will be looking at a different entry in the table each time around. The test at the start of my while loop is written to work regardless of how many entries there are in the table. When you use a table entry in a test like this then it will be true as long as the entry has some value (even if the value is zero) and false if there is no value there at all.

Functions

Within script there are a whole range of pre-defined operations that you can call when writing your own scripts. Some of these are provided by the Lua language and are fully described in its documentation. Others have been provided by Pharos to allow you to interact with the Controller from script and are fully described in the manual. They are all called functions and accessed using a similar syntax. For example:

```
x = math.random(1,100)
```

This will assign variable `x` a value that is a random number between 1 and 100. The function `math.random()` is a standard function provided by Lua and we can control its behaviour by passing in an argument - in this case the values 1 and 100 to tell it the range within which we want our random number to fall.

```
t = 5
get_timeline(t):start()
```

`start_timeline` is one of the functions provided by Pharos and it will start the timeline with the number passed in as an argument.

It is also possible to define your own functions as part of script. You might do this if there is a block of script that you know you will need to reuse in a lot of different places. It will be much easier to write the script in one place and then call it from wherever you need it.

```
function diff(a, b)
    if a > b then
        return a - b
    else
        return b - a
    end
end

v1 = 10
v2 = 6
v3 = diff(v1,v2)
```

Note that the script containing the function definition must have been run before we try to call the function. It is often useful to have a script that is run by the Controller startup trigger which defines your functions and creates any tables - other scripts that are run by triggers can make use of those functions and tables.

More information

In this document we have only covered the basic concepts that are needed to understand or write useful scripts for the Controllers. For more extensive information on the Lua language there are two documents, both of which are available online at http://www.lua.org or can be bought as books from Amazon.

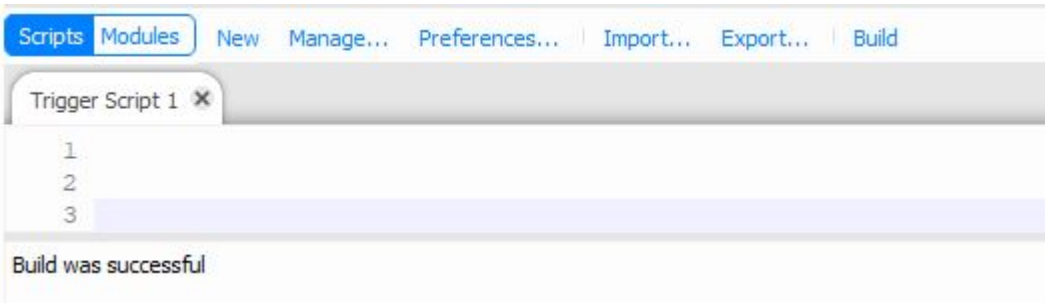
- Lua 5.3 Reference Manual
- Programming in Lua

Lua API (Triggering)

We use a scripting language called Lua, which has been extended to provide functionality specific to the Pharos Controllers. Tutorials and reference manuals for the Lua language can be found at www.lua.org. We will not attempt to document the Lua language here, but just the Pharos specific extensions. Please contact support if you need assistance with preparing a script or if you would like some examples as a starting point.

Lua script editor

The Lua Script Editor allows you to edit scripts from Triggers, Conditions and Actions within Designer. The Script Editor is launched by pressing the Script Editor button on the Trigger Toolbar:



The main area of the editor is the code editor where you enter the source code of the script. The code editor will colour the Lua syntax to aid readability. Standard clipboard shortcuts and undo/redo are supported.

To create a new script for use in Conditions or Actions click New Script.

Scripts can be opened using the Open option and closed with the on the Script Tab.

To import a Lua script from an external file, use Import.

To save a Lua script to a file, use Export.

To compile the script and check for syntax errors, use Build. If there are errors in the script, they will be displayed at the bottom of the window.

Changes to scripts are saved automatically.

Pharos extensions for Trigger Scripts

Syntax

Where a function returns an Object (e.g. get_timeline(num)) additional functions and variables become available. To access a function, add a colon (:) between the functions:

```
get_timeline(1):start() -  This will get the timeline object for timeline 1
and apply the start function to it (starting timeline 1)
```

To access a variable, add a period (.) between the function and the variable:

```
get_timeline(1).is_running - This will return a boolean value indicating
whether timeline 1 is running
```

Variants

A variant is a user object within the Lua Scripting environment which includes information about what type of information is stored.

Variant(val, [range])	creates a Variant, val can be a number or a string. If it is a number then including range will create an Integer Variant, otherwise it is a Real Variant
:is_integer()	returns a boolean (true if the Variant is an integer and false if it isn't)
:is_real()	returns a boolean (true if the Variant is a real number and false if it isn't)
:is_string()	returns a boolean (true if the Variant is a string and false if it isn't)
:is_ip_address()	returns a boolean (true if the Variant is an IP Address and false if it isn't)
.integer	getter/setter for integer values
.range	getter/setter for the range of an integer value
.fraction	getter/setter for an integer converted to a fraction
.real	getter/setter for real values
.string	getter/setter for string values
.ip_address	getter/setter for IP Addresses (a string in dotted decimal form e.g."192.168.1.23")

When a variant is returned by a function, you can run one of the above functions to determine the data type or use a variable to get the data out of the object:

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

For more details, see Variants.

Timeline Management

NOTE: The default fade time for these is 2 seconds unless specified.

stop_all([fade])	stops all timelines and scenes running in the project
stop_all_timelines([fade])	stops all timelines with optional fade time
get_timeline(num)	returns a Timeline object
:start()	starts the timeline
:stop([fade])	stops the timeline with optional fade time
:pause()	pauses the timeline
:resume()	resumes the timeline
:set_default_source()	set which bus is used for the timeline's position
:set_timecode_source(bus [, offset])	set the timecode bus with optional offset
:set_audio_source(bus, band, channel [,peak])	set the time source of the timeline to an audio bus.
.name	returns the timeline's name
.length	returns the timeline's length

.source_bus	
.timecode_format	
.audio_band	returns information about the timeline's time source (depending upon which source is configured)
.audio_channel	
.audio_peak	
.time_offset	
.is_active	returns a boolean value based on whether the timeline is reporting as active
.is_running	returns a boolean value based on whether the timeline is reporting as running
.is_onstage	returns a boolean value based on whether the timeline is reporting as onstage
.is_released	returns a boolean value based on whether the timeline is reporting as released
.is_halted	returns a boolean value based on whether the timeline is reporting as halted or paused
.position	returns the current position of a timeline
.priority	returns the timeline's priority

Timecode Bus Options:

- TCODE_1
- TCODE_2
- TCODE_3
- TCODE_4
- TCODE_5
- TCODE_6

Audio Bus Options:

- AUDIO_1
- AUDIO_2
- AUDIO_3
- AUDIO_4

Audio Channel Options:

- LEFT
- RIGHT
- COMBINED

Scene Management

NOTE: The default fade time for these is 2 seconds unless specified.

stop_all([fade])	stops all timelines and scenes running in the project
stop_all_scenes([fade])	releases all scenes being played back
get_scene(num)	returns a Scene object
:start()	starts the scene
:stop[fade])	stops the scene with optional fade time
:toggle()	toggles the scene
.is_onstage	returns a boolean value based on whether the scene is onstage

Overrides

NOTE: Overrides default to IRGB levels of (0,255,255,255), and if you don't change a level it will be at its default value.

NOTE: The default fade time for these is 2 seconds unless specified.

clear_all_overrides([fade])	Clears all overrides with optional fade time
get_group_override(num)	returns an override object for the specified group number
get_fixture_override(num)	returns an override object for the specified fixture number
:set_irgb(intensity,red, green, blue [, fade[,crossfade]])	overrides the IRGB values with those specified, with an optional fade time and crossfade path
:set_intensity(intensity, [, fade [,crossfade]])	overrides the fixtures intensity with the specified value with an optional fade time and crossfade path
:set_red(red [, fade[,crossfade]])	overrides the red value with the specified value, with an optional fade time and crossfade path
:set_green(green [, fade[,crossfade]])	overrides the green value with the specified value, with an optional fade time and crossfade path
:set_blue(blue [, fade [,crossfade]])	overrides the blue value with the specified value, with an optional fade time and crossfade path
:clear()	clears all overrides from the group/fixture

Crossfade Paths:

- "Linear"
- "NonDim"
- "5% Preheat"
- "10% Preheat"
- "Accelerate"
- "Brake"
- "Damped"
- "Square Law"

Inputs

get_input(input)	returns the state of the local controller's inputs (true = high, false = low)
get_dmx_input(channel)	returns the level of the dmx input channel
get_rio(type, number)	returns a RIO object
:get_input(input)	returns the state of the RIO's inputs (true = high, false = low)

RIO Types:

- RIO80
- RIO44
- RIO08

Lighting Outputs

get_dmx_universe(index)	returns a universe object for the specified universe number
get_artnet_universe(index)	returns a universe object for the specified universe number
get_pathport_universe(index)	returns a universe object for the specified universe number
get_sacn_universe(index)	returns a universe object for the specified universe number
get_kinet_universe(powersupply, port)	returns a universe object for the specified power supply (power supply number in Patch) and port number

:park(channel,value)	parks the specified channel
:unpark(channel)	unparks the specified channel
:get_channel_value(channel)	returns the current value of the specified channel

TPC

set_control_caption(controlName, value)	set the control caption to value = "string"
set_control_state(controlName, name)	set the control state to name ="state name"
set_control_value(controlName[, index], value[,executeChangeTriggers])	set the control value to value with optional index, and optionally execute change triggers
set_interface_page(num)	sets the local TPC to the specified page number
set_interface_locked([lock])	locks the local TPC, lock = boolean, default = true
set_interface_enabled([enable])	enables the local TPC, enable = boolean, default = true

Project Properties

get_current_project()	returns a project object
.name	returns the project name
.author	returns the project author
get_current_controller()	returns a controller object
.number	returns the controller's number
.name	returns the controller's name

Remote Devices

get_bps(num)	returns a BPS object
:get_state(button)	returns the current state of the specified button as an integer:(0 = Pressed, 1 = Held, 2 = Repeat, 3 = Released)
:set_led(button, effect [, intensity] [, fade])	set the specified button to an effect with optional intensity and fade time

Button Effects:

- OFF
- ON
- SLOW_FLASH
- FAST_FLASH
- DOUBLE_FLASH
- BLINK
- PULSE
- SINGLE
- RAMP_ON
- RAMP_OFF

Others

enqueue_trigger(num[,variables])	enqueue trigger num, optionally passing variables
get_resource_path(filename)	returns the path to a file on the SD card (see storing data example)
set_timecode_bus_enabled(bus, [enable])	enables the specified timecode bus, enable = boolean

get_trigger_variable(index)	returns a variant captured by a trigger
log(string)	writes string to the controller's log
push_to_web(name,value)	sends a variable to the web interface in JSON format, eg. { name : value }
set_text_slot(name, text)	sets the value of text slot "name" to "text"
get_text_slot(name)	returns the current text within a text slot
is_controller_online(controller_number)	returns true if the specified controller has been detected online

Timecode Bus Options:

- TCODE_1
- TCODE_2
- TCODE_3
- TCODE_4
- TCODE_5
- TCODE_6

Time Namespace

All the functions below must start with "time."

NOTE: time is a reserved variable name. If you create a Lua variable called time, it will prevent these functions from running

time.get_current_time()	returns a date/time object
time.get_sunrise()	returns a date/time object
time.get_sunset()	returns a date/time object
time.get_civil_dawn()	returns a date/time object
time.get_civil_dusk()	returns a date/time object
time.get_nautical_dawn()	returns a date/time object
time.get_nautical_dusk()	returns a date/time object
time.get_new_moon()	returns a date/time object
time.get_first_quarter()	returns a date/time object
time.get_full_moon()	returns a date/time object
time.get_third_quarter()	returns a date/time object
.year	returns the year of the date/time object
.month	returns the month of the date/time object
.monthday	returns the day of the month of the date/time object
.weekday	returns the day of the week of the date/time object (0 = Sunday, 1 = Monday etc.)
.hour	returns the hour of the date/time object
.minute	returns the minute of the date/time object
.second	returns the second of the date/time object
.utc_timestamp	returns the utc_timestamp of the date/time object
time.is_dst	returns true if the current location is in DST
time.gmt_offset	returns the GMT offset for the controller's location

Example

The script below will from a string in the format hh:mm containing the current time returned by the controller

```
NOW = time.get_current_time().hour .. ":" .. time.get_current_time().minute
```

Hardware namespace

All the functions below must start with "hardware."

NOTE: hardware is a reserved variable name. If you create a Lua variable called hardware, it will prevent these functions from running

hardware.get_last_power_on()	returns a date/time object (see above)
hardware.type	returns the controller's type
hardware.channel_capacity	returns the controller's channel capacity
hardware.serial_number	returns the controller's serial number
hardware.memory_total	returns the controller's total memory
hardware.memory_used	returns the controller's used memory
hardware.memory_free	returns the controller's free memory
hardware.memory_card_size	returns the controller's memory card size
hardware.bootloader_version	returns the controller's bootloader version
hardware.firmware_version	returns the controller's firmware version
hardware.reset_reason	returns the controller's reset reason
hardware.ip_address	returns the controller's IP address
hardware.subnet_mask	returns the controller's subnet mask
hardware.default_gateway	returns the controller's default gateway

Scripting Examples

In this section we will go through a number of practical examples of how script can be used with a Controller. These examples are all based on real projects that are installed and working. They do get progressively more involved, so do not worry if you don't follow the later ones - you will still be able to use script successfully to solve many problems.

If you are working through this document on your own then look out for where I ask a question and rather than reading straight on I recommend stopping and trying to answer it yourself. You will only get truly comfortable with writing scripts by doing it!

Conditions

Running a Trigger 50% of the time

The script below can be used to only run the trigger 50% of the time randomly

```
-- returns true randomly, 50% of the time

return math.random(1,2) == 1
```

Actions

Cycling through different timelines

We are installing a wall of RGB LED fixtures in a children's play area. There is a single large button that the kids are supposed to press. Each time they press it they should get a different colour or effect on the wall.

Each colour or effect would be programmed as a different timeline in Designer. The button will connect to a contact closure and so we will have a single Digital Input trigger. Rather than starting a timeline directly we will instead run the following script:

```
-- which timelines should we cycle through?
timeline = { 22, 14, 24, 16, 15, 17, 21 }
n_timeline = 7

-- on first time of running, initialise index
if not index then
    index = 1
end

-- start the timeline whose number is at entry 'index'

get_timeline(timeline[index]):start()

-- increment index
index = index + 1

-- should we go back to the beginning of the table?
if index > n_timeline then
    index = 1
end
```

How would this change if we wanted each button press to choose a timeline at random rather than cycling through them in order?

```
-- which timelines should we cycle through?
timeline = { 22, 14, 24, 16, 15, 17, 21 }
n_timeline = 7

-- use the random function to set index
index = math.random(1,n_timeline)

-- start the timeline whose number is at entry 'index'
get_timeline(timeline[index]):start()
```

Of course if the timeline selection is truly random then it will sometimes select the same timeline twice in a row. If we wanted to prevent this from happening how could we do it?

```
-- which timelines should we cycle through?
timeline = { 22, 14, 24, 16, 15, 17, 21 }
n_timeline = 7

-- find an index different from the old one
while index == oldIndex do
    -- use the random function to set index
    index = math.random(1,n_timeline)
end

-- store the index for next time round
oldIndex = index

-- start the timeline whose number is at entry 'index'
get_timeline(timeline[index]):start()
```

Stopping a Range of Timelines

We need to stop a large number of timelines in one go, but not all of them.

You can use up to 32 Actions on a Trigger, so if you need to stop more than 32 Timelines at once, you will need to use a script.

You can stop a single timeline from a script with the following:

```
get_timeline(1):stop()
```

This allows you to stop a single timeline from a script, but if you have a large number to stop, adding this for each timeline is a lot of work.

A FOR loop can be used to reduce the amount of scripting required.

```
for i=1,10 do -- run through the values 1-10
    get_timeline(i):stop() -- Stop the timeline defined by i
end
```

This script can be used to run through from 1 to 10 (or a different range by changing the values), and will stop the timeline with those numbers. To make this more useful, you can put it in a function which allows you to call it with any range of timeline numbers.

```
function stop_range(a, b) -- this defines the script as a function with two
variables (a and b)
    for i=a,b do -- a FOR loop which runs through from a to b
        get_timeline(i):stop() -- stop the timeline defined by i
    end
```

```
end

stop_range(1,10) -- call the function with the variables 1 and 10
```

NOTE: It is generally best practice to define the function in a script that is run at startup, and then call the function when it is needed

Make a timeline loop N times

The designer has requested that a particular timeline runs once at sunset on a Monday, but twice at sunset on a Tuesday, three times at sunset on Wednesday, etc. He is planning to keep changing the timeline so does not want to have lots of copies.

There are actually lots of perfectly reasonable ways to solve this using script. Let's assume we have a single astronomical clock trigger that fires at sunset and runs the following script:

```
N = get_current_time().weekday -- 0 is Sunday, 1 is Monday,...

-- we want Sunday to be 7 rather than 0
if N == 0 then
    N = 7
end

get_timeline(1):start()
```

The timeline would be set to loop when it was programmed. We also put a flag on the timeline at the end and make a flag trigger that runs a second script:

```
-- decrement N
N = N - 1

if N == 0 then
    -- release timeline 1 in time 5s
    get_timeline(1):stop(5)
end
```

Note how this works by setting the value of the variable `N` in one script and then using that variable in another script, which is often a useful technique.

I have used two scripts here, but it is possible to do the same job using only one - can you see how?

In this case you would have the sunset trigger start the timeline directly and use the following script on the flag trigger:

```
-- is this the first time round?
if not N or N == 0 then
    N = get_current_time().weekday -- 0 is Sunday, 1 is Monday,...

    -- we want Sunday to be 7 rather than 0
    if N == 0 then
        N = 7
    end
end

-- decrement N
N = N - 1

if N == 0 then
    enqueue_trigger(2) -- runs action on trigger 2
end
```

The trick here is to detect whether it is the first time round the loop - if the Controller has started up today then `N` will have no value and so `not N` will be `true`, otherwise `N` will have been left with the value zero when the script ran yesterday. When we detect it is the first time then we set its initial value in the same way as before.

I have also used a different method to do the timeline release. Rather than calling `get_timeline (num):stop()` directly from the script I am causing trigger number 2 to fire. We can then configure trigger number 2 to have an action that releases the correct timeline. It is sometimes easier to write scripts like this when they are going to be sent out to a customer who does not know how to modify them. In this case all the customer needs to know is to modify the start and release timeline actions in the trigger window if they want to change which timeline is run - they do not need to modify the script.

Storing Data to the Memory Card

In the event that our controller reboots, we want it to start running the timeline that was running prior to the reboot.

The Lua library contains functions that make it possible to read and write files to the device the Lua is running on. This includes reading and writing files on the Controller's Memory card.

```
running = 1
```

This variable will be used to store the number of the timeline that was started most recently,using a Timeline Started Trigger (set to Any) with a Run Script action as below:

```
running = get_trigger_variable(1)
```

Storing data on the memory card involves two steps, writing to the card and reading back from the card.

```
function writeToCard() -- Write the running timelines table to the memory
card
	file = io.open(get_resource_path("timeline.txt"), "w+") -- Open or create a
file (in write mode) called timeline.txt
	if (file ~= nil) then -- Ensure the file has been opened
		io.output(file) -- Set the open file as the default output location for the
io library
		io.write("running = " .. running) -- Write the line "running = [value]" to
the file. The value will be the value of the variable
		io.flush() -- Clear the output buffer
		io.close() -- Close the file
	end
end
```

Whenever the function writeToCard is called it will store the current value of the variable running to the memory card in a file called timeline.txt. The file will be in the format:

running = [value]

This is the syntax for defining a variable in Lua and means that if we run the file on startup, it will set the variable running to be the value stored in the file (with no parsing required)

```
function readFromCard() -- Read the stored running timelines table and start
the timelines specified
	file = io.open(get_resource_path("timeline.txt"),"r") -- Open the file
timeline.txt (in read mode) if it exists
	if file ~= nil then -- Ensure the file has been opened
```

```
                dofile(get_resource_path("timeline.txt")) -- Run the file to set the
variable
       end
       get_timeline(running):start() -- Start the timeline stored in the running
variable
end
```

Whenever the function readFromCard is run, it will find the file called timeline.txt, run it so that the stored variable is set on the controller and then start the relevant timeline

Note: Functions should be placed in a Run Script action at Startup to ensure they are declared. They can then be used at any time within the show file.

Push data to the web interface

If you are using a Custom Web Interface, it is possible to push data to it from the project file e.g. when a TPC Slider is moved. You will then need to set up some Javascript within your custom web interface to read the data in.

If we want to send the level of a TPC slider to the web interface, we would use a TPC Slider Move Trigger set to match the Slider's Key. This would have a Run Script Action attached with the following Lua Script

```
level = get_trigger_variable(1) -- Capture the level set on the slider

push_to_web("slider_level",level) -- creates a JSON packet in the form
{slider_level:level}, where level is the value stored previously
```

Then within the web interface, we need to use the *subscribe_lua()* function to process this data.

Implementing an interactive game for a Science Museum

In an exhibit children are posed questions and have to select answers from an array of numbered buttons. The buttons are large with RGB backlights that are controlled by a Controller to highlight choices and indicate right and wrong answers. Questions are displayed by a slide projector which is under RS232 control from the Controller. The buttons are wired to contact closures on the Controller and on RIOs, so that the Controller can check answers and determine the progress of the game accordingly. The lighting in the rest of the room is designed to mimic a popular TV quiz show to retain the children's interest, with different timelines for each stage of the game.

I am not going to work through this example - but the key point is that it should now be clear to you that a Controller could be used to implement this sort of advanced interactive exhibit with the use of script. Try breaking down the problem into discrete parts and you will find that no individual part of this is difficult - although getting it all to function together reliably would no doubt require a lot of work. The Controller is a viable alternative to custom software running on a PC and has clear advantages in terms of durability and cost.

API Change Log

NOTE: API information is now available via this link.

Changes in API v6 from API v5

Introduced in Designer 2.9

- API v6 has been moved to an online resource.

Changes in API v5 from API v4

Introduced in Designer 2.8

- Added Controller API propagation functionality to certain actions.
- memory_free changed to memory_available.
- get_trigger_number function added.
- vlan_tag property added to Controller.
- is_network_primary property added to Controller.
- dns_servers property added to system namespace.

Changes in API v4 from API v3

Introduced in Designer 2.7

- Add get_network_primary to Controller API.
- Add is_network_primary flag to controller entries in Controller API query response.
- Add description to trigger entries in Trigger HTTP API response.
- Add timeline custom properties to Lua and Web APIs.
- Add type query param to trigger GET request to filter returned triggers on type.
- Add HTTP API endpoint and Query API call for getting information about the current replication.
- get_dmx_input now returns nil if channel is not in range.
- Add NODE protocol cases to Protocol and Output API queries.
- Add NODE protocol cases to Disable Output, Park a Channel and Unpark a Channel actions.
- Add get_log_level, get_syslog_enabled, get_syslog_ip_address, get_ntp_enabled, and get_ntp_ip_address to Controller API.
- Add HTTP API endpoint and Query API call for getting information about the controller's configuration.
- Add set_log_level to Controller API.
- Document HTTP API endpoint for editing the controller's configuration.
- Add Query API call corresponding to the configuration edit endpoint.

Changes in API v3 from API v2

Introduced in Designer 2.6

- Add broadcast_address to System API query.
- Remote Device query
  - Added get_output function for RIO 44 and RIO 08.
- Added Adjustment lua object

Changes in API v2 from API v1

Introduced in Designer 2.5

- Temperature query
  - 'core1_temp' and 'core2_temp' have been replace with 'core_temp'
- Time query
  - Renamed 'utc' to 'local_time'
- Group query
  - 'num' entry is now only present for user created groups
- Remote Device query
  - RIO 44 outputs are now numbered 1-4 rather than 5-8
  - Added set_output function for RIO 44 and RIO 08.
- Protocol query
  - Now indicates if the protocol is disabled
- Output query
  - Now indicates if the protocol of the universe is disabled
- Set TPC page
  - new, optional, 'transition' argument

Issues

As you go through Designer, there will be occasions where something has been configured incorrectly. These will be shown by the Issues icon . These issues, and more will also be shown when you Upload.

If any issues are found, the Issues tab will be opened automatically and a description of each issue will be listed so that you can take corrective action (or you can ignore and proceed if you like):

Issue - '?' will provide the specific details:	Solution
RIO A doesn't provide frequency band ?	Adjust the number of bands on the RIO A
? doesn't support Analog Input	Change the Trigger or Controller/Remote Device type
? doesn't support Digital Input	Change the Trigger or Controller/Remote Device type
? doesn't support DMX Input	
? doesn't support eDMX pass-through	Change the controller
? doesn't support MIDI In	Change the device setting in the Midi Input trigger to a device with MIDI Input
TPC has patch on DMX 1 but hasn't been configured with an EXT or a proxy LPC	Configure an EXT or DMX Proxy, or change the patch to eDMX
? is not a TPC	Change the TPC triggers to match a TPC
? is not found in the project	Add the remote device to the project
? isn't configured to receive DMX Input	Configure the DMX input for the controller
? must be configured as an Audio input device	Change the Mode of the RIO A to Audio
? must be configured to have a held timeout for digital inputs	Set a held timeout in the controller interfaces
? must be configured to have a repeat interval for digital inputs	Set a repeat interval in the controller interfaces
? must be configured to have an EXT	Configure an EXT for the specified controller
A content target hasn't been set	Select a content target for the action
A group hasn't been set	Select a group for the action
A location must be set in Project Properties for waypoint timelines to operate correctly	Set a location for the project
A property hasn't been set	Set the properties for the action
A scene hasn't been set	Select a scene for the action
A timeline hasn't been set	Select a timeline for the action or condition
All inputs on ? should be configured as either digital inputs or contact closures	Configure all inputs on the Controller or Remote Device
Button number hasn't been set	Set a button number on the BPS button condition
Content target transitions are only supported on VLC+	Add a VLC+ to your project, or remove the

Issue - '?' will provide the specific details:	Solution
	action
Controller doesn't have a MIDI Out port	Change the configured controller to a controller with a MIDI Output (LPC)
Controller has configured serial port 2 but the associated device only has one serial port	Change the serial port number in the serial input and/or output to 1
Controller has live video programming but the associated device hasn't been upgraded to support video capture	Remove the Live Video preset, or add a LPC X with video capture card or a VLC/VLC+
Controller has no fixtures patched to it	Patch one or more fixtures to the controller
Controller is configured to receive DMX Input but the associated device doesn't support DMX Input	Change the DMX input to an eDMX source
Controller is configured with an EXT but the associated device has no attached EXT	Connect the TPC to an EXT, ensure EXT is in v2 or uncheck the Configure EXT box
Controller is not a TPC	Change the controller associated with the TPC actions to a TPC
Controller is outputting KiNET so you must ensure that the protocol network interface is on the same network as the patched power supplies	Configure the Network 2 (Protocol) settings to the same network as the KiNet power supplies
Controller isn't associated with a device	Associate the controller with an attached device.
Device number is not set	Set a remote device number
Device type is not set	Set a remote device type
Digital Word is not supported on ?	Change the remote device type
eDMX pass-through for universe ? requires a controller with at least ? DMX ports	
eDMX pass-through requires TPC to be configured with an EXT	Configure the TPC with an EXT
A trigger hasn't been set	Select a trigger in the Enqueue Trigger action
Ethernet bus ? uses port ? which clashes with the internal web server and will prevent correct operation	Change the HTTP port or the ethernet bus port
Input ? must be configured as a digital input or contact closure	Change the input to a Digital or Contact Closure
Input ? must be configured as an analog input	Change the input to Analog
Input ? of ? must be configured as a digital input or contact closure	Change the input on the remote device to a Digital or Contact Closure
Input ? of ? must be configured as an analog input	Change the input on the remote device to Analog
Input number hasn't been set	Set an Input number on the trigger/condition
Interface font is not found	Choose a different font for the interface

Issue - '?' will provide the specific details:	Solution
Location hasn't been set in Project Properties	Set a location in the project properties
No colours have been enabled	Select at least one colour in the Set RGB action/s
No DALI interface is selected	Select a DALI interface
The remote device associated with ? cannot report incorrect DALI power	Interfaces associated with a RIO D4 cannot detect incorrect DALI power
No DMX port is selected	Select a DMX port in the eDMX pass through action
No error has been selected to check	Select an error in the DALI ballast error
No group has been set to release	Select a group in the Release All action
No interface has been chosen	Add an interface to the TPC
No page has been assigned	Select a page in the Set TPC Page action
No script has been assigned	Select a Script in the Run Script Action
No targets have been set to release	Configure the Release All action
Not set to match any data	Set the input string for the trigger
Not set to send any data	Set the string to output
Note On with zero velocity will never match	Reconfigure the MIDI input
Power supply has an invalid IP address	Set a valid IP address for the KiNET Power Supply
Source is not set	Set the Bus for the Ethernet Input
Source media is not found	Replace the source media
The associated device is missing its memory card	Replace the memory card
The associated device is not online	Reassociate the controller, or reconnect the controller
The associated device is on the wrong network	Change the network settings of the controller or computer
The associated device is running the wrong firmware	Reload firmware
The associated device's memory card is corrupt	Replace the controller's memory card
The associated device's memory card is read-only	Unlock the memory card
The chosen audio bus doesn't have band ?	Increase the number of audio buses on the RIO A
The chosen audio bus has no input assigned to it	Change the audio bus, or link a RIO A to the audio bus
The chosen timecode bus has no input assigned to it	Change the timecode bus, or link a timecode source to the timecode bus
The same variable is used to select the BPS number and the button number	Change the variable number to use for the BPS or button
The same variable is used to select the IP address and the port	Change the variable numbers to use for the IP address or port

Issue - '?' will provide the specific details:	Solution
The same variable is used to select the slot name and set the slot value	Change the variable numbers to use for the slot name and slot value
The script has no source	Check the source of the script
The script has not compiled	Check the source of the script for errors
The serial port of ? is set to DMX Out	Change the Serial port to a serial protocol, or choose a different port
The serial port of ? must be set to DMX Input	Change the Serial port to DMX input, or choose a different port
The serial port of ? must be set to RS232 or RS485	Change the Serial port
The slot name has not been set	Set a slot name for the Set Text Slot action
There is only 1 serial port on ?	Change the serial port number to 1

Frequently Asked Questions

Software

Is the free software a cut-down demo version?

No. The free Designer software is the full software package. Downloads and updates can be found at our website.

What are the PC minimum requirements for Designer software?

- Microsoft Windows 7/8/10/11 (64bit)
- macOS 10.13.x (High Sierra) – 13.1.x (Ventura)
- Intel Core i3 processor or above
- 2GB RAM
- 1GB free hard disk space
- 1024×768 screen resolution
- OpenCL 1.2 graphics support (for VLC/VLC+ simulation)
- Network connection (for connecting to Pharos hardware)

Are project files compatible across versions and platforms?

Any project file saved in an earlier version of Designer 2 can be loaded by a later version.

However, a project file saved in a later version of Designer may not be backward compatible as we reserve the right to make structural changes to improve the product.

Project files are compatible between the PC and Mac versions of the software.

Project files created in Designer v1.x.x are incompatible with Designer 2, but can be converted using the Project Migration Tool.

Can I have multiple versions of Designer on my computer?

Yes, provided the installation location (Windows) or application name (Mac) are different.

Can I have Designer v1.x.x and Designer 2 on my computer?

Yes, provided the installation location (Windows) or application name (Mac) are different (this is allowed by default).

What do you do with my software registration information?

We only capture your details so we can inform you of news and software updates, etc. The data is not distributed to 3rd parties nor used for any other purpose.

What documentation is available?

For setup and configuration of the hardware we provide an Installation Guide. This is available as a PDF and a printed version is shipped with every Controller. There is user help available within the software (this document), this is also available as a PDF for printing. See our website for digital copies of all documentation.

How many timelines can I program? How many fixtures, etc?

See system limits and capacities.

How can I tell what DMX levels are being generated?

During programming, when simulating using Output Live, there is a DMX viewer available in the View menu which displays the DMX values generated by Designer. During Controller playback you can use the web interface to view the Controller's DMX output.

Backing up?

Designer can keep a number old versions of the project file when you save. In the Preferences dialog you can set the number of old files to keep. Before saving your project, Designer will rename the project file on disk by adding the current time and date to the file name, such as my_project_bak_2007-04-18_15-58-09.pd2. If you already have the number of specified backups, the oldest backup will be removed from the disk.

The rest is up to you so save early, save often. Use File > Save As to produce manual backups of the project at each important programming milestone.

What are the Pharos Designer file extensions?

*.pd2	Pharos Designer Project file
*.archive.pd2	Pharos Designer Archived project file, contains referenced media & background Layout image so can grow quite large, use this to transfer and archive projects
.upload.pd2	Pharos Designer snapshot file, compressed file that can be uploaded to a controller

Can the project file be retrieved from the Controller(s)?

Yes, there is an option to download the project file from the controller's Web Interface, or from within Network Mode of Designer

How best to Share a Project?

Archive the project (Main Menu> Archive Project) and save the *.archive.pd2 file. This file will include any media that was added to the project file.

How do I programme RS232, RS485 or Ethernet triggers?

Pharos Controllers can send and receive triggers from 3rd party devices over RS232 Serial, RS485 Serial and Ethernet protocols. Programming these all involve entering a string of characters in the trigger parameter pane. These can be entered in three formats; ASCII, Hex & Decimal.

Essentially, the format and string entered depends entirely on the other device. In the Designer trigger/action fields enter exactly the same string as the other device sends, or is expecting to receive. Simply match the string.

Some devices will have a fixed string. For example, the Dynalite button panel sends "1c0164000000ff<c>" if button 1 is pressed. Therefore in an RS485 trigger the same hex characters are entered in the String box.

Other devices may accept customisable messages, allowing meaningful names, descriptions or comments to be entered in an ASCII format (ASCII being standard letters & numbers). In this instance you may create a string in the other device such as "play timeline 6". Then in Designer, the ASCII string entered for the trigger will also be "play timeline 6". The action will be to start timeline 6. For a useful user interface it may be that the user choice and string may be descriptive, such as "red walls, blue ceiling". The trigger string in Designer will be "red walls, blue ceiling", and start the appropriate timeline without the user needing to know what the timeline number is for the desired look.

Variables can also be used within these Serial and Ethernet strings. A message such as "GOxx" is valid, where the xx is replaced with a two digit decimal number which will start the timeline with the corresponding number, rather than having to assign distinct strings to every timeline. In Designer the string would be "GO<2d>" which indicates the system will expect a 2 digit variable. The action would still be Start Timeline, but rather than selecting a specific timeline, set the Variable Index to 1.

A useful tip for programming the Controller is to be connected and utilize the Controller's log. Any messages on the line will appear here, be it in nice friendly ASCII, or other formats. The string can be copied straight out of the log and pasted into the parameters string box - eliminating the possibility for typos, etc. The trigger types can be mapped out in advance with comments and actions, then whatever the other device sends can be grabbed accurately at the time. The log is a great troubleshooting tool for checking what triggers have been received and what actions are fired as a result.

Where are the variables in Actions?

Variables are now hidden behind an Advanced Feature button•••.

Fixtures

What happens if I need a fixture that isn't in the library?

If your fixture isn't in the library within Designer, check the Online Fixture Library.

If you need a fixture that is the same personality as an existing fixture, you can create a Fixture Alias, and if you need a very similar fixture, you can create a Custom Fixture.

Alternatively, contact support with the DMX specification to get a personality written.

I have a fixture with lots of DMX modes, which mode should I use?

The "flat" mode that addresses each cell/element individually with no additional intensity master nor effects/macro channels. Designer has been developed to get the most out of compound fixtures, typically LED battens, tubes and tiles, by driving each element individually, without "help" or complication - virtual intensity channels are created as required and arrays can be constructed onto which effects and media, far more powerful than any built-in function, can be applied and precisely controlled. If in doubt please contact support.

I am trying to output RGB+ fixtures to a DVI Output and I'm not seeing the output I expect, why not?

DVI only supports RGB, fixtures that support more colours than this will not be represented as expected .

An issue should be displayed within Designer to warn of this.

Hardware

What show control interfaces does the LPC 1, 2 & 4 support?

By providing RS232/RS485 serial (including DMX input), MIDI,Ethernet & digital/analog inputs, the LPC can interface with many generic, off-the-shelf products and devices. Via the built-in web interface any browser (PC, Mac, Mobile, etc) can utilise the hyperlinks for triggering and a custom web interface can be designed to provide a user-friendly skin. Remote Devices provide further interfacing options e.g. remote RS232/RS485, SMPTE/EBU timecode, audio input, DALI, MIDI, etc.

Is Pharos RDM compatible?

Yes, the hardware and Designer software supports RDM addressing and mode setting and we will continue to add features in future software versions.

Will I need more memory on the Controller?

The LPC 1, 2 & 4 ships with a 2GB memory card and the LPC X and VLC ships with an SSD. Project data is very memory efficient and so it is unlikely that you will need more memory.

However, heavy use of imported media may necessitate an upgrade and so you can easily replace the card for one of greater capacity, please contact support for recommended cards.

Are there any diagnostic tools?

The LED status indicators on the Controllers serve a dual purpose. In normal operation they indicate system functionality and activity. In an error state, they provide diagnostics, refer to the Installation Guide for details. There is also a recovery tool that guides you through recovery for all controllers.

When should I use reset?

The reset button provides a convenient way to cycle power. It has exactly the same effect. There is no recommendation to reset the Controllers periodically.

Should I keep Controllers in the field up-to-date with the latest firmware?

No, not unless you know that a problem you are having would be solved with an update or you need to change the programming and need the new Designer features. For minor tweaks it's probably best to install the relevant Designer version and do it with that. As a rule, if it's working, leave it be.

What warranty does Pharos offer?

Pharos hardware is warranted for 5 years. Please contact support if you are experiencing any issues.

What user serviceable parts are there in a Controller or Remote Device?

The Pharos product range has been designed for longevity and reliability. There is almost nothing user serviceable apart from a battery for the realtime clock and the DMX driver ICs. Please contact support if you are experiencing any issues.

Standards compliance?

The Pharos product range is manufactured to the highest quality in compliance with international standards, refer to the product's Installation Guide for details.

How do I get a controller out of a Watchdog cycle?

Sometimes programming or a hardware error can cause a controller to rest due to the Watchdog straight after booting. This can normally be corrected by removing the project file from the controller's memory card.

Additional, optional, "Watchdog Protection" on page 282 has been added such that if the controller reboots more than 5 times, within 90 seconds of startup (each time).

Network

How do the Pharos products cope with sharing a network with other, non-lighting devices?

Pharos products can happily sit on any network. They do not broadcast a high volume of management messages and will only listen to Pharos specific messages. However when using a Pharos system with eDMX networks can be slowed down by a large amount of eDMX being transmitted as most protocols utilise network broadcast settings.

Which Network Interface is used for different Ethernet communcations?

Management	**Network 2 (Protocol)/Data**	**Both/Either**
Designer Communications	eDMX Output	Ethernet Input Triggers
Firmware Updates	eDMX Input (for Triggering)	Ethernet Output Actions
Controller to Controller/Remote Device communications	eDMX for Pass Through	IO Module Communications
HTTP API		
Web Interface		

NOTE: If an LPC or TPC is used without Network 2 configured, those communications will use the main Management interface.

What about remote focus units, portable control stations, IR, etc?

With a wireless network access point, any mobile device with wireless capability can be set to browse to the Controller's web interface. The web interface includes status monitoring and logging, hyperlinks of all trigger events and a command line.

The BPS and TPC have learning IR sensors, so they can be programmed to respond to any IR remote.

Is there a way to call up channels for focus?

Yes, on the Control page of the web interface there is a command line that can be setup to allow the user to enter fixture intensity & RGB values. Alternatively, the Output page has a Park option to force a channel to a particular level.

Troubleshooting

The following section lists common problems and their solutions, beginning with an explanation of the Controller's LED indicators:

What are the Controller's LEDs telling me?

LPC Status LEDs

The Pharos logo will illuminate when power is applied to the Controller. The red LEDs on the top/front of the Controller indicate the unit's current status:

- The Active LED illuminates once the boot-up procedure has completed and is indicative of a fully functional unit.
- The Ethernet LED(s) indicates network activity (not network link) while the remaining LEDs indicate communication on the various ports of the Controller.
- The DMX (LPC), Output (TPC, LPC and LPC X) LEDs indicate that a valid project file has been loaded from the memory card and that playback has started.

TPC Status LEDs

The Status LEDs on the TPC are located on the front under the magnetic overlay and on the back next to the Ethernet connection.

- The Power LED illuminates to indicate that power is applied.
- The Active LED illuminates once the boot-up procedure has completed and is indicative of a fully functional unit.
- The Ethernet LED indicates network activity (not network link).
- The Output LED indicates that a show is loaded and eDMX data is being output. (Only on the front)

Error codes

The red status LEDs are used to indicate any boot failures of the Controllers or Devices that prevent the unit from going active.

LPC

The Ethernet, USB and MIDI LEDs will blink; the next three (Serial, Inputs, DMX) indicate the fault type:

- Serial on solid - SD card not present (insert or replace SD card)
- Inputs on solid - cannot mount filesystem on SD card (re-format or replace SD card)
- Serial & Inputs on solid - coprocessor fault (contact our support)
- DMX on solid - invalid hardware error (contact our support)
- Serial & DMX on solid - hardware fault (contact our support)

TPC

The Active LED will be off (Power LED will remain on). Error codes are indicated by double flashing LEDs followed by a 1 second pause, the following combinations indicate the error:

- Ethernet & Output double flashing - memory card missing (insert or replace card)
- Ethernet & Output triple flashing - internal flash error (contact our support)

EXT

Error codes are displayed by a repeating pattern of flashing all four LEDs a number of times in succession, followed by a 1 second pause:

1 flash: Invalid firmware version (reload firmware from Designer)
2 flashes: Invalid device type or serial number
3 flashes: Internal memory test error
4 flashes: Unable to perform factory restore due to corrupt factory firmware
5 flashes: Current firmware is corrupt, no valid firmware versions available to restore
6 flashes: Restored firmware is corrupt

Codes 2 through 6 indicate a hardware error; please consult your distributor, representative or Pharos Support for assistance.

TPS

These codes are outlined below and in all cases the Active LED will be off (Power LED will remain on):

- Ethernet & Output double flashing - memory card missing (insert or replace card)
- Ethernet & Output triple flashing - internal flash error (contact our support)

LPC X rev 2

Error codes are indicated by double flashing of the Ethernet M, Ethernet D, and Serial I/O LEDs, followed by a 1 second pause. The Active LED will also be off.

The bottom four LEDs indicate the error:

- DVI Input on solid -No SSD detected (contact our support)
- Output on solid - Corrupt SSD (recover via USB)
- Overtemp on solid - Invalid Hardware Type (reseat internal dongle)

Should the LPC X be unresponsive you can use the "Controller Recovery" on page 695 or contact our Pharos Support Team for further assistance. Please note that the Recovery Tool may format the SSD and reinstall the firmware. In such a case all project data will be erased and so an upload will be required to restore programming.

LPC X S3

The Error LED in combination with other status LEDs is used to indicate any boot failures that prevent the unit from going active. In the event of a boot error, the Active LED will remain off, and the Error LED will illuminate with the following additional LEDs indicating the specific issue:

- Output -SSD Missing or Failed. Indicates the unit cannot detect or communicate with the SSD.
- Ethernet - SSD Corrupt. Indicates the unit can detect the SSD, but it does not contain a valid boot image.
- Output + Ethernet - Unable to communicate with front panel. Contact Support.
- Remote - Invalid hardware configuration. Contact Support.

Main board errors can usually be resolved by running the LPC X Recovery Tool on a PC. This may format the SSD and reinstall the firmware. In such a case all project data will be erased and so an upload will be required to restore programming.

VLC

Error codes are indicated by double flashing of the Ethernet M, Ethernet D, and Serial I/O LEDs, followed by a 1 second pause. The Active LED will also be off.

The bottom three LEDs indicate the error:

- DVI Input on solid - No SSD detected
- Output on solid - Corrupt SSD - recover from USB
- Overtemp on solid - Invalid hardware type

VLC+

Error codes are indicated by double flashing of the Ethernet M, Ethernet D, and Serial I/O LEDs, followed by a 1 second pause.

The bottom three LEDs indicate the error:

- DVI Input on solid - No SSD detected
- Output on solid - Corrupt SSD - recover from USB
- Overtemp on solid - Invalid hardware type
- DVI and Overtemp on solid - Graphics Driver overheated (contact our support)

EDN

Error codes are displayed by a repeating pattern of flashing all four LEDs a number of times in succession, followed by a 1 second pause:

1 flash: Invalid firmware version (reset to factory default required)
2 flashes: Invalid device type or serial number
3 flashes: Hardware fault with mezzanine card

The second error code indicates a hardware error; please consult your distributor, representative, or Pharos Support for assistance.

RIO G4

Error codes are displayed by a repeating pattern of flashing the LEDs a number of times in succession, followed by a 1 second pause:

1 flash: Invalid firmware version (reset to factory default required)
2 flashes: Invalid device type or serial number

The second error code indicates a hardware error; please consult your distributor, representative, or Pharos Support for assistance.

RIO

Error codes are displayed by a repeating pattern of flashing all four LEDs a number of times in succession, followed by a 1 second pause:

1 flash: Invalid firmware version (reload firmware from Designer)
2 flashes: Invalid device type or serial number
3 flashes: Internal memory test error
4 flashes: Unable to perform factory restore due to corrupt factory firmware
5 flashes: Current firmware is corrupt, no valid firmware versions available to restore
6 flashes: Restored firmware is corrupt

Codes 2 through 6 indicate a hardware error; please consult your distributor, representative or Pharos Support for assistance.

BPS

Error codes are displayed by a repeating pattern of flashing all four LEDs a number of times in succession, followed by a 1 second pause:

1 flash: Invalid firmware version (reload firmware from Designer)
2 flashes: Invalid device type or serial number
3 flashes: Internal memory test error
4 flashes: Unable to perform factory restore due to corrupt factory firmware
5 flashes: Current firmware is corrupt, no valid firmware versions available to restore
6 flashes: Restored firmware is corrupt

Codes 2 through 6 indicate a hardware error; please consult your distributor, representative or Pharos Support for assistance.

Why can't I see the Controller in the Designer Network window?

Presuming that the Controller has successfully booted its firmware (thus its Active LED illuminated) then there is a communication problem between the Controller and the PC running Designer:

Ethernet problems (network)

- Quit and restart Designer again once you're sure that the network is up, use your PC's LAN status tools.
- By default, the Controllers are set to obtain an IP address from a DHCP server, is there one on the network? Put one up and reset the Controller or set a static IP address via USB.
- Is the Controller's firmware compatible with Designer? Update it with the Recovery Procedure.
- Is there a firewall running on your computer or network? This could be blocking the multicast discovery packets.
- Are there any managed switches on the network? Traffic storms from 3rd party devices? Try "pinging" the Controller and other network debugging ploys beyond the scope of this document.

Ethernet problems (one-to-one)

- For a direct, one-to-one connection between a computer and the controller, you can, use a normal network cable because the network interfaces are auto-sensing.
- Quit and restart Designer again once you're sure that the is network up, use your PC's LAN status tools.
- By default, the Controllers are set to obtain an IP address from a DHCP server, is the PC running one? Set a static IP address for both parties, the Controller via USB. If the controller doesn't receive an IP Address from a DHCP Server, it will choose a 169.254.x.x address.
- Is the Controller's firmware compatible with Designer? Update it with the TPC recovery procedure, the LPC recovery procedure or LPC X Recovery Tool.

Incorrect Ethernet cable (CAT5/5E/6) pairing

Not all electrical installers are aware of the subtleties of Ethernet cabling, in particular the correct pairing scheme. While incorrectly paired short cables may work, longer cables almost certainly won't or may exhibit intermittent errors. Note that simple continuity testers will NOT expose an incorrectly paired cable. See this Wikipedia topic for details.

Firmware Issues

For a Controller (or Remote Device) to appear in the network table, it must be on firmware version v2.x.x. If the controller is on firmware version v1.x.x, the Firmware Migration Tool can be used to upgrade the controller to

v2.x.x.

NOTE: The migration tools stopped being updated in v2.5. These tools can be used to perform the migration, but a further firmware upgrade will now be required.

I can see the Controller in Network but it is shown in grey?

Controllers must be on the same Ethernet subnet as the PC running Designer. Select the controller and change it's IP settings accordingly.

I can see the Controller in Network but it is shown in red?

Controllers must be running the same version of firmware as the Designer software. Controllers with incompatible firmware will be highlighted in red. Select the Controller in the network window and press Reload Firmware.

Simulation looks fine but when I upload to the LPC nothing happens?

- Fixtures not patched. Try Output Live or examine the DMX Viewer to debug.
- Output Live left turned on (although a dialog now warns of this when uploading).
- The LPC or TPC hasn't received a valid trigger to commence playback. Use the web interface to check status, examine the log and fire triggers.

Trigger conditions do not work in simulation, why?

Trigger conditions are not tested by the simulator.

Output Live does nothing?

- Fixtures not patched.
- The Output Live Mask has been incorrectly set.

The Controller's playback performance is deteriorating over time, why?

If your project has large numbers of timelines set to Hold or Loop, and these timelines are never explicitly released, then over time they will build up in the background and cause the Controller to struggle. Program your triggers to ensure that such timelines are explicitly released when no longer needed.

Uploading was working OK but now always fails?

The memory card has become corrupt and must be formatted, use network configuration or the web interface.

The controller doesn't load the project, why?

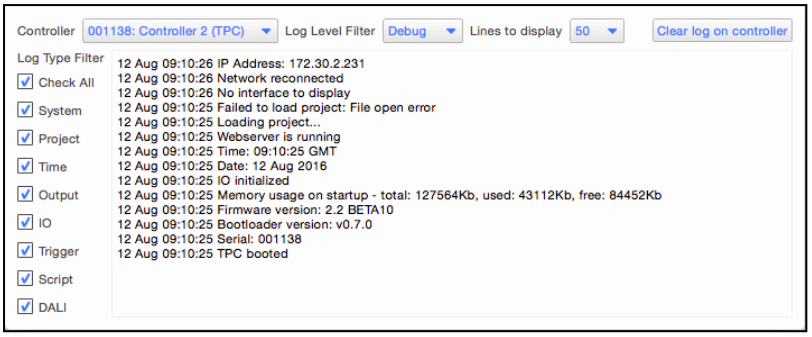
Sometimes a project file can be corrupted during transfer and cause the controller to reset shortly after booting. The controller tracks this and will prevent the project file from being loaded so that the controller can still be accessed. Review the project for potential errors and try to upload over a stable connection.

When I try to Upload I see a list of issues instead?

Designer will check things like triggers and hardware configuration to make sure that there are no inconsistencies. If any issues are found, the Issues tab will be opened automatically and a description of each issue will be listed so that you can take corrective action, see Issues.

Is there a way of seeing what the Controller is doing?

Yes, Controllers generate a log which can be viewed either via the web interface or from within Designer using the Controller Log in the main menu:



See Log viewer for more details.

I have forgotten the Controller's password?

You will need to go on site and gain access to the Controller then contact support for further instructions.

When using DMX In on a LPC, is my DMX line terminated?

No. To terminate the DMX line you should add a 120 Ohm resistor across the positive and negative terminals.

I can connect to the controller, but uploads fail

This is common with remote controllers when the Find function is used and is typically caused by only allowing connections to Port 80 of the controller. Designer also uses port 38008 to upload the project to the controller.

The web interface doesn't populate with data

The web interface uses Web Socket connections (RFC 6455), and some managed networks, internet security software and proxy servers block these connections. Ensure that your network and computer security allow this connection.

I have checked the FAQ and troubleshooting but I'm still stuck?

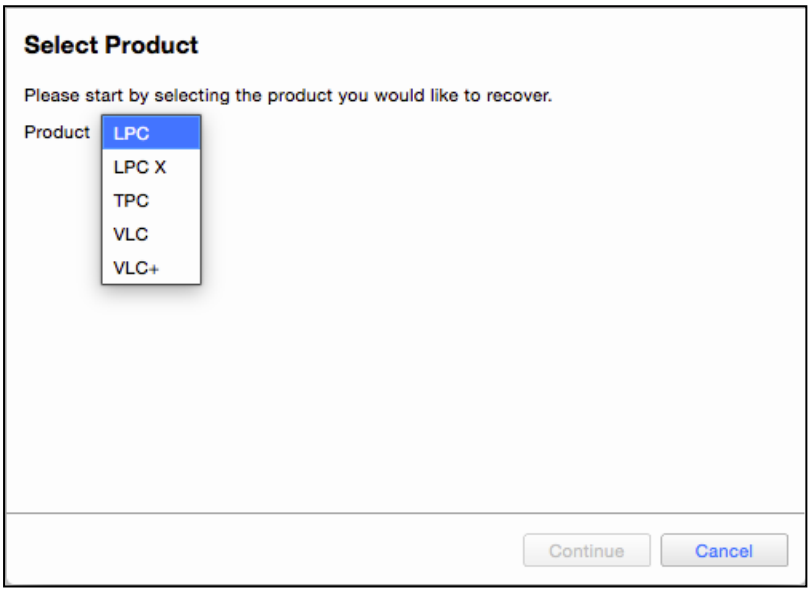
Contact support, please be prepared to send in your project file.

Controller Recovery

The Pharos Recovery Tool can be used to recover a controller if its firmware becomes corrupted, or in the case of the LPC X or VLC/VLC+, if the project file or settings need wiping.

The Controller Recovery Tool is a separate application to Designer. It is located:

- On Windows: in the directory where you installed Designer (by default C:\Program Files\Pharos Designer Controls\Designer 2)
- On macOS: in the "Pharos Designer Utilities" folder of the downloaded application bundle



LPC

The LPC has a built-in failsafe against firmware problems: it stores two versions of firmware. So if one copy of the firmware fails to load, or becomes corrupted due to a loss of power during a firmware reload, the other can be used instead. However, in the event that the LPC will not startup, there is a method to recover the LPC using the memory card.

Please follow these instructions carefully:

1. Remove the memory card from the LPC and insert it into your computer
2. Wipe all files on the memory card (ensure you have made any necessary backups)
3. Locate the firmware folder in the Designer installation location or app bundle (see below)
4. Copy the file lpc.app from the firmware folder to the memory card
5. Reinsert the memory card into the LPC and restart the LPC. The LPC will boot, but will take longer to boot than normal. Please be patient and wait for the Active LED to illuminate continuously
6. Connect to the LPC using Designer and reload the firmware as normal
7. Remove the memory card from the LPC and insert it into your computer again
8. Delete the lpc.app file from the memory card
9. Reinsert the memory card into the LPC and restart the LPC

Locating the controller firmware

Windows

The controller firmware is located in the firmware folder in the Designer installation location, which is in the Program Files folder by default.

macOS

To locate the firmware folder in the app bundle on macOS, please follow these steps:

1. Navigate to the Applications folder located on the Hard Drive, typically named Macintosh HD
2. Locate the Designer application
3. Right-click (or control-click) on it and choose Show Package Contents from the menu that appears
4. Now navigate to Contents/Resources/firmware to find the file lpc.app

TPC

The TPC has a built-in failsafe against firmware problems: it stores two versions of firmware. So if one copy of the firmware fails to load, or becomes corrupted due to a loss of power during a firmware reload, the other can be used instead. However, in the event that the TPC will not startup, there is a method to recover the TPC using the memory card.

Please follow these instructions carefully:

1. Remove the memory card from the TPC and insert it into your computer
2. Wipe all files on the memory card (ensure you have made any necessary backups)
3. Locate the firmware folder in the Designer installation location or app bundle (see below)
4. Copy the file tpc.app from the firmware folder to the memory card
5. Reinsert the memory card into the TPC and restart the TPC. The TPC will boot, but will take longer to boot than normal. Please be patient and wait for the Active LED to illuminate continuously
6. Connect to the TPC using Designer and reload the firmware as normal
7. Remove the memory card from the TPC and insert it into your computer again
8. Delete the tpc.app file from the memory card
9. Reinsert the memory card into the TPC and restart the TPC

Locating the controller firmware

Windows

The controller firmware is located in the firmware folder in the Designer installation location, which is in the Program Files folder by default.

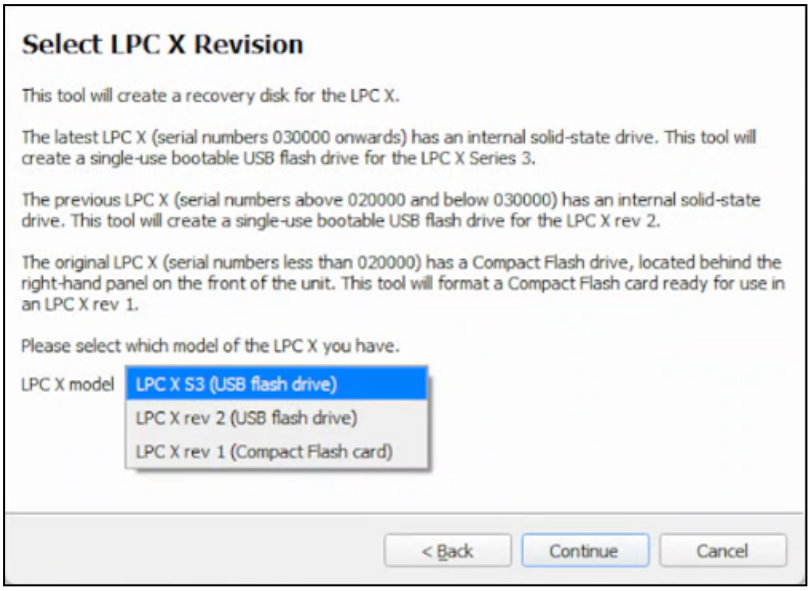
macOS

To locate the firmware folder in the app bundle on macOS, please follow these steps:

1. Navigate to the Applications folder located on the Hard Drive, typically named Macintosh HD
2. Locate the Designer application
3. Right-click (or control-click) on it and choose Show Package Contents from the menu that appears
4. Now navigate to Contents/Resources/firmware to find the file tpc.app

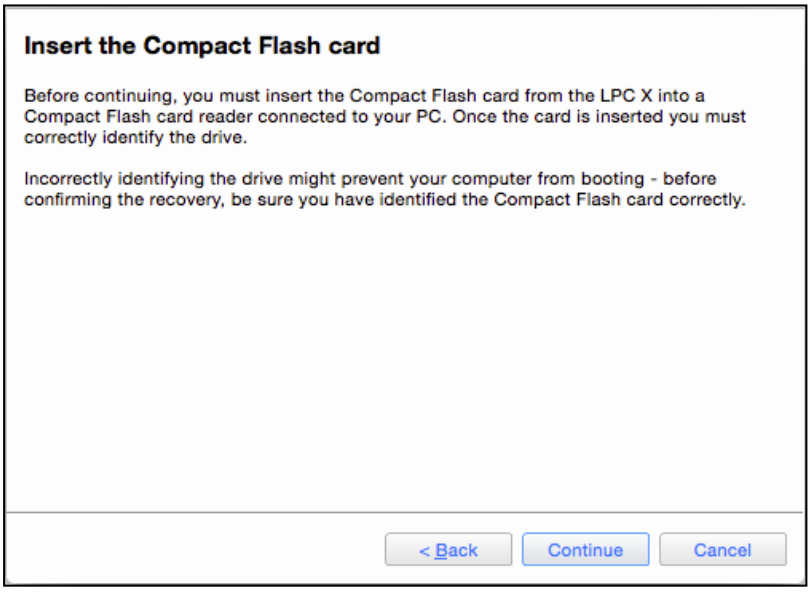
LPC X

The first thing to note with the LPC X is that there are two recovery procedures, with the correct procedure depending on the hardware revision in use. Select your hardware revision below to see the required recovery procedure.

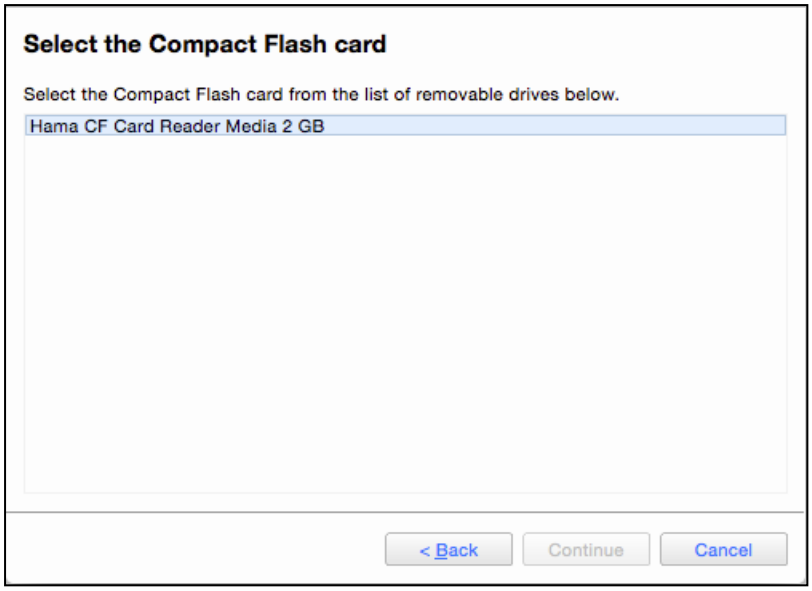


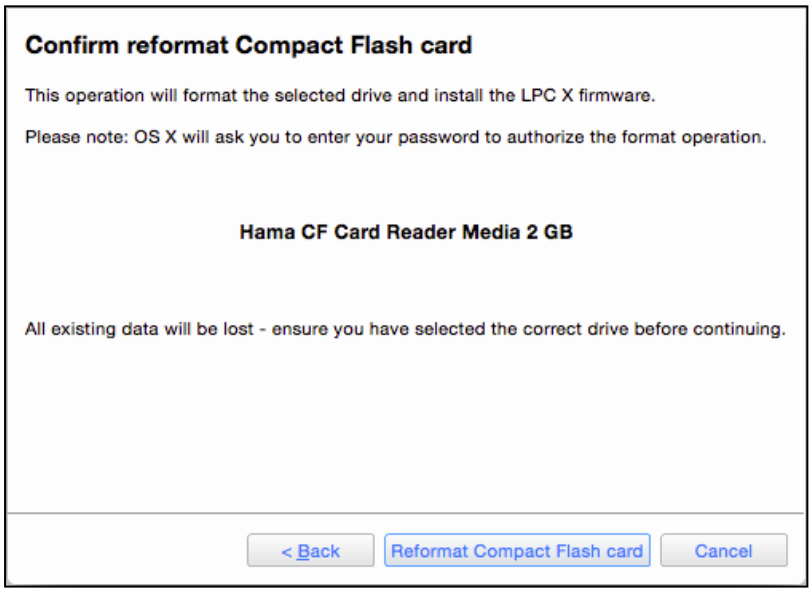
LPC X rev1

The LPC X rev1 contains a CF Card under the right hand front panel. This will need to be removed using a 2.5mm Allen Key, and the CF Card inserted into your computer (or a connected card reader).

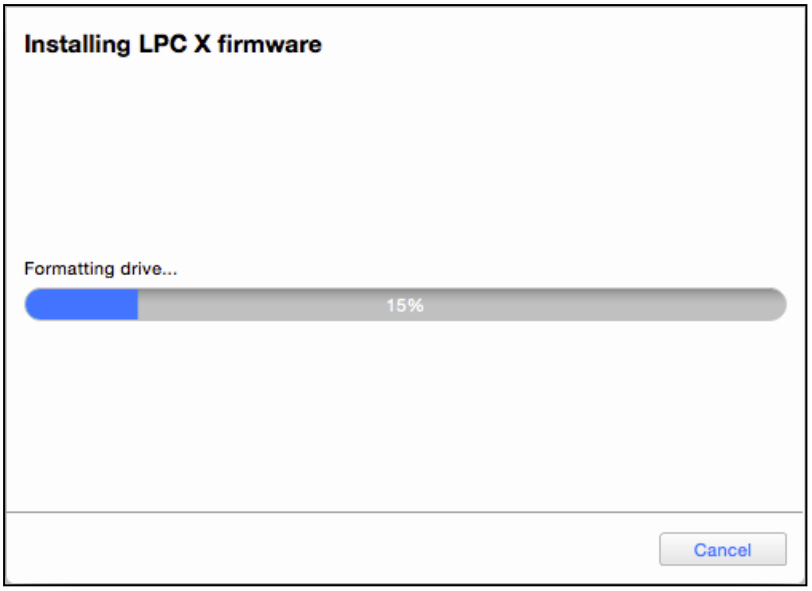


Once you have connected your CF Card you will be prompted to choose it from your connected devices.





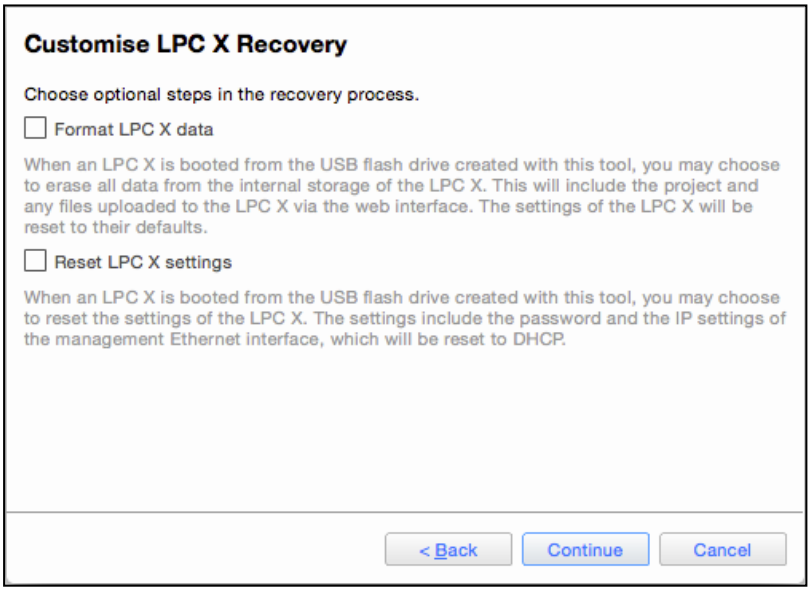
Finally choosing continue will reformat your CF Card with the correct firmware version



LPC X rev2 & LPC X S3

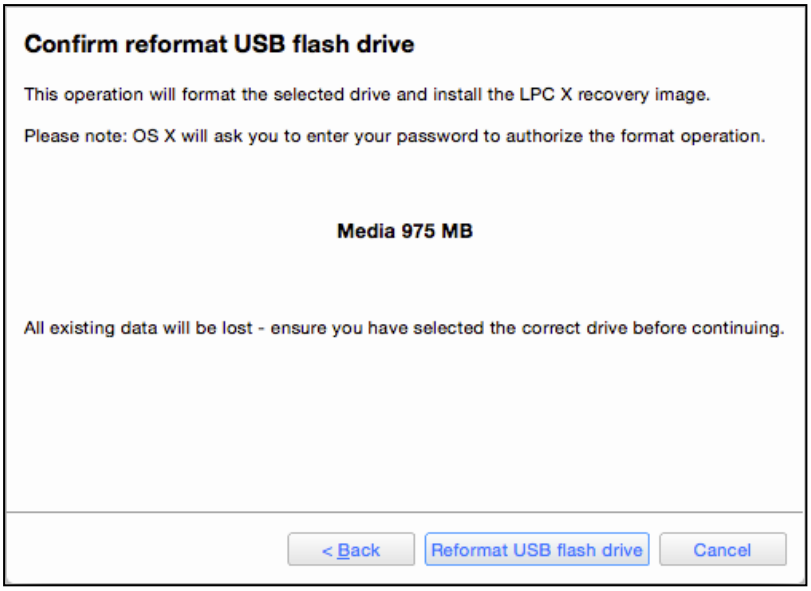
The LPC X rev2 & LPC X S3 can be recovered using a single use bootable USB Memory stick, which is created with the rev2 option in the recovery tool.

You will need to select which of the additional optional steps that you want to perform.

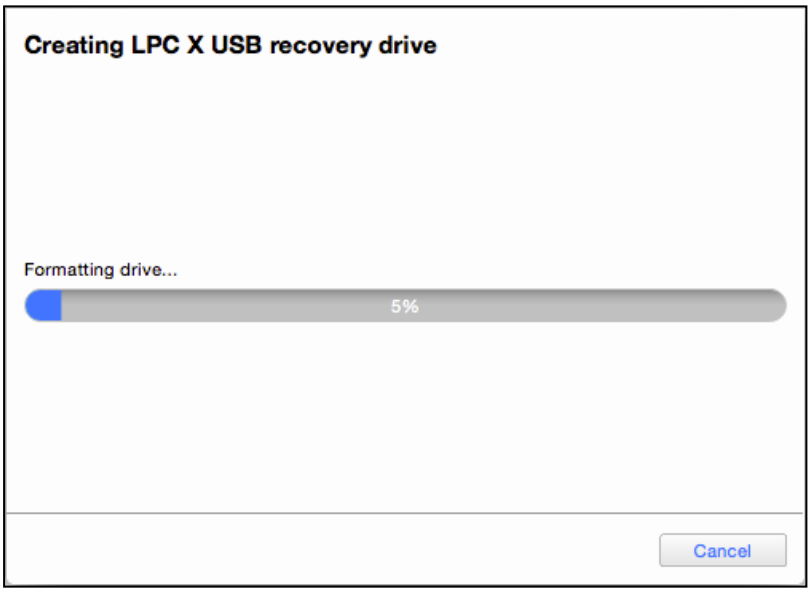


You will need to insert a USB memory stick into your computer. Bear in mind that the memory stick will be reformatted and all data wiped during this process.

Once you have connected your memory stick you will be prompted to choose it from your connected devices.



Finally choosing continue will create the bootable memory stick to recover the firmware on the LPC X and perform any additional steps that you selected earlier.



Additional steps:

- Insert the recovery media into your controller and allow the controller to boot.
- Once booted, reload firmware from Designer using the usual methods from the Netowrk tab.
- After the reload is successful, power off the unit and remove the recovery media.
- The controller should now be ready to be used.

VLC

The VLC is recovered in the same way as the LPC X rev2

VLC+

The VLC is recovered in the same way as the LPC X rev2

Conversion Overview

NOTE: The tools mentioned below are available up to v2.5. Migrating a project or controller can still be done with these Migration Tools but further upgrade will be required.

Projects

Pharos Designer 2 is an upgrade to the existing Pharos Designer v1.12.1, and as such all programming logic and techniques are still valid, however there are new tools and processes to make this programming methodology quicker and simpler:

- Transform Tools
- Project Customisation Tool
- Trigger Filtering

If you want to upgrade an installation from v1.x.x to v2, or you are working on a project in v1.x.x and want to continue working in v2, there is a tool which can be used to convert the v1.x.x file to a v2 file. See Migration Tools for more information.

Hardware

Designer 2 can work with all Pharos hardware, except:

- LPC Rev 1 (Serial numbers lower than 006xxx)
- AVC

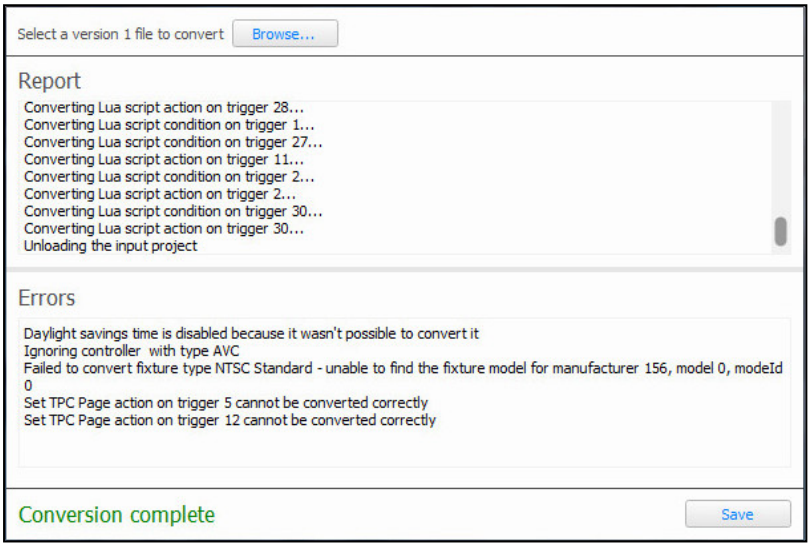
All other Pharos hardware can be used with Designer 2 and the firmware can be upgraded from v1.x.x to v2 using the Firmware Migration Tool . See Migration Tools for more information.

Migration Tools

In the event that you are working on a project which has previously been programmed in a v1.x.x release of Designer, there are some Migration Tools available to convert the v1.x.x project file and controller firmware to 2.5.x.

These Migration Tools are separate from Designer and must be downloaded and installed separately to Designer 2.

Project Migration Tool

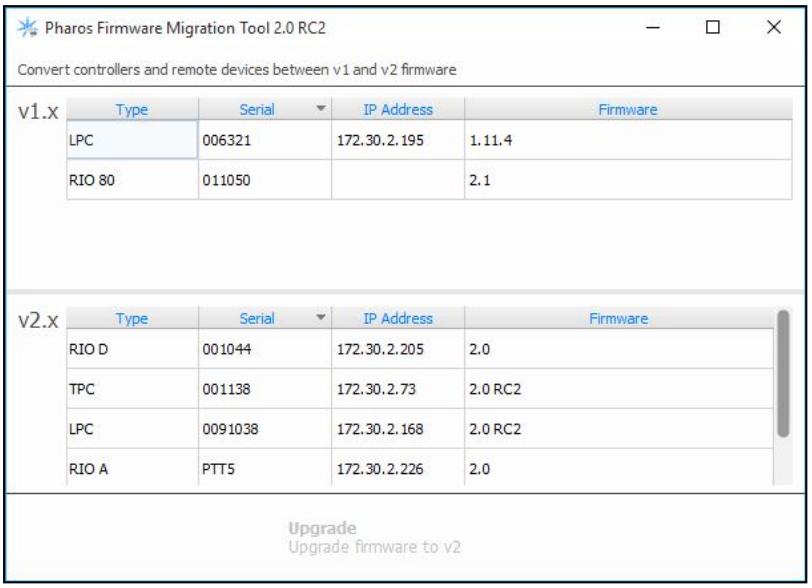


The Project Migration Tool can import a v1.x.x file and convert it to work with Designer 2.5.x.

To use this tool:

- Browse to the v1.x.x file with the v1 Browse button
- Select the output location for the v2 file with the Browse button
- Press the Convert button
- The Report section will display the progress of the conversion and the Errors section will display any potential issues with the converted project file
- The Result of the conversion will be displayed at the bottom (Success or Failure)

Firmware Migration Tool



The Firmware Migration Tool can be used to upgrade a controller with v1.x.x firmware to 2.5.x or to downgrade a controller with 2.5.x firmware to v1.12.

To use this tool:

- Select the controller or remote device that you want to convert (v1 or v2)
- Click the button at the bottom (Upgrade/Downgrade) and the firmware will be converted.

If a controller is displayed in grey, it cannot be converted either because it is on the wrong IP range, or the controller type is not supported by Designer 2.5

Special Considerations

If you are trying to migrate a TPC+EXT:

- The TPC should NOT have a project file loaded.
- The TPC must be on a v2.x.x firmware to upgrade or downgrade the EXT

To migrate a remote device, the Remote Device must not be actively connected to a controller (meaning the Active light should be flashing).

What's Changed from v1.x.x to v2.x.x

General

- Multi-screen support – pop out any tab into its own window
- Multiple instances of the application can be run at the same time with different projects
- Open multiple projects – copy and paste between projects
- Multi-level Undo/Redo
- Auto-save – your work is stored to disk as you go
- New Tab names and re-ordered to follow typical project workflow
- Advanced features are hidden until needed to simplify workflow
- Issues with your project configuration are automatically detected and reported

Project

- New Project wizard allowing quick setup of controllers and features.
- New view for managing projects and their properties
- Projects created in version 1 can be migrated to version 2 using a standalone tool
- Project files can be downloaded from Controllers and opened in Designer
- New 'Import Object' feature making importing fixture and pixel matrix layouts, KiNET power supplies and your patch from any delimited file type
- New 'Export Object' feature allows you to export fixture positions and pixel matrix layouts, KiNET power supplies and your patch to CSV files for manipulation in spreadsheet tools such as MS Excel
- User can define custom properties for fixtures, layouts and timelines that will be shown in reports to help manage other project data

Plan

- Multiple fixture layouts within a single project – simpler to model different areas of an installation or different views of a 3D structure
- Fixture instances – allow a single fixture to appear multiple times on your layouts
- A fixture does not need to be on a layout to still be in your project
- New automatically generated 'All Fixtures' group in the fixture browser
- Searchable fixture library – easier to find the fixture you need
- Folders for Recent and Used fixture types in the fixture library
- Online fixture library – wider range of fixtures can be downloaded from our servers
- Fast single-click placement of new fixtures – no more drag-and-drop to add fixtures
- Change Fixture Type function supported
- Create Custom Type From allows custom fixtures to be created from fixtures in library
- New Transform Tools allow selected fixtures to be aligned, rotated and redistributed
- New 'Snap to Fixtures' toggle
- Fixture groups can now be numbered and used in scripts
- New Stretch and Fill setting for background images
- Holding shift while doing a lasso selection allows fixture position to determine group order
- Press Escape to clear fixture selection – press it again to recall last selection

Patch

- Add the universes you are going to use in the project rather than having them all present
- Offset patching so a group of fixtures can be patched with custom spacing
- Two universes of Ethernet DMX pass-through to local DMX Output Ports

Mapping

- Importing media no longer requires Quicktime to be installed
- Media files can be organised with folders and searched
- Media files can be previewed in a built in media player

Scene

- New “Scene” tab – replaces the Movers tab and as well as controlling moving lights this also offers a new way to create static colour looks for regular fixtures
- Folder structure for scenes

Timeline

- Improved management of timelines
- Tabs for fast switching between the timelines you are working on
- Fast single-click placement of presets – hold Ctrl or Cmd if you want to add multiple
- Modify preset settings before or after you place on the timeline
- Favourite colours can now more easily be saved and managed
- Multi-element copy and paste
- After changes programming rebuilds in the background to make the user interface more responsive
- A timeline can be given one of 4 group designations and timelines can be acted upon by group in triggers

Interface Editor

- The touch panel Interface Editor is now an integrated part of the Designer software
- Have multiple layouts for each TPC open at once
- Can copy and paste entire pages of an interface

Triggers

- Complete rework of the triggers user interface and how you add or edit triggers
- Copy/paste conditions or actions into other triggers
- Edit multiple actions/conditions at the same time
- Filter triggers by type or tag them with group designation
- Script editor moved to its own area and ability to manage scripts independently
- Set RGB actions can now also set intensity
- Set RGB actions can now be given a playback priority
- Groups can now be selected by number in Set/Clear RGB actions
- Improved handling for variables when passed into and out of scripts

Simulate

- Tabbed environment to see simulation on any layout
- Audio track selection and sync’d playback for listening to an audio track while simulating.
- Output Live now allows the user to selectively take control of fixtures being programmed so that other parts of the project continue to run uninterrupted

Network

- Network tab can be used to configure Controllers without an open project
- Projects can be downloaded from a Controller from within Designer

- Discover controllers by IP address allows user to enter IP address of controllers that are not on the local network
- Connect to controllers over USB behaves the same way as Ethernet

Web Interface

- New improved layout
- Projects can be uploaded to or downloaded from controllers using their web interface
- Project status shows scenes and timelines
- Log filtering options with more than 5 times the log information stored
- User access control by view
- New file manager for uploading additional project files
- New network view for an easy overview of project Controllers and Remote Device status
- Formatting (CSS) and images of the standard web interface can be customised
- Simpler management of custom web interfaces

Script Conversion

The PharosProject Migration Tool can be used to convert a v1.x file into a v2.x file, and this should convert any scripting within projects to the updated Lua API, however the conversion table below can be used when writing new scripting from scratch:

V1.X.X	V2.X.X	Notes
realtime.XXX	time.get_current_time().XXX	
sunrise.XXX	time.get_sunrise().XXX	
sunset.XXX	time.get_sunset().XXX	
civil_dawn.XXX	time.get_civil_dawn().XXX	
civil_dusk.XXX	time.get_civil_dusk().XXX	
nautical_dawn.XXX	time.get_nautical_dawn().XXX	
nautical_dusk.XXX	time.get_nautical_dusk().XXX	
digital[index]	get_input(index)	Returns digital and analogue values
DMXIN[channel]	get_dmx_input(channel)	
get_controller_number()	get_current_controller().number	
set_timecode_source_enabled (source,enabled)	set_timecode_bus_enabled (source,enabled)	
start_timeline(num)	get_timeline(num):start()	
stop_timeline(num,time)	get_timeline(num):stop(time)	
halt_timeline(num)	get_timeline(num):pause()	
resume_timeline(num)	get_timeilne(num):resume()	
set_timecode_source(num,source, offset)	get_timeline(num):set_timecode_ source(source,offset)	
set_timecode_source(num,source, band, channel, peak)	get_timeline(num):set_audio_source (source, band, channel, peak)	
is_timeline_running(num)	get_timeline(num).is_running	
is_timeline_onstage(num)	get_timeline(num).is_onstage	
stop_all()	stop_all_timelines(fade)	fade is optional
set_intensity(fixture,value, time)	get_fixture_override(fixture):set_ intensity(value,time)	
set_red(fixture, value, time)	get_fixture_override(fixture):set_red (value, time)	
set_green(fixture, value, time)	get_fixture_override(fixture):set_ green(value, time)	
set_blue(fixture, value, time)	get_fixture_override(fixture):get_blue (value, time)	

V1.X.X	V2.X.X	Notes
clear_fixture(fixture, time)	get_fixture_override(fixture):clear()	
clear_all(time)	clear_all_overrides(time)	
get_dmxout(universe)	get_dmx_universe(universe)	if universe is 1 or 2
get_dmxout(ARTNET + universe)	get_artnet_universe(universe)	
get_dmxout(PATHPORT + universe)	get_pathport_universe(universe)	
get_dmxout(SACN + universe)	get_sacn_universe(universe)	
get_dmxout(get_kinet_universe (powerSupplyNum, portNum))	get_kinet_universe(powerSupplyNum, portNum)	
DMXOUT[channel]	get_XXX_universe(universe):get_channel_value(channel)	DMXOUT is object returned from get_dmxout(universe)
park(universe, channel, value)	get_XXX_universe(universe):park (channel, value)	
unpark(universe, channel)	get_XXX_universe(universe):unpark (channel)	
rio[input]	rio:get_input(input)	rio is object returned by get_rio
bps:set_LED(button,effect, intensity, fade)	bps:set_led(button,effect, intensity, fade)	
variable[index]	get_trigger_variable(index)	

Please Note: These are only the functions that have changed between V1.x and V2.x. There are also newly added functions which can be used to provide additional functionality which wasn't previously available. These can be found here.

Software Release Notes

Release Notes

These are provided in the About tab in the Project Mode of Designer.

Software Licences

GPL

Portions of this software are licensed under the GNU General Public License version 2. The license is available in the About section of the Project Mode of Pharos Designer.

To obtain this software either visit www.carallon.com or send a stamped self-addressed envelope containing a blank CD or USB memory stick to:

GPL Compliance,
Carallon Limited,
272 Gunnersbury Avenue,
Chiswick,
London, W4 5QB
England

System Limits & Capacities

Pharos Designer imposes the following project limits which can not be exceeded:

General Limitations

Timelines	1000	
Custom Presets	256	
Media Presets	256	Imported media clips
Fonts	128	
Scenes	1000	
Folders (for Scenes, Media, Custom Presets)	256	
Triggers	1024	
Conditions Per Trigger	32	
Actions Per Trigger	32	
Trigger Scripts	256	
IO Modules	256	
IO Module Instances	32768	
LPC Family Controllers	40	
VLC Family Controllers	10	
Remote Devices in project	200	Total number of remote devices in a single project. Remote Devices are RIO, EDN, TPS and BPS.
Remote Devices	A TPC or an LPC 1, 2 or 4 can support up to 64 remote devices.	Per controller per Designer project.
	An LPC X can support up to 100 remote devices.	
	A VLC/VLC+ can support up to 100 remote devices	
Install Replications	100	
DALI		Please see DALI Interfaces for "DALI" on page 108 limitations
KiNET power supplies	10000	
Timecode Buses	12	MIDI or linear (SMPTE/EBU) timecode sources
Network Buses	5	Ethernet trigger sources eg. UDP
Audio Buses	4	Audio trigger sources
Web Interface Connections	6	Connections to the controller's web server (each tab in a browser, or separate device counts towards this limit)

Patch Universes	10000	Total number of patched universes in the project
Frame Arrays	1000	Instances of media, perlin noise, starfield, text and custom presets deployed on timelines

LPC Family Limitations

Layouts	64	
Layout Size (pixels)	8192x8192	
Fixture Groups	2000	
Fixtures	40000	Discrete or compound fixtures
Fixture Elements	60000	Elements within compound fixtures eg. 18 per James Thomas Pixeline 1044
Pixel Matrices	256	
Pixel Matrix Size (Pixels)	4096 x 4096	
LPC Layers (Transparency)	4	Exceeding this will remove the first layer so that only 4 are running at once
LPC X Video Input Resolution	Max 1920x1080p30	Maximum video input resolution (DVI/DisplayPort Input)
Touch Device Pages in Project	10240	Total number of Touch Device pages within the project
Touch Device Interfaces in Project	40	
Maximum SD Card Size	8Gb	
Maximum ARP table size (LPC/TPC)	500 devices	Devices that add to the ARP table include Pharos Remote Devices, Art-Net nodes, KiNET Power Supplies, etc.; any device with an IP address that the Controller must talk to.
Maximum ARP table size (LPC X)	1000 devices	

VLC Family Limitations

VLC/VLC+ Layouts Per Controller	1	Each VLC Controller can only have 1 layout.
VLC Layout Size (pixels)	16384x16384	Layout and fixture library scale is 1cm:1pixel (0.394":1pixel)
VLC Fixture Limit	256,000	Each VLC within a project can have up to 256,000 fixture elements on its layout
VLC+ Fixture Limit	512,000	Each VLC+ within a project can have up to 512,000 fixture elements on its layout
VLC/VLC+ Group Limit	2000	VLC groups are designed purely for patching and cannot be used for programming.

VLC/VLC+ Fixtures per Group Limit	10000	
VLC/VLC+ Compositions	100 per project	
VLC/VLC+ Content Target Size	Max area 2,073,600 pixels	1920x1080 by default. The longest x narrowest the content target can be is 16384x126
VLC Players	2	This is our suggested limit, as using more players will prevent fading from one Timeline to another.
VLC+ HD Players	4	
VLC Layers (Transparency)	4	Exceeding this will remove enough layers from the bottom of the LTP stack so that only the given amount are running at once. This allows fading from one timeline using the full player count (mentioned above) to another.
VLC+ Layers (Transparency)	8	
VLC+ Adjustment Targets	8	
VLC/VLC+ Video Input Resolution	Max 1920x1080p60	Maximum video input resolution (DVI Input)
Maximum ARP table size VLC/VLC+	2000 devices	Devices that add to the ARP table include Pharos Remote Devices, Art-Net nodes, KiNET Power Supplies, etc.; any device with an IP address that the Controller must talk to.

As you can see from the above limits, the Pharos control system can scale to an impressive size that rivals even state-of-the-art lighting consoles.

For very large projects, or projects where some of the above limitations are restrictive, please contact support to discuss your requirements in advance.

Best Practices

Just like any other computational device, Controllers have a finite amount of resources available to them.

Triggering and Playback Expectations

The Pharos system is designed to spread the load across all Controllers in a project. You can aid this by patching fixtures as evenly as possible across all Controllers.

As well as lighting playback, Controllers are often used as interfaces to a wider system. This was intentional and is why the system is so flexible but bear in mind that if a Controller is being asked to deal with a substantial amount of incoming triggers or outgoing actions (such as Ethernet or serial communications) then this may have a negative impact on show playback and system responsiveness. If you have any questions about this, please contact support with your project requirements and we'll be happy to advise.

Looping and Holding at End

If you need a timeline to run continuously, we'd always recommend setting it to hold at end (rather than loop) where appropriate. A looping timeline requires significantly more processing power because the Controller has to track the time position until released. You should always release timelines when they are not actively affecting output. More information about the differences between loop and hold at end can be found here.

Transparency

Transparency is a very processor-intensive effect for Controllers to generate. Having one or two layers of transparency on some fixtures will be fine, but having several layers on a fully patched LPC 4 may cause some playback to be choppier than normal.

Touch Device Pages

The limit above is for the total number of pages in a project, not for the number of pages in an interface.

The practical limit for number of pages is affected by the complexity of the pages in the interface, as this is due to the internal memory size of the controller. A better real-world limit would be around 20 pages.

Media Encoding for Pixel Matrices and Primary/Secondary Content Targets

A Pharos project should be able to run any video, however some settings that are known to work well for video without audio are as follows:

- Codec: h.264
- Frame rate: 33fps
- Keyframes: 33 frames
- Bitrate: 5Mbps ought to be sufficient for 1080p30, reducing to 1Mbps for 360p30
- Resolution: 1080p - Best results will be achieved by matching the media resolution to the output resolution (Pixel Matrix or Content Target)

Live Video Settings

The live video input on the LPC X and VLC/VLC+ can accept a variety of incoming resolutions and frame rates. Choose a resolution below to show the accepted frame rate/s.

Resolution	Frame Rate/s

Silent Install

There are circumstances where it may be required to perform a Silent Install of Pharos Designer. This can be achieved using the /S argument when installing from the command line.

The following arguments are also available:

/USB_ETH_BRIDGE="[path]" : installs the Pharos USB to ethernet bridge at the given path

/SHORTCUT : creates a desktop shortcut

/STARTMENU : creates start menu shortcuts

/INSTDIR="[path]" : specified the installation directory.

Glossary

B

bootloader

Bootstrap loader; a small software program, stored in internal flash memory, that is responsible for loading the firmware or operating system.

C

CIDR

"Classless Inter-Domain Routing", a way of specifying the range of IP Addresses that this device is able to communicate with. The number (e.g. 24) refers to the number of bits of the address that must be the same. CIDR 24 is equivalent to a Subnet Mask of 255.255.255.0

Composition

A collection of Content Targets, Content Overlays and Content Masks

compound fixture

A lighting fixture containing more than one controllable element, for example an LED batten consisting of a number of identical elements or pixels.

Content Mask

A defined area which can reduce(or increase) the RGBI levels of the output

Content Overlay

An additional output layer than can be used to output effects over the top of other content within a composition

Content Target

An output layer that media or effects can be mapped onto before being output to fixtures

D

DALI

"Digital Addressable Lighting Interface"; an industry standard digital lighting control protocol.

DHCP

"Dynamic Host Configuration Protocol"; a method of automatically assigning IP addresses.

DMX

USITT DMX512; an industry standard digital lighting control protocol.

E

eDMX

A shorthand term for DMX-over-Ethernet protocols, for example Art-Net.

F

favicon

an icon associated with a particular website, typically displayed in the address bar of a browser accessing the site or next to the site name in a user's list of bookmarks.

firmware

The embedded operating system, stored in internal flash memory or on the memory card.

fixture

Lighting instrument or luminaire.

G

Glossary

Example glossary term

group

A collection of fixtures or elements (pixels) within a fixture that provide a very useful shortcut for selecting and programming them together as one.

I

IP address

"Internet Protocol" address, in the form xxx.xxx.xxx.xxx, which specifies the unique address for networked equipment.

M

matrix

A two-dimensional array of fixtures such that each fixture, or element within a compound fixture, is mapped to a pixel of the array.

Pixel Matrix

A two-dimensional array of fixtures such that each fixture, or element within a compound fixture, is mapped to a pixel of the array.

Pixel Matrices

A two-dimensional array of fixtures such that each fixture, or element within a compound fixture, is mapped to a pixel of the array.

Matrices

A two-dimensional array of fixtures such that each fixture, or element within a compound fixture, is mapped to a pixel of the array.

MIDI

"Musical Instrument Digital Interface"; an industry standard communications protocol for musical instruments.

mover

Any fixture that has control parameters beyond colour mixing (RGB, CMY etc) and intensity, typically an automated light.

moving light

Any fixture that has control parameters beyond colour mixing (RGB, CMY etc) and intensity, typically an automated light.

N

namespace

A collection of Lua variables and functions collected together into a group (e.g. system contains system.hardware_type and system.channel_capacity etc.)

NTP

"Network Time Protocol"; a means of transmitting time signals over a computer network, used to set realtime clocks automatically to the correct time.

P

preset

The basic building block that is placed on a timeline to define what a fixture, or group of fixtures, is to do. Roughly analogous to a cue.

R

RDM

"Remote Device Management"; an extension to the USITT DMX512 protocol that allows for bi-directional communication with the fixture for remote configuration and diagnostics purposes.

Replication

A "copy" of a project file with a different collection of hardware

RS232

EIA-232; an industry standard communications protocol for computing and telecommunications equipment.

RS485

EIA-485; an industry standard communications protocol for computing and industrial equipment.

T

timeline

The framework used to determine which presets are applied to which fixtures, when and for how long. Roughly analogous to a cuelist.

V

variable

A value that can be captured from an input string that is used to determine the outcome of an action.

W

watchdog

A hardware device that monitors a microprocessor and automatically forces a reset if the microprocessor stops responding.

wildcard

A method of specifying which character(s) of an input string should be ignored as padding. Wildcards are also captured as variables and can be considered such if used to determine the outcome of an action.

DMX Record Help

Introduction

Welcome and thank you for using version v1.0 of Pharos DMX Record.

Capture Your Console

DMX Record is a free, standalone utility for capturing multiple universes of sACN or Art-Net lighting data. DMX Record lets you capture complex lighting programming from another lighting control system and import recordings straight into Designer 2 for playback on Pharos Designer LPC family controllers.

DMX Record

Console operators can download DMX Record and capture recordings of programming, with no familiarity of Designer 2 necessary. Recorded data can be reviewed and trimmed in DMX Record, and metadata - such as lighting sequence descriptions and patch information – can be embedded with the recording.

Capture lighting programming from another control system to:

- Playback detailed moving light sequences.
- Save any lighting programming for long term permanent installation.
- Upgrade a legacy system by recording the existing looks and sequences without having to reverse engineer old programming.
- Take mockup and concept programming to a live project.

Move lighting data from a console to an LPC family controller to benefit from the LPC’s autonomous playback and show control features.

Designer 2 Integration

In Pharos Designer 2, import the files shared from DMX Record, then simply drop the recording preset onto a group of fixtures patched the same as in the recording, then trigger the controller to play back the recording on a timeline exactly as programmed on the console. Or, benefit from Designer's powerful and flexible playback tools, override, priority and LTP rules to use the recording data as a foundation, with huge potential to manipulate and programme on top of it:

- Selectively playback only some of the capture by dropping the recording onto a sub-group and it will play only on those fixtures.
- Combine captures by placing multiple recording presets on different fixture groups.
- Colour and intensity can be easily changed with Designer presets programmed as normal on a different timeline, whilst the rest of the captured data continues to playback from the recording preset.

- Gobo, beam, position and other parameters can also be programmed in Scenes in Designer as before, to override parts of a recording.
- Adjust recording speed by setting timeline rate or change the preset length to trim the recording.

Help Overview

The **Help** is split into four sections "Record" on the facing page, "Review" on page 725, "Reference" on page 728, and "Support" on page 733.

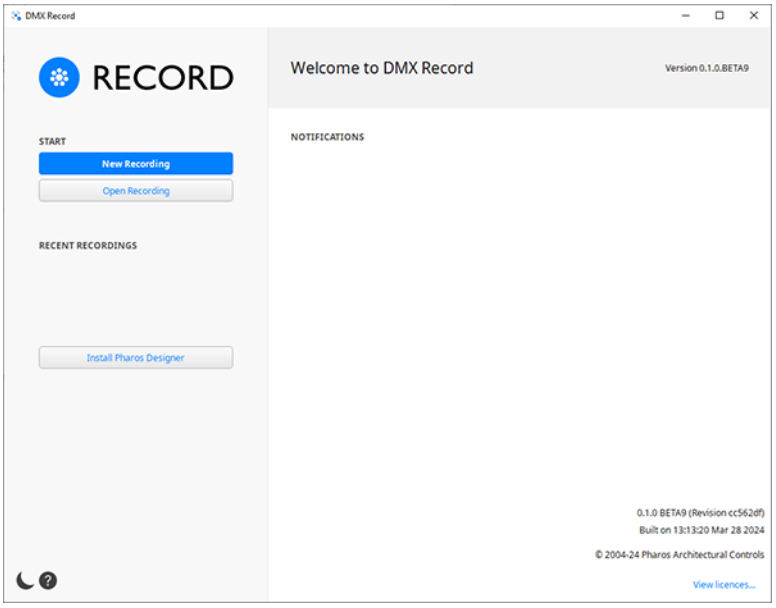
This Help will cover setting up and editing recordings in Pharos DMX Record. For details of applying the recordings in Designer 2 projects, please refer to Designer 2 Help.

Images can be made larger by clicking on them.

Record

Getting Started

Welcome pane: Once you launch the DMX Record utility you will be given options to create **New Recording** or **Open Recording** for any saved recordings. DMX Record lists the last five recordings you have previously opened under **Recent Recordings**, double click to open any listed file.



License Agreement and Notifications – found on the right side of the software UI, shows the license text, and alerts you to any updates and provides access to release notes.

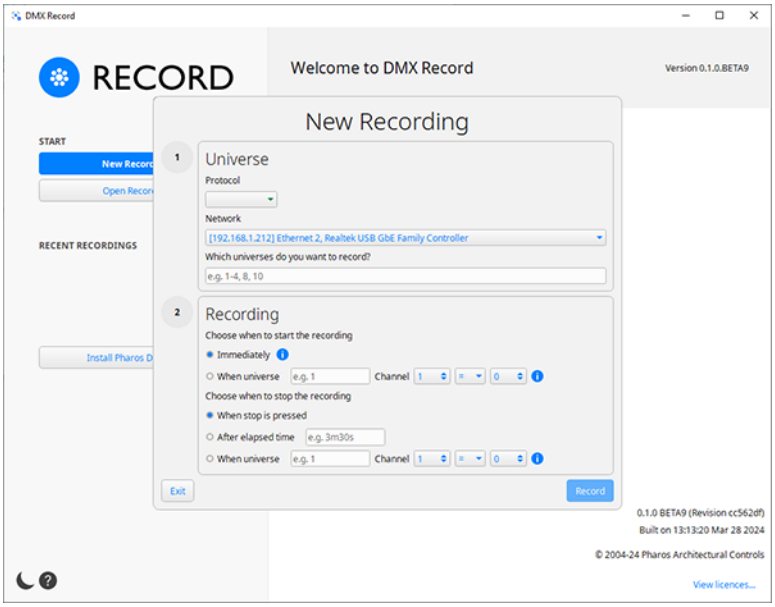
There is also a convenient link to install Pharos Designer 2 software if required. Clicking will open the Designer 2 download page of the Pharos website in the system's default browser.

Theme - clicking this icon toggles between a **Light Theme** or a **Dark Theme**.

Help - clicking this icon opens the DMX Record Help in the system's default browser.

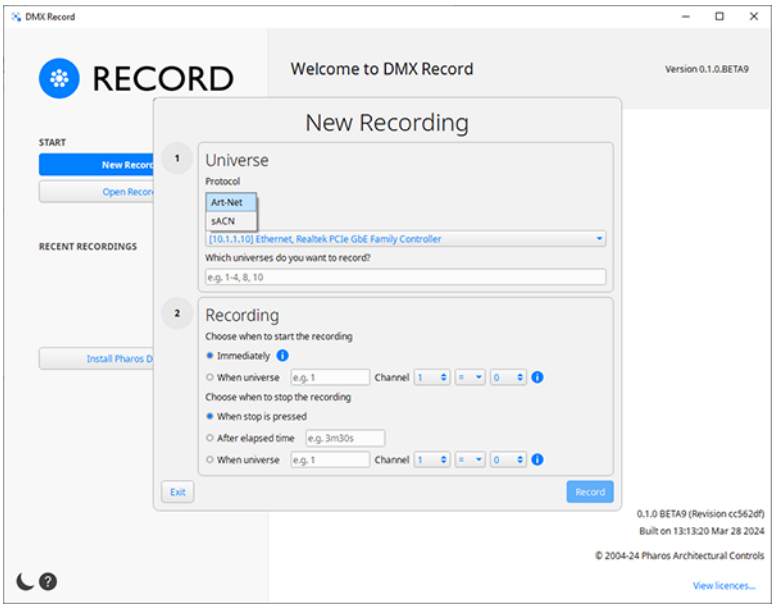
Recordings

To create a **New Recording** click on the button. The **New Recording** window will open with options for setting the type of data you would like to capture and options for starting and ending the recording. Any selected settings from your recording will be remembered for any subsequent recordings.



Universe settings

Protocol - DMX Record supports two eDMX protocols: Art-Net and sACN, both commonly used for the transmission of DMX data over Ethernet. Select the appropriate **Protocol** for your console's output to the network.



For more information see "DMX Glossary" on page 729.

Network - Here you can select the network adapter on your computer that your lighting console is connected to. If there is more than one network adaptor available, select the desired network from the dropdown list.

Which universes do you want to record? - Specify which universes you wish to capture data from, up to a maximum of 16 universes in one recording. You can specify just one universe number, or several, e.g. 1,3,7,9 or a range of universes, e.g. 5-16.

Recording settings

Choose when to start the recording:

Immediately - This is a default option and will enable you to start capturing data as soon as you click the **Record** button.

When universe - This option allows a channel value to trigger the start of the recording. You can either select an active channel that is part of the recording to start the capture, or programme a dummy fixture on any universe (it does not have to be one you are including in the recording) to trigger the start, e.g. a channel on a submaster with a 50% threshold - when the slider is moved up, it starts the recording and when down it stops the recording. Set the universe and channel number (between 1 and 512) then channel value (between 0 and 255) and the behaviour of the channel reaching or passing the threshold value to act as the cue:

=	Equal to
>	Greater than
<	Less than

Choose when to stop the recording:

When stop is pressed - This is a default option and will allow you to stop your recording manually using the stop button in the recording window.

After elapsed time - In this option you can specify a duration for your recording. The maximum length is 30 minutes. You can also set the time by using keys: s for seconds, e.g. 10s will display 10.00, and m for minutes, e.g. 10m will display 10.00.00. Combine these for more precise times, e.g. 25m40s will display 25:40.00.

When this option is selected, the stop button in the recording window can still be used to manually stop a recording.

When universe - This option allows a channel value to trigger the end of the recording. You can either select an active channel that is part of the recording to finish the capture, or programme a dummy fixture on any universe (it does not have to be one you are including in the recording). Set the universe and channel number (between 1 and 512), then channel value (between 0 and 255), and the behaviour of the channel reaching or passing the threshold value to act as the cue:

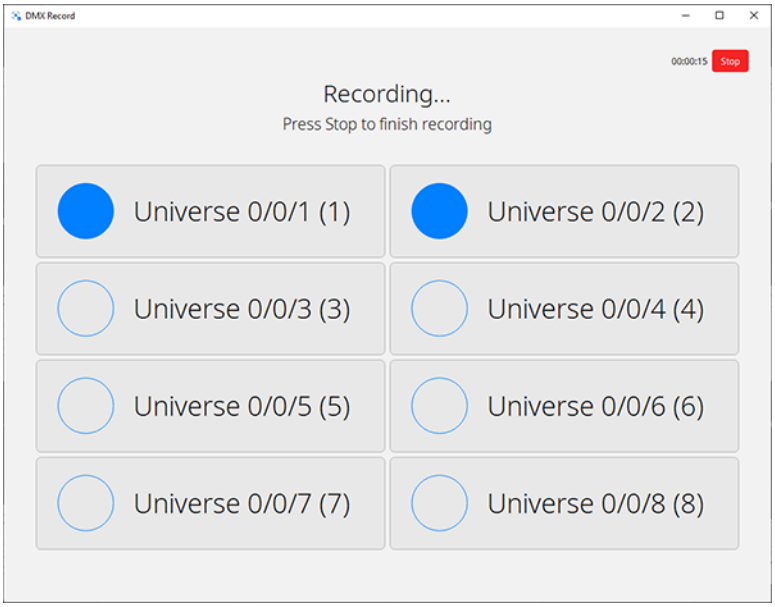
=	Equal to
>	Greater than
<	Less than

When this option is selected, the stop button in the recording window can still be used to manually stop a recording.

Once you have chosen your required settings, select the **Record** button to start your recording.

Recording Live

Here you will see a new window open. This window will show details of what you are recording, including the universes you have selected to record, and a countdown timer if using elapsed time to end the recording.

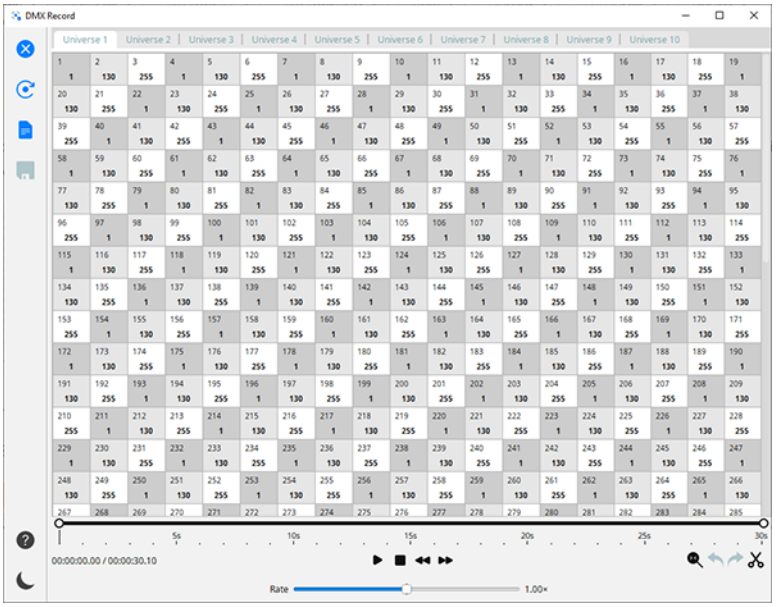


Whilst the recording is in progress the circle beside each numbered universe will flash indicating that the universe is recording changes on one or more channel. If the circle is empty, the universe is not receiving any changes. Depending on the data being recorded, this could be correct.

Press the **Stop** button to end the recording manually, or allow the elapsed time counter or selected channel level to complete the capture as required.

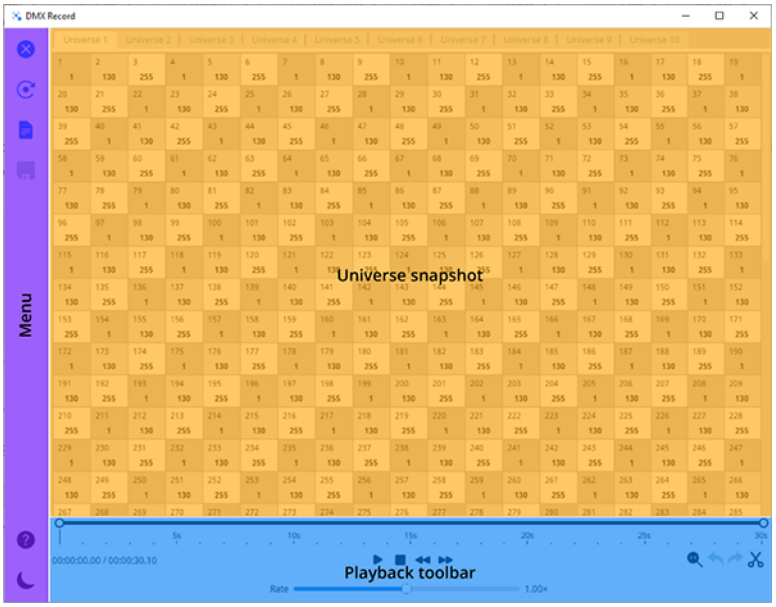
Review

Once your recording has completed, it can be viewed to check the data, and edited to trim the ends if needed. Metadata that will accompany the recording can also be included here.



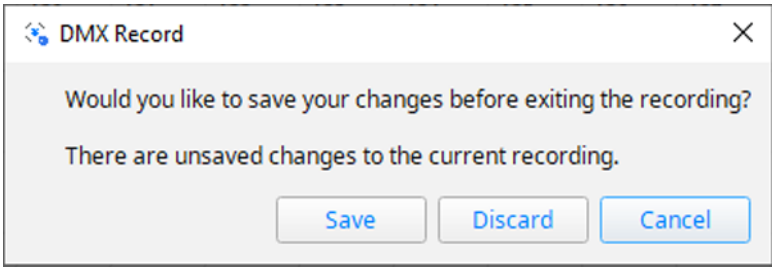
Saved recordings also open in the review window.

User Interface



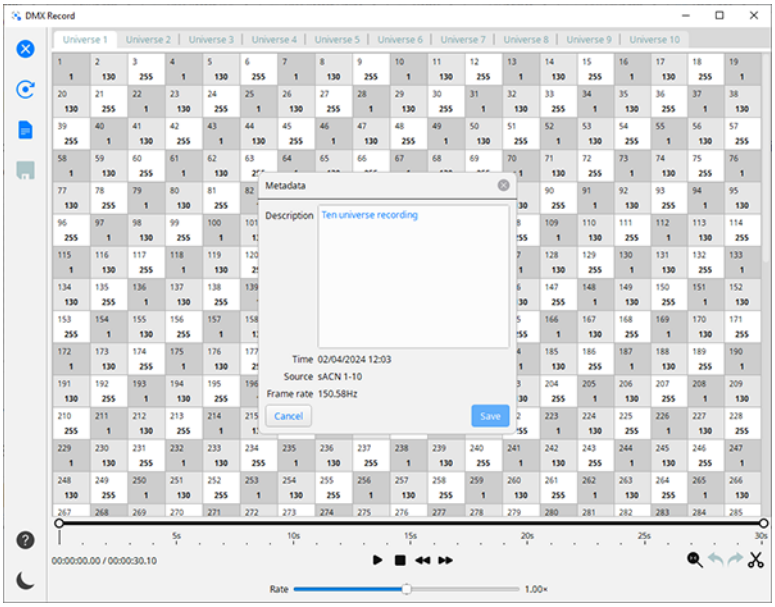
Menu

Exit - To exit the recording. If you exit the recording without saving you will be prompted to save.



Restart Recording - This allows you to restart your recording using the same settings.

View Metadata - This will allow you to add useful information about your recording. The description could include details about the lighting sequence captured, the number and type of fixtures that have been recorded plus patching information. Adding metadata to your recording can be very useful when it is imported into a Designer 2 project.



Save Recording - This allows you to save any changes you may have made to your recording.

Help - Clicking this icon opens the DMX Record Help in the system's default browser.

Theme - Clicking this icon will allow you to toggle between a **Light Theme** or a **Dark Theme** environment.

Universe snapshot

This area of the UI has a tab for each universe included in the recording. The universe view has shaded boxes, which show the channel numbers, from 1 to 512, and the channel values, between 0 and 255. This is a static representation of the data that has been captured for each frame.

Playback toolbar

Contains the tools for viewing and editing the DMX recording:

Playback - The overall duration of the recording is indicated here, along with the playhead showing the position of the current frame being viewed. Clicking on the time strip moves the playhead. The zoom/scroll bar allows you to zoom in to view the time strip at finer resolution and grab the bar to scroll. This is particularly useful in tandem with the Trim function to identify the exact start and stop frames required.

Timestamp showing playhead position / recording duration.

Play/**Pause**, **Stop**, **Rewind** and **Fast forward** controls allow you to view levels of the recorded data playback in the window frame-by-frame in real time.

Playback Rate - This can be adjusted to play the recorded data faster or slower, by moving the slider up to 10 times and down to 0.1 times the real time recorded rate. Double clicking the slider returns it to zero.

Zoom to Fit - This will reset the zoom back to full view.

Undo and Redo applies to Trim edits only.

Trim - With this tool you can edit the length of the recording, by trimming off any frames not required at the beginning or the end of the capture.

In trim mode handles appear at the beginning and the end of the time strip, and the zoom bar highlights the current recording section in view. As the time handles are dragged, the playhead comes under the cursor to indicate the current frame that is being displayed. Clicking on the time strip moves the playhead and updates the frame in view in the snapshot window. Use the handles on the zoom bar to zoom in and scroll to a section of the recording at finer resolution if it is crucial to trim to a particular frame – for instance when a channel value first changes.

Press to confirm or to cancel your trim edits to the recording. The duration time will update.

Reference

In this section you can find additional useful information.

DMX Glossary

Protocols and terminology

Term	Description	For more information:
DMX	A digital serial control protocol for entertainment lighting. Officially called DMX512-A, it was developed by the USITT and has become the standard protocol for entertainment lighting control using the RS485 physical layer.	DMX512
eDMX	A shorthand term for DMX-over-Ethernet protocols, see Art-Net, and sACN below.	
Art-Net	A DMX-over-Ethernet protocol developed by Artistic Licence and widely used in the entertainment industry to distribute multiple universes of DMX data. Current supported version is v4 (transmitting only. v3 for other functionality).	
sACN	Streaming ACN (Advanced Control Network), a DMX-over-Ethernet protocol developed by ESTA to distribute multiple universes of DMX data.	ESTA
Universe	A common term given to a single DMX data link or port. A DMX universe carries 512 channels of control data each with 8 bit resolution. A single dimmer will use one channel while more complex fixtures will use multiple channels as required.	
DMX Address	The term used to determine which of the 512 control channels of a DMX universe a fixture should look at to take its own control data. This "start address" must be set on the fixture or dimmer rack itself as well as patching the control system.	
Start Address	The start address is the first channel that the fixture will take control data from	
Footprint	The footprint is how many channels the fixture needs to function. For instance, an RGB (red, green, blue) fixture has a footprint of 3, whereas an RGBWA (red, green, blue, white, amber) fixture has a footprint of 5.	

Keyboard Shortcuts

Action	Shortcut (Windows)	Shortcut (macOS)	Supported page
Create a new recording	Ctrl + N	⌘ + N	Welcome
Open recording	Ctrl + O	⌘ + O	Welcome
Move up in recent recordings list	Up Arrow	Up Arrow	Welcome
Move down in recent recordings list	Down Arrow	Down Arrow	Welcome
Open selected recent recording	Return	Return	Welcome
Record	Ctrl + R	⌘ + R	New recording popover
Exit	Esc	Esc	New recording popover
Stop recording	Esc	Esc	Recording
Undo	Ctrl + Z	⌘ + Z	Review
Redo	Ctrl + Y or Ctrl + Shift + Z	⌘ + Y or ⌘ + Shift + Z	Review
Save	Ctrl + S	⌘ + S	Review
Show next universe tab	Ctrl + Right Arrow	⌘ + Right Arrow	Review
Show previous universe tab	Ctrl + Left Arrow	⌘ + Left Arrow	Review
Exit recording	Esc	Esc	Review (when playback is not running)
Restart recording	Ctrl + N	⌘ + N	Review
View metadata	Ctrl + M	⌘ + M	Review
Close metadata	Esc	Esc	View Metadata popover
Save metadata	Ctrl + S	⌘ + S	View Metadata popover
Play / Pause	Space	Space	Review
Stop	Esc	Esc	Review (during playback)
Rewind	Left Arrow	Left Arrow	Review
Fast forward	Right Arrow	Right Arrow	Review
Decrease rate	Shift+Left Arrow	Shift+Left Arrow	Review
Increase rate	Shift+Right Arrow	Shift+Right Arrow	Review

Action	Shortcut (Windows)	Shortcut (macOS)	Supported page
Zoom to fit	Ctrl + 0	⌘ + 0	Review
Trim	Ctrl + T	⌘ + T	Review
Cancel	Esc	Esc	Review (while trimming)
Commit	Return	Return	Review (while trimming)

Notes for macOS users

Unless otherwise noted, keyboard shortcuts on macOS are the same as Windows, except Ctrl is replaced with ⌘ . Shift and Alt work as described for Windows.

Frequently Asked Questions

FAQS

What if the console/control device is only outputting local DMX?

Third party dongles are readily available to convert DMX512 to Art-Net so DMX Record can capture the data on a laptop via the Ethernet port.

How many universes?

Up to 16 universes of Art-Net or sACN can be captured in one recording. If more are required, simply make further recordings, which can be merged in Designer by placing recording presets in adjacent groups on the same timeline.

How long?

Up to 30 minutes (at any refresh rate) can be recorded. Longer sequences can be accommodated in Designer by placing another recording immediately after the first on the timeline.

Can I record different protocols at the same time?

Each recording must be the same protocol (either Art-Net or sACN), however multiple recordings can be placed on the same timeline, so captures from different protocols or even different consoles can be combined once the files are in Designer.

Can my controller record DMX at runtime?

No

DMX Record is a commissioning utility for capturing looks before your project is active. It requires a computer to capture data, then move to Designer software to program and upload to an LPC.

Can I use DMX recordings for VLC / VLC+ projects?

VLCs are optimised for fixture arrays and no requirement for DMX Record is anticipated.

What versions of Designer 2 support DMX recordings?

v2.12 and later.

Support

As with all successful control products, support is crucial and the team at Pharos will do everything possible to ensure that your project is a success.

Please do not hesitate to contact us with your questions, bug reports and suggestions at:

T: +44-(0)20-7471-9449

E: support@pharoscontrols.com

Please also visit our website to keep up to date with the latest product news and software releases: www.pharoscontrols.com.