Pylux User Manual

for Pylux v0.1-alpha2

J. Page

2016

Contents

1 Introduction

1.1 About Pylux

Pylux is a program written in Python for the manipulation of OpenLighting Plot documents. Pylux currently allows for the basic editing of the OpenLighting Plot XML files, including referencing OpenLighting Fixture files to obtain additional data.

In the future, Pylux will be extended with modules that allow for the exporting of documentation in the LaTeX format, and creating lighting plots in SVG format.

Bugs and feature requests should be submitted to `https://github.com/jackdpage/pylux/issues`.

1.2 About this Manual

This manual is intended for general users of Pylux. If you are a developer who wants to contribute to Pylux, you should read the Developer Reference. The first section of this manual details all of the actions that you can enter on the command-line interface of Pylux. The second section goes into more detail about what tags you should use, etc.

2 Command-line Options

Pylux is invoked simply by running the `pylux` package with Python.

`--help`, `-h` Display the usage message of the program then exit.

`--version`, `-v` Display the version number of the program then exit.

`--file` *`FILE`*, `-f` *`FILE`* Load the file with path *`FILE`* as the plot file on launch.

`--gui`, `-g` Launch Pylux with the graphical user interface. This is not yet implemented. Omitting this flag launches Pylux with its standard CLI interface.

`--verbose`, `-V` Set the verbosity level of output. Don't include to keep logging level at the default of `WARNING`. Include once for logging level of `INFO` and include twice for a logging level of `DEBUG`.

3 Using the Command Line Interface

The command line interface (CLI) is the default interface used by Pylux. It allows for very efficient editing of plot files with very little CPU overhead.

The CLI is interacted with using a series of commands, each of which may have one or more arguments. When the prompt is displayed, the program is waiting for the user to enter one of these commands. Each command is a memorable two character sequence (apart from the utility commands), where the first character is the object that is going to be acted upon and the second character is the action to perform.

3.1 Piping Complex Objects into Commands

Many commands require that you specify an object other than one which can be represented by a simple command line string. For example, the `xs` command requires that you supply a fixture object as an argument. This is made possible whilst retaining the simplicity of text based entry through the interface reference system.

When you run a listing or filtering command such as `xl`, you will notice that the objects in the list each have a number which is underlined. This is that objects's interface reference. Using this, you can pipe objects into other commands, simply by specifying the number where the command calls for another object type.

To allow for the efficient manipulation of objects in batch, you can specify more than one object at once using a comma-separated list (provided of course that the command allows for multiple objects to be piped into it) such as `a,b,c`. You can also specify ranges of numbers if you are piping sequential objects from a list using the format `a:b` where `a` and `b` are the inclusive limits. You can of course use any combination of these formats, such as `a:b,c,d:e,f`.

If you need to pipe the same object or group of objects into multiple commands, you can use the `this` reference instead of a number or list of numbers. This points to the last used reference, so will pipe in the same object or objects that were used for the last command, unless the reference list has changed since (i.e. a listing command has been run again).

3.2 Utility Commands

`h` Display a list of the available commands for the interactive prompt. This prints the contents of `help.txt` to the output.

`c` Clears the screen of previous input and output. This uses the system screen clearing command. (`clear` on UNIX, `cls` on Windows)

`q` Exit the program and autosave any changes that have been made to the tree.

`Q` Exit the program without saving any changes to disk.

3.3 File Commands

`fo` *`FILE`* Open the file with path *`FILE`* as the plot file. This will override any unsaved buffer associated with the previous plot file, if there was one.

`fw` Save the current file buffer to its original location.

`fW` *`PATH`* Save the current file buffer to a new file with location *`PATH`*.

`fg` Print the path of the file that is currently loaded.

`fn` *PATH* Create an new empty plot file at the location with path *PATH* and load it as the new plot file

3.4 Metadata Commands

`mG` List all the metadata tags associated with the currently loaded plot file.

`ms` *TAG VALUE* Set the value of the metadata with tag *TAG* to *VALUE*. If the metadata already exists, it will be overridden.

`mr` *TAG* Remove the piece of metadata from the file which has the name *TAG*.

`mg` *TAG* Get the value of a piece of metadata. Prints the value of the metadata with name *TAG* on the screen.

3.5 Fixture Commands

`xn` *TEMPLATE* Add a new fixture to the plot. This will load the contents of the fixture file with the name `TEMPLATE` into the new fixture, including DMX functions. This will not allocate DMX addresses to the fixture, use `xA` for that.

`xc` *REF* Clone the fixture with interface reference *REF* into a new fixture. This does not reassign any DMX values.

`xl` List all the fixtures in the plot. This will generate a list of every fixture in the plot, listing an interface reference, the fixture olid, and the fixture UUID.

`xf` *TAG VALUE* List all the fixtures in the plot which have a tag called *TAG* with a value of *VALUE*. Like the list function, this will list an interface reference, the fixture olid and UUID, and also the value of the tag that was given for verification purposes.

`xg` *REF TAG* Print the value of *TAG* for the fixture with interface reference *REF*.

`xG` *REF* List all the information associated with the fixture with interface reference *REF*.

`xr` *REF* Remove the fixture with the interface reference *REF*. This fixture will be removed from the plot, but associated DMX channels will not be removed, use `xp` for that.

`xs` *REF TAG VALUE* Set the value of *TAG* in fixture with interface reference *REF* to *VALUE*. For a list of standard fixture tags, see subsection 6.2. There are also some shortcuts to set mulitple tags at once, which can be found in the pseudo tags section.

`xa` *`REF UNIVERSE ADDR`* Assign DMX addresses to the fixture with interface reference *`REF`*. This will add the fixture to the universe *`UNIVERSE`*, beginning at the start address *`ADDR`*. *`ADDR`* can either be a user-assigned number or auto to allow Pylux to choose the most appropriate start address.

`xp` *`REF`* Remove the fixture with interface reference *`REF`* from the plot and also remove any DMX channels associated with it.

3.6 DMX Registry Commands

`rl` *`UNIVERSE`* List all the used channels in *`UNIVERSE`*. This will list the DMX address, fixture UUID and function of every channel in the DMX registry with identifier *`UNIVERSE`*.

3.7 Using Extensions

You cannot directly interact with extensions from the `editor` interface, you must first load the extension using the `:` command. For example, to load the `texlux` extension, use `:texlux`. This will then present you with the interface as defined by that extension which may vary but in practise should be a prompt of the form `pylux:extension` to indicate to you which extension you are operating in and some commands, much like in the `editor` interface. The extension defines its own way of returning to `editor` but this should in general be `::` or `q`.

4 Using the Graphical User Interface

You may instead choose to launch Pylux using its GUI. This is NYI so please don't.

5 Extensions

In the previous sections, we have discussed the usage of the base program in Pylux: `editor`. This is the program that you will use to edit your plots, however, beyond that, it doesn't do much. That is why Pylux is also bundled with extensions to provide extra functionality. In the current version, Pylux comes bundled with the `texlux` extension only.

5.1 `texlux`

`texlux` is an extension to the base `editor` program which allows for the creation of reports in the LaTeX format, which can then be post-processed to create a PDF file, or many other formats through the use of an external tool such as Pandoc.

5.1.1 Commands

`rn` *`TEMPLATE OUTPUT TITLE`* Generates a report using the template *`TEMPLATE`*, with the title *`TITLE`*, writing the output to a file with path *`OUTPUT`*.

5.1.2 Processing

`texlux` uses built-in functions to generate LaTeX docments with pre-defined structures. Each built-in function has a corresponding LaTeX style file installed in `~/texmf` which is required to build the PDF report. Currently the only built-in function is `dimmer`, which produces a report categorised by dimmer and containing power draw totals.

5.2 `plotgen`

`plotgen` generates, from the fixture list and the fixture's symbol files, a plan view of the lighting plot in SVG format. Currently WIP but functional. Includes support for positioning, rotation and colouring based on fixture data.

6 Standard Tags

Below is a list of standard tags for each section, to advise which tags you should apply to your metadata and fixtures. Also included is a list of pseudo-tags: tags which are not added to the file but actually represent one or more other tags to make adding them easier.

6.1 Standard Metadata Tags

Whilst you can use any name for a tag you wish, there are some standard ones which are used by Pylux and its bundled extensions.

`production` The name of the production for which the lighting documentation is being produced, e.g. 'Romeo and Juliet'. Used by: `texlux`.

`designer` The name of the lighting designer for this production. Used by: `texlux`.

`board_operator` The name of the person operating the main lighting board for this production. Used by: `texlux`.

`spot_operator` The name of the person operating the primary followspot for this production. Used by: `texlux`.

`director` The name of the director of the production. Used by: `texlux`.

`venue` The location at which the production is taking place. Used by: `texlux`.

6.2 Standard Fixture Data Tags

`dmx_functions` This is the parent of a list of empty elements, each of which represents a function that the fixture has that requires the use of a DMX channel. For example, traditional fixtures will have an `intensity` function whilst modern LED fixtures may have `colour` and `mode` functions.

`dmx_channels` The number of DMX channels that a fixture needs. This is automatically calculated from the `dmx_functions` tag, so should not be changed manually.

`dmx_start_address` The start address of this fixture, if it has been addressed. This is automatically assigned using the address function so shouldn't be changed manually.

`universe` The universe in which the DMX channels for this fixture are located. This too is set when the address command is run so shouldn't be changed by the user.

`posX` The x position in 2D space where this fixture is located. Measured in metres.

`posY` The y position in 2D space where this fixture is located. Measured in metres.

`focusX` The x position in 2D space where the centre of this fixture's beam is focused. Measured in metres.

`focusY` The y position in 2D space where the centre of this fixture's beam is focused. Measured in metres.

`rotation` The rotation of this fixture about its centre. Measured anticlockwise from the positive x axis in degrees. This can be automatically calculated if the preceding four data tags are present.

`circuit` For traditional fixtures only. The circuit into which the fixture is patched. Used by: `texlux`.

`power` For traditional fixtures only. The maximum power draw by the lamp in this fixture.

`dimmer_uuid` For traditional fixtures only. The UUID of the dimmer (which must exist as a separate fixture in the plot file) which is controlling this fixture.

`dimmer_chan` For traditional fixtures only. The name or number of the dimmer channel by which this fixture is controlled.

`gel` The manufacturer's code of the gel that is being used in this fixture. The automatic colour calculation currently supports Rosco Supergel and E-colour and the named HTML (X11) colours.

`colour` A hexadecimal colour code indicating the colour which best represents the gel in this fixture. Can be automatically calculated if gel is present.

6.2.1 Pseudo Fixture Tags

These tags can be used to set multiple attributes of a fixture at once.

`position` *X*,*Y* Sets `posX` to *X* and `posY` to *Y*.

`focus` *X*,*Y* Sets `focusX` to *X* and `focusY` to *Y*.

`dimmer` *REF* *CHAN* Sets `dimmer_uuid` to the uuid of the dimmer represented by *REF* and `dimmer_channel` to *CHAN*.