DMX Lighting Server

Installation

- Open Lighting Architecture if using OLA for the DMX output
  - With Python API enabled
- PySerial
- CherryPy
- ws4py
- Python dateutil
- Apache2 and PHP5, if using the html command line interface
- websocket-client, for calling single commands

Other Resources

You may need to install and configure your raspberry pi in other ways before you can install the lighting controller. Here are some resources for common setups:

- Use `sudo raspi-config` to configure basic system settings
- Configuring a wireless adapter
- Setting up RPI to log in automatically
- Use `sudo apt-get install vim` to install the vim text editor

Steps

Install the Raspian (Debian Wheezy) image onto an SD card.

Install the Dependencies:

- Update aptitude:
  - `sudo apt-get update`
- Install PIP to install Python dependencies:
  - `sudo apt-get install python-pip`
- Install Python development headers:
  - `sudo apt-get install python-dev`
- Install Python dependencies:
  - `sudo pip install pyserial cherrypy ws4py python-dateutil`
- Install Apache and PHP:
  - `sudo apt-get install apache2`
  - `sudo apt-get install php5-common libapache2-mod-php5 php5-cli` More info
  - Restart Apache: `sudo service apache2 restart`

Clone the repository:

- `git clone https://github.com/eburlingame/lightingserver.git`

Setting up HTML Command Line To access the HTML command line page we must serve it using Apache. To do this we must set Apache's document root to the `/web` folder.

- Change the virtual hosts configuration:
  - `sudo vim /etc/apache2/sites-available/default`
  - Change the line `DocumentRoot /var/www/` to `DocumentRoot /home/pi/lightingserver/web/`
  - Change the line `<Directory /var/www/>` to `<DocumentRoot /home/pi/lightingserver/web/>`
  - Use `Escape` and `:wq` to save and quit vim
- Restart Apache with `sudo service apache2 restart`

Running the Server

To start the server and it's command line interface simply run:

`python lightingserver/main.py`

Running the Server at Startup

If you wish to start the server automatically at startup:

- Edit `/etc/profile` with `sudo vim /etc/profile`
- Add `python ~/lightingserver/main.py` at the end of the file
- Use `Escape` and `:wq` to save and quit vim

Usage

The server takes commands through the command line interface or through a websocket connection. These commands change the state of the controller, and thus the state of the DMX output. These commands are designed to be basic, but flexible, allowing a user, or another piece of software, to call commands and provide ample control of the output.

Notes

Commands are processed without whitespace, which means the program does not rely on splitting by spaces, etc. Bear in mind that this can cause issues if two words run into one another and form a different word.

Commands

Command Description Format We will use these special characters to describe the format of commands:

- `(*, *, *)`: a, b, and c will all perform the same function
- `[*]` : A value that will be entered by the user
- `~*` : Denotes an optional field

Channel Selection

Channels can be selected in the following ways:

- `[channel #1] (/, thru, through) [channel #2]`
  - Selects channel #1 through channel #2 (inclusive)
- `[channel #1] (&, and, +) [channel #2]`

  - Selects channel #1 and channel #2
- `[channel #1] (&, and, +) [channel #2] (except, -) [channel #2]`
  - Selects channel #1 through channel #2, discluding channel #3
- `Group [Group name]`
  - Selects the channel in the group name
- `Fixture [Fixture label]`
  - Selects the fixture(s) matching that label
- `Fixture [range selection] ~channel ~[channel number]`
  - Selects the channels associated with the given fixture (only a single channel can be selected)
  - If the channel is supplied, then it will select the relative channel associated with that fixture

Basic Control

- `[Channel Selection] (@, at, *) [percent]`
  - Set the given output DMX channel to a given percent
- `(@, at, *) [percent]`
  - Sets the last selected channels to percent.

Patching

Channels must be patched to an output DMX channel to be set.

- `patch channel [channel selection] dmx [dmx channel selection] ~fixture [fixture number] ~label [fixture label]`
  - Patches a new channel in the given channel number to the starting DMX address described
  - A fixture number can be optionally specified
  - A label can be optionally specified
- `patch (one-to-one, one2one) channel [channel selection] dmx [channel address selection]`
  - Creates a 1 to 1 channel patch with matching the two selection
  - The same number of channels and DMX address must be inputed
- `unpatch channel [channel selection]`
  - Unpatched the given channels
- `unpatch dmx [dmx channel selection]`
  - Unpatched the channels associated with the given DMX output channels
- `print patch`
  - Prints a text description of the current channel patching

Groups

- `save group [name]`
  - Saves the last selected channels into a group with the specified name
- `save group [name] channel [channel selection]`
  - Saves the specified channels into a group with the specified name
- `list groups`
  - Prints a list of all the groups currently saved
- `print group [group name]`
  - Prints a text description of the given group

Scenes

- `save (scene) [scene name] ~fade [~fade time] ~channel ~[channel selection]`
  - Saves the current channel values (those that are above 0%) as a scene with the given name
  - Specifying a fade time will make that its default fade up time
  - Specifying the channels will save only those channels
- `save (scene) [scene name] ~fade [~fade time] { fixture commands }`
  - Works the same as above, except the channel commands will define the scene contents
- `load scene [scene name] ~fade [~fade time] ~%~[percent]`
  - Loads the given scene with the given fade.
  - Uses the saved fade time if none is supplied, or the default if it is not saved.
  - Specifying a percent will load the scene at that intensity
- `load scene [scene name] ~fade [~fade time] ~%~[percent] channel [channel selection]`
  - Loads the given scene with the given fade.
  - Uses the saved fade time if none is supplied, or the default if it is not saved. The channel options lets you specify a range of channels you want the scene to act over.
- `delete scene [scene name]`
  - Will delete the scene
- `clear scenes`
  - Will delete all the scenes save in the controller
- `list scenes`
  - Will list a text representation of the scenes currently saved in the controller
- `print scene [scene name]`
  - Will print a textual representation of the given scene

Sequences

- `save sequence [sequence name] ~insert ~step ~[step] ~fade [~fade time] ~wait [~wait time] ~all ~cued ~channel ~[channel selection]`
  - Saves the current channel values (those above 0%) as the step for a sequence
  - Specifying the `step` will overwrite that step, or insert it if it does not exist.
  - If `insert` is provided then it will insert it at that location
  - Omitting `fade` or `wait` will use the default fade and wait time, or the times defined by the user when loaded
  - Specifying the channel will save only those channels specified
  - Adding `cued` to the first save of the sequence will require the sequence to be stepped through manually using the `advance` command
- `save sequence [sequence name] ~insert ~step ~[step] ~fade [~fade time] ~wait [~wait time] ~(cued) { [channel commands] }`
  - Works the same as above, except the contents of the sequence step will be defined by the channel commands surrounded by brackets
- `load sequence [sequence name] ~fade [~fade time] ~wait [~wait time] ~step ~[step number] ~(cued)`
  - Loads the given sequence with the designated fade time and wait time, if supplied
  - The saved fade and wait times will be used, or the ones stored with the sequence if supplied.
  - If step is supplied it will start the sequence at that step.
  - When the same sequence is running multiple times at once (for instance over a different set of channels) it will be given a unique running id, which can be used to advance, pause or stop the sequence

- `delete sequence [sequence name]`
  - Deletes the entire sequence
- `delete sequence [sequence name] step [step number]`
  - Deletes the specified step in the given sequence
- `unload ~sequence [sequence name] ~id [running id] ~all`
  - Stops the given sequence and removes it from the running queue. Restarting requires the load command.
  - Supplying the running id parameter will unload the sequence with that matching id; not supplying it will unload all sequences matching that name.
  - If all is supplied it will stop all sequences
- `pause ~[sequence name] ~id [running id]`
  - Pauses the given sequence, where it can be resumed with the unpause command.
- `(pause, hold) all`
  - Pauses all sequences
- `(unpause, play) [sequence name] ~id [running id]`
  - Resumes the given sequence.
- `advance [sequence name] ~fade [fade time] ~id` [running id]
  - Advances the given sequence to its next step.
  - If the sequence is not running, it will load the sequence with default parameters
- `clear sequences`
  - Deletes all sequences in the controller
- `print sequence [sequence name]`
  - Prints a text description of the sequence given
- `list sequences`
  - Lists the currently save seqeuences
- `list running`
  - Lists the sequences that are currently running along with their running ids
- `load-csv [filename]`
  - This will read in data from a comma-separated text file and put the contents in its own sequence named the same as the file
  - The `filename` is the relative file path from the root folder of the server (where `main.py` is located) to a valid comma-separated file
  - The columns of the CSV file should be formatted as followed:
    * Step number (ignored)
    * Label
    * Fade time
    * Wait time
    * Channel 1 value
    * Channel 2 value
    * ...

Shortcuts

Shortcuts are defined to more easily run commands. Before any commands are run, the shortcuts are processed and the shortcut is replaced with its command.

- `define "[shortcut]" as "[command to be run]`
  - Creates a custom command/macro that can be used as a shortcut for a command
  - The first argument is the shortcut the user will type in.
  - The second argument is the command that will be run (it will replace the shortcut).
  - You cannot use reserved keywords (words used in built-in commands) in the shortcut
  - You can use custom arguments in the shortcut by replacing them with a #. In the command, you can use the syntax [0] to refer to the first arguments [1] to the second, and so on
    - Consider 'define "# half" as "[0] at 50" ', when '2 half' is entered, it will be replaced with '2 at 50', and set channel 2 to 50%
- `list shortcuts`
  - Lists the currently saved shortcuts
- `delete shortcut "[shortcut]"`
  - Deletes the shortcut

Scheduling

- `schedule "[command]" at [time]`
  - Will execute the given command at the specified time
  - The time format is flexible, and can be entered in a variety of ways including:
    - HH:MM
    - HH:MM:SS
    - HH:MM am/pm
    - ...

Utility Functions

- `print ~channels`
  - Running `print` or `print channels` will print out the current channel values
- `open`
  - Open the DMX output on the USB interface
- `close`
  - Close the DMX output on the USB interface
- `write`
  - Writes all stored information to the `data.txt` file
- `read`
  - Reads all the information from the `data.txt` file
- `server start ~port [port]`
  - Attempts to open the server on the specified port
  - Port will default to 8080 if not specified
- `server stop`
  - Stops the server if it is running.