iot-ledmatrix Documentation

Release 0.1

enra64, gSilas

Apr 22, 2017

Contents:

CHAPTER 1

Custom scripts

Custom scripts are what makes iot-ledmatrix powerful. You can add any gimmick you want by adding an implementation to one of the subfolders of *scripts*.

Scripts beginning with _ will not be displayed in the app to be manually loaded. Only a custom fragment can load them, because they can only run properly with that custom fragment. Scripts *not* beginning with _ can be loaded by the user in a default fragment that lists them. Other than that, new custom scripts do not need any configuration effort.

When your new script is requested in the app, it will be loaded and can draw to the led matrix.

How to draw from within the script

Drawing to the matrix is done by using the functions of the canvas supplied with the draw calls. Detailed documentation is available here: *Canvas class*

Creating a new script

Custom scripts must contain a class that is exactly the name of the source file minus the *.py*.

For example, if you create a "flashlight" script, the file name would be `flashlight.py`, and the class name would be `flashlight`.

For debugging, a simple GUI was implemented. Use the `--enable-gui` flag to display it.

The class must inherit from CustomScript, which is documented here:

CustomScript class

class `CustomScript.`**`CustomScript`**(*canvas*, *send_object*, *send_object_to_all*, *start_script*, *restart_self*, *set_frame_period*, *set_frame_rate*, *get_connected_clients*)

The CustomScript class is the class you want to inherit from to implement a new matrix mode.

In addition to the constructor, there are six methods that will be called by the manager:

- *update()*, where the state may be updated
- *draw()*, where the matrix content may be drawn
- *on_data()*, called when messages from clients arrive
- *exit()*, last call before the instance is discarded
- *on_client_connected()*, called with an id parameter when a new client is approved
- *on_client_disconnected()*, called with an id parameter when a client has disconnected

They have default implementations, so you only need to override them if you need to do anything.

A few methods can also be called by the script itself (self.<function_name>(<param1...>)):

- *send_object()* to send objects to specific clients
- *send_object_to_all()* to send objects to all clients
- *start_script()* start a script by name. will replace current script.
- *restart_self()* restarts the current script
- *get_connected_clients()* gets a list of approved client ids
- *set_frame_period()* allows to set a custom update cycle period in seconds
- *set_frame_rate()* allows to set a custom update cycle calling rate in Hz

All of these functions are documented more detailed in their method documentations.

Script Lifecycle

The constructor will always be called first. Do your initialization here. Update will always be called before draw. The two functions are called in a loop, and will repeatedly execute. exit is always the last method call.

See the method documentations for further information.

draw(*canvas: Canvas.Canvas*)
Called after update. Make any modifications of the canvas you want to do here. After this method has finished executing, the canvas buffer will be sent to the arduino and displayed.

Parameters **canvas** – the canvas you can draw on. will be displayed on the arduino

Returns nothing

exit()
Called when the manager gracefully wants to stop this script. This instance will be discarded of after.

get_connected_clients()
Get a list of connected clients. The list will only contain the ids as given by zmq, which may be used to send data to clients. To send data to all clients, use send_object_to_all.

Returns list of zmq ids.

on_client_connected(*id*)
Called when a client connects

Parameters **id** – id of the client that disconnected

Returns

on_client_disconnected(*id*)
Called when a client disconnects

Parameters **id** – id of the client that disconnected

Returns

`on_data`(*data_dictionary*, *source_id*)
Called whenever the android app sends data for the script.

Parameters

- **`data_dictionary`** – a dictionary of data received from the android app.
- **`source_id`** – the network id of the sending android device

Returns nothing

`restart_self`()
Will restart the current script. exit() will be called on this instance. A new instance will be created. No additional arguments can be given.

Returns nothing

`send_object`(*obj*, *target*)
Send an object to the target id. The object can be anything, but a dict is probably easiest. No JSON serialization needs to be performed by you.

Parameters

- **`obj`** – the object to be sent
- **`target`** – target id of the client

Returns nothing

`send_object_to_all`(*obj*)
Send an object to all connected clients. TThe object can be anything, but a dict is probably easiest. No JSON serialization needs to be performed by you.

Parameters **`obj`** – the object to be sent

Returns nothing

`set_frame_period`(*period*)
Change the frame period with which the script will be updated

Parameters **`period`** – the target frame period. resulting frame rate must be 0 <= f <= 60, in Hz

Returns nothing

`set_frame_rate`(*frame_rate*)
Change the frame rate with which the script will be updated

Parameters **`frame_rate`** – the target frame rate. must be 0 <= f <= 60, in Hz

Returns nothing

`start_script`(*script_name*, *source_id*)
Will load the class in the scripts/ folder that has the given name in the file with the same name.

Parameters

- **`script_name`** – the name of _both_ the script and the class implementing the callback functions
- **`source_id`** – the id of the client requesting the script to be loaded

`update`(*canvas*)
Called before draw. Do any updating you want to do here.

Parameters `canvas` – canvas object for information like width and height

CHAPTER 2

Canvas class

class `Canvas.`**`Canvas`**(*width*, *height*)

A canvas makes it easy to display using the matrix by providing a translation layer between pixels on a cartesian coordinate system and color data readable by the arduino and the WS2812B RGB leds.

The canvas uses colors from the colour library to represent the requested colors. See https://github.com/vaab/colour

The user functions are:

- draw_pixel(x, y, color)
- draw_line(x_start, y_start, x_end, y_end, color)
- draw_rect(x, y, width, height, color)
- draw_text(x, y, text, color, ignore_height_warning=False)
- set_font(path, size)
- clear(color)

`clear`(*color: colour.Color = <Color black>*)

Set all pixels to some color

Parameters **`color`** – the color that should be applied

`draw_line`(*x_start: int*, *y_start: int*, *x_end: int*, *y_end: int*, *color: colour.Color*)

An implementation of bresenhams line drawing algorithm. Draws a line from <x_start, y_start> to <x_end, y_end> in the given color.

Parameters

- **`x_start`** – x position where the line should start
- **`y_start`** – y position where the line should end
- **`x_end`** – x position where the line should start
- **`y_end`** – y position where the line should end
- **`color`** – color the line should be drawn in

Returns nothing

`draw_pixel` (*x: int*, *y: int*, *color: colour.Color*)

Set a pixel to a color. Most basic canvas function.

Parameters

- **`x`** – x position of pixel; counted from zero beginning on the left, must be smaller than the canvas width
- **`y`** – y position of pixel; y is zero for the top row of pixels, must be smaller than the canvas height
- **`color`** – the description of the color that should be set

`draw_rect` (*x: int*, *y: int*, *width: int*, *height: int*, *color: colour.Color*)

Parameters

- **`x`** – x position of pixel; counted from zero beginning on the left, must be smaller than the canvas width
- **`y`** – y position of pixel; y is zero for the top row of pixels, must be smaller than the canvas height
- **`width`** – how wide the rectangle should be.
- **`height`** – how high the rectangle should be
- **`color`** – the color the rectangle should have

Returns nothing

`draw_text` (*text: str*, *x: int*, *y: int*, *color: colour.Color*, *ignore_height_warning=False*)

Draw text on the canvas. Rendering over the borders is cut off, so you do not need boundary checking.

Parameters

- **`text`** – the text to be rendered
- **`x`** – the top-left starting position of the text
- **`y`** – the top-left starting position of the text
- **`color`** – color of the text
- **`ignore_height_warning`** – if true, no warning will be logged that the font does not fit into the available height. if false, a warning will be printed in the log on each such occasion

Returns nothing

`get_buffer_for_arduino` () → bytearray

This method can be used to retrieve the internal data buffer. Modifications will probably do weird shit. Mostly useful for pushing the data out to the arduino, who actually understands what all the numbers mean.

Returns a bytearray with all color values

`get_color` (*x*, *y*) → colour.Color

Get a Color instance describing the color of the led at x,y

Parameters

- **`x`** – x position of pixel; counted from zero beginning on the left, must be smaller than the canvas width

- **y** – y position of pixel; y is zero for the top row of pixels, must be smaller than the canvas height

Returns a Color instance

get_pixel_index (*x*, *y*)

Convert a cartesian coordinate for an led into the index that represents red for that led in the buffer.

Currently, the chaining is assumed to be in a zig-zag, as follows:

	col0	col1	col2	col3	col4	col5	col6	col7	col8	col9
row0	99	98	97	96	95	94	93	92	91	90
row1	80	81	82	83	84	85	86	87	88	89
row2	79	78	77	76	75	74	73	72	71	70
row3	60	61	62	63	64	65	66	67	68	69
row4	59	58	57	56	55	54	53	52	51	50
row5	40	41	42	43	44	45	46	47	48	49
row6	39	38	37	36	35	34	33	32	31	30
row7	20	21	22	23	24	25	26	27	28	29
row8	19	18	17	16	15	14	13	12	11	10
row9	0	1	2	3	4	5	6	7	8	9

Parameters

- **x** – x coordinate of the led in the matrix (counted left-to-right)
- **y** – y coordinate of the led in the matrix (counted top-to-bottom)

Returns index of the red value of that led (g, b, are +1, +2 of that position respectively) in the buffer

get_red_index (*x*, *y*)

Pretty much like get_pixel_index, but this function returns the position of the red value of the given led in the byte buffer.

Parameters

- **x** – x coordinate of led in cartesian system
- **y** – y coordinate of led in cartesian system

Returns position of "red" in the background buffer

set_font (*path: str*, *size: int*)

Load a font to be used for rendering all following text. (see draw_text)

Parameters

- **path** – path to the font
- **size** – size of the font. For a 10x10 matrix, 13 is an acceptable, if rather large, choice.

Returns nothing

CHAPTER 3

Configuration options

All configuration options are command-line arguments. The script will change its working directory to the main.py location.

```
--test-with-serial                 run only tests testing serial connection
--test                             run all tests but those requiring an arduino + leds␣
↪be connected
--set-arduino-port=                set the port the arduino is connected on manually,␣
↪like /dev/ttyUSB0
--name=                            set the name the ledmatrix will advertise itself as
--width=                           set the horizontal number of leds
--height=                          set the vertical number of leds
--data-port=                       set the data port the ledmatrix server will use
--discovery-port=                  set the discovery port the led matrix discovery␣
↪server will use
--loglevel=                        set python logging loglevel
--disable-arduino-connection       disable arduino connection. mostly useful for␣
↪debugging without an arduino
--errors-to-console                divert errors to console instead of logfile
--logfile=                         set log file location. best to use absolute paths.
--start-script=                    set starting script, defaults to 'gameoflife'
--enable-gui                       enable a simplistic gui displaying what the matrix␣
↪should currently show. combine with --disable-arduino-connection for easy testing.␣
↪will fuck up stopping. recommended for debugging only
```

iot-ledmatrix is a code base produced to use a diy rgb led matrix made from WS2812B leds. The code was written by enra64 and gSilas.

The code consists of three parts:

- the python code used on the raspberry pi inside the matrix
- the android code making up the control app
- the arduino code required to talk to the leds

This documentation is mostly concerned with the python code used on the rpi, since at the moment the other code is only written by enra64.

CHAPTER 4

Raspberry pi (host) code

The raspberry pi code is responsible for pushing the correct colors to the arduino, and also constitutes the bridge between the matrix, the internet and an optional android phone.

custom scripts

Custom scripts enable you to easily create new features for the matrix. They are discussed in detail here: *Custom scripts*

CHAPTER 5

android code

The android app included in client-android makes working with the matrix really easy. It supports some administration features, and it is the basis for interactive scripts.

administration

Users can reboot the raspberry pi, shut it down or simply restart the host code. A log viewer is also implemented, so failures can be quickly debugged.

host script fragments

Programmers can write Fragments that display an arbitrary user interface to implement any required custom functionality. Two-Way communication with the matrix is available.

CHAPTER 6

arduino sketch

The arduino code is simple, but `NUM_LEDS_CURRENT` must be set before uploading the code. The arduino will partake in a simple handshake to confirm correct initialization. After that, the arduino writes all received data into the led buffer. Whenever enough bytes for a single frame have arrived, the leds will show the new data.

Python Module Index

Index