Adafruit_ImageLoad_Library Documentation

Release 1.0

Scott Shawcroft

Aug 22, 2019

Contents

This library decodes an image file into new bitmap and palette objects of the provided type. It's designed to load code needed during decoding as needed. This is meant to minimize the memory overhead of the decoding code.

CHAPTER 1

Usage Example

```
import displayio
import adafruit_imageload

image, palette = adafruit_imageload.load("images/4bit.bmp",
                                         bitmap=displayio.Bitmap,
                                         palette=displayio.Palette)
```

CHAPTER 2

Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 3

Building locally

3.1 Zip release files

To build this library locally you'll need to install the circuitpython-build-tools package.

```
python3 -m venv .env
source .env/bin/activate
pip install circuitpython-build-tools
```

Once installed, make sure you are in the virtual environment:

```
source .env/bin/activate
```

Then run the build:

```
circuitpython-build-bundles --filename_prefix adafruit-circuitpython-imageload --
↪library_location .
```

3.2 Sphinx documentation

Sphinx is used to build the documentation based on rST files and comments in the code. First, install dependencies (feel free to reuse the virtual environment from above):

```
python3 -m venv .env
source .env/bin/activate
pip install Sphinx sphinx-rtd-theme
```

Now, once you have the virtual environment activated:

```
cd docs
sphinx-build -E -W -b html . _build/html
```

This will output the documentation to `docs/_build/html`. Open the index.html in your browser to view them. It will also (due to -W) error out on any warning like Travis will. This is a good way to locally verify it will pass.

CHAPTER 4

Table of Contents

4.1 Simple test

Ensure your image loads with this simple test.

Listing 1: examples/imageload_simpletest.py

```
import displayio
import adafruit_imageload

image, palette = adafruit_imageload.load("images/4bit.bmp",
                                         bitmap=displayio.Bitmap,
                                         palette=displayio.Palette)
```

4.2 adafruit_imageload

Load pixel values (indices or colors) into a bitmap and colors into a palette.

- Author(s): Scott Shawcroft

adafruit_imageload.**load**(*filename*, *, *bitmap=None*, *palette=None*)

Load pixel values (indices or colors) into a bitmap and colors into a palette.

bitmap is the desired type. It must take width, height and color_depth in the constructor. It must also have a _load_row method to load a row's worth of pixel data.

palette is the desired pallete type. The constructor should take the number of colors and support assignment to indices via [].

4.3 adafruit_imageload.bmp

Load pixel values (indices or colors) into a bitmap and colors into a palette from a BMP file.

- Author(s): Scott Shawcroft

adafruit_imageload.bmp.**load**(*file*, *, *bitmap=None*, *palette=None*)
Loads a bmp image from the open `file`.

Returns tuple of bitmap object and palette object.

Parameters

- **bitmap** (*object*) – Type to store bitmap data. Must have API similar to `displayio.Bitmap`. Will be skipped if None
- **palette** (*object*) – Type to store the palette. Must have API similar to `displayio.Palette`. Will be skipped if None

4.4 adafruit_imageload.bmp.indexed

Load pixel values (indices or colors) into a bitmap and colors into a palette from an indexed BMP.

- Author(s): Scott Shawcroft

adafruit_imageload.bmp.indexed.**decode_rle**(*bitmap*, *file*, *compression*, *y_range*, *width*)
Helper to decode RLE images

adafruit_imageload.bmp.indexed.**load**(*file*, *width*, *height*, *data_start*, *colors*, *color_depth*, *compression*, *, *bitmap=None*, *palette=None*)
Loads indexed bitmap data into bitmap and palette objects.

Parameters

- **file** (*file*) – The open bmp file
- **width** (*int*) – Image width in pixels
- **height** (*int*) – Image height in pixels
- **data_start** (*int*) – Byte location where the data starts (after headers)
- **colors** (*int*) – Number of distinct colors in the image
- **color_depth** (*int*) – Number of bits used to store a value
- **compression** (*int*) – 0 - none, 1 - 8bit RLE, 2 - 4bit RLE

4.5 Developing

Strategy: * read headers to determine file type * keep a pointer to the start of data * read data into the Palette for all colors present * rewind the file pointer back to start of data * read data into the Bitmap * return a bitmap and palette instance

4.5.1 Shared Bindings

This library uses interfaces from CircuitPython's `displayio` module (Bitmap and Palette) to load images from disk into memory.

The Bitmap and Palette objects are related, and together can be used to display the image on a screen for the user.

The Palette is a list of colors present in the image. Its constructor takes a single argument: (int) max_colors, representing how many colors will be populated in the palette.

4.5.1.1 Palette code example

```
palette = Palette(4)  # 4 represents that we will define four colors in palette
palette[0] = b'\x00\x00\x00\x00'  # white
palette[1] = b'\xFF\x00\x00\x00'  # red
palette[2] = b'\x00\xFF\x00\x00'  # green
palette[3] = b'\x00\x00\xFF\x00'  # blue
```

4.5.1.2 Bitmap code example

```
bitmap = Bitmap(3, 2, 4)  # 3 pixels wide, two pixels tall, 4 colors
bitmap[0,0] = 0  # palette color 0
bitmap[0,1] = 1  # palette color 1
...
```

4.5.1.3 Example of Palette and Image

The example is 4bit.bmp from the examples/images folder:

The Palette object appears like this after loading:

```
Palette:
[0] b'\x00\x00\x00\x00'
[1] b'\x7f\x00\x00\x00'
[2] b'\xff\x00\x00\x00'
[3] b'w\x00\xb1\x00'
[4] b'\xff\x00\x9d\x00'
[5] b'\x00\x00\xff\x00'
[6] b'\xff\x00\xfe\x00'
[7] b'\xbf\x80\x00\x00'
[8] b'zzz\x00'
[9] b'\xff\x9e\xa5\x00'
[10] b'\x00\x98\xff\x00'
[11] b'\x00\xff\x00\x00'
[12] b'h\xff\x00\x00'
[13] b'\xfb\xff\x9e\x00'
[14] b'\x00\xfb\xff\x00'
[15] b'\xfb\xfb\xfb\x00'
```

This palette has 16 colors. The value in square brackets [] is the color's index in the palette. The byte values are the RGB or RGB + padding of each color.

The Bitmap is an grid of which palette color to use in each position of the image.

The corresponding Bitmap to the example above appears like this after loading:

```
5   5   5   5   5   5   5   5   5   5   5   5   5   5   5
5   5   5   5   5   5   5   5   5   5   5   5   5   5   5
5   5   5   5   5   5   5   5   5   5   5   5   5   5   5
11  11  11   5   5   5  15  15  15   5   5   5   2   2   2
11  11  11   5   5   5  15  15  15   5   5   5   2   2   2
6   6   6   5   5   5   1   1   1   5   5   5  10  10  10
6   6   6   5   5   5   1   1   1   5   5   5  10  10  10
6   6   6   5   5   5   1   1   1   5   5   5  10  10  10
14  14  14   5   5   5   9   9   9   5   5   5   8   8   8
14  14  14   5   5   5   9   9   9   5   5   5   8   8   8
14  14  14   5   5   5   9   9   9   5   5   5   8   8   8
3   3   3   5   5   5   0   0   0   5   5   5  13  13  13
3   3   3   5   5   5   0   0   0   5   5   5  13  13  13
4   4   4   5   5   5  12  12  12   5   5   5   7   7   7
4   4   4   5   5   5  12  12  12   5   5   5   7   7   7
4   4   4   5   5   5  12  12  12   5   5   5   7   7   7
5   5   5   5   5   5   5   5   5   5   5   5   5   5   5
```

This grid represents the example image (`15 pixels wide` and `17 pixels tall`). The coordinates are arranged in a zero indexed grid, starting in the top left at `[0,0]`, and continuing down and to the right to a final coordinate of `[14,16]`.

The value at each position is an integer, representing an entry in the palette object.

For example, the Bitmap coordinate `[0,0]` has the value (integer) `5`.

This corresponds to the the Palette object's, `[5]` which is `b'\x00\x00\xff\x00'`. This is a byte string that represents a color.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index