uFlash Documentation

Release 1.2.4

Nicholas H.Tollervey

Dec 14, 2018

Contents

THIS MODULE ONLY WORKS WITH PYTHON 2.7 or 3.3+.

A utility for flashing the BBC micro:bit with Python scripts and the MicroPython runtime. You pronounce the name of this utility "micro-flash". ;-)

It provides two services:

1. A library of functions to programatically create a hex file and flash it onto a BBC micro:bit.
2. A command line utility called *uflash* that will flash Python scripts onto a BBC micro:bit.

Several essential operations are implemented:

- Encode Python into the hex format.
- Embed the resulting hexified Python into the MicroPython runtime hex.
- Extract an encoded Python script from a MicroPython hex file.
- Discover the connected micro:bit.
- Copy the resulting hex onto the micro:bit, thus flashing the device.
- Specify the MicroPython runtime hex in which to embed your Python code.

CHAPTER 1

Installation

To install simply type:

```
$ pip install uflash
```

. . . and the package will download from PyPI. If you wish to upgrade to the latest version, use the following command:

```
$ pip install --no-cache --upgrade uflash
```

NB: You must use a USB *data* cable to connect the micro:bit to your computer (some cables are power only). You're in good shape if, when plugged in, the micro:bit appears as a USB storage device on your file system.

Linux users: For uflash to work you must ensure the micro:bit is mounted as a USB storage device. Usually this is done automatically. If not you've probably configured automounting to be off. If that's the case, we assume you have the technical knowledge to mount the device yourself or to install the required kernel modules if they're missing. Default installs of popular Linux distros "should just work" (tm) out of the box given a default install.

CHAPTER 2

Command Usage

To read help simply type:

```
$ uflash --help
```

or:

```
$ uflash -h
```

To discover the version information type:

```
$ uflash --version
```

If you type the command on its own then uflash will attempt to find a connected BBC micro:bit and flash an unmodified default version of the MicroPython runtime onto it:

```
$ uflash
Flashing Python to: /media/ntoll/MICROBIT/micropython.hex
```

To flash a version of the MicroPython runtime with a specified script embedded within it (so that script is run when the BBC micro:bit boots up) then pass the path to the Python script in as the first argument to the command:

```
$ uflash my_script.py
Flashing Python to: /media/ntoll/MICROBIT/micropython.hex
```

You can let uflash watch for changes of your script. It will be flashed automatically every time you save it:

```
$ uflash -w my_script.py
```

or:

```
$ uflash --watch my_script.py
```

At this point uflash will try to automatically detect the path to the device. However, if you have several devices plugged in and/or know what the path on the filesystem to the BBC micro:bit already is, you can specify this as a second argument to the command:

```
$ uflash myscript.py /media/ntoll/MICROBIT
Flashing Python to: /media/ntoll/MICROBIT/micropython.hex
```

You can even flash multiple devices at once:

```
$ uflash myscript.py /media/ntoll/MICROBIT /media/ntoll/MICROBIT1
Flashing Python to: /media/ntoll/MICROBIT/micropython.hex
Flashing Python to: /media/ntoll/MICROBIT1/micropython.hex
```

To extract a Python script from a hex file use the "-e" flag like this:

```
$ uflash -e something.hex myscript.py
```

This will save the Python script recovered from "something.hex" into the file "myscript.py". If you don't supply a target the recovered script will emit to stdout.

If you're developing MicroPython and have a custom runtime hex file you can specify that uflash use it instead of the built-in version of MicroPython in the following way:

```
$ uflash -r firmware.hex
```

or:

```
$ uflash --runtime=firmware.hex
```

CHAPTER 3

Development

The source code is hosted in GitHub. Please feel free to fork the repository. Assuming you have Git installed you can download the code from the canonical repository with the following command:

```
$ git clone https://github.com/ntoll/uflash.git
```

Ensure you have the correct dependencies for development installed by creating a virtualenv and running:

```
$ pip install -r requirements.txt
```

To locally install your development version of the module into a virtualenv, run the following command:

```
$ python setup.py develop
```

There is a Makefile that helps with most of the common workflows associated with development. Typing `make` on its own will list the options thus:

```
$ make

There is no default Makefile target right now. Try:

make clean - reset the project and remove auto-generated assets.
make pyflakes - run the PyFlakes code checker.
make pep8 - run the PEP8 style checker.
make test - run the test suite.
make coverage - view a report on test coverage.
make check - run all the checkers and tests.
make package - create a deployable package for the project.
make publish - publish the project to PyPI.
make docs - run sphinx to create project documentation.
```

CHAPTER 4

Contributing to uFlash

Hey! Many thanks for wanting to improve uFlash.

Contributions are welcome without prejudice from *anyone* irrespective of age, gender, religion, race or sexuality. If you're thinking, "but they don't mean me", then we especially mean YOU. Good quality code and engagement with respect, humour and intelligence wins every time.

- If you're from a background which isn't well-represented in most geeky groups, get involved - *we want to help you make a difference*.
- If you're from a background which *is* well-represented in most geeky groups, get involved - *we want your help making a difference*.
- If you're worried about not being technical enough, get involved - *your fresh perspective will be invaluable*.
- If you think you're an imposter, get involved.
- If your day job isn't code, get involved.
- This isn't a group of experts, just people. Get involved!
- We are interested in educational, social and technical problems. If you are too, get involved.
- This is a new community. *No-one knows what they are doing*, so, get involved.

We expect contributors to follow the Python Software Foundation's Code of Conduct: https://www.python.org/psf/codeofconduct/

Feedback may be given for contributions and, where necessary, changes will be politely requested and discussed with the originating author. Respectful yet robust argument is most welcome.

Finally, contributions are subject to the following caveat: the contribution was created by the contributor who, by submitting the contribution, is confirming that they have the authority to submit the contribution and place it under the license as defined in the LICENSE file found within this repository.

4.1 Checklist

- Your code should be commented in *plain English* (British spelling).

- If your contribution is for a major block of work and you've not done so already, add yourself to the AUTHORS file following the convention found therein.
- You MUST include tests. We have 100% test coverage.
- Have fun!

CHAPTER 5

API

This module contains functions for turning a Python script into a .hex file and flashing it onto a BBC micro:bit.

uflash.**MICROPYTHON_VERSION = '1.0.1'**
The version number reported by the bundled MicroPython in os.uname().

uflash.**embed_hex**(*runtime_hex*, *python_hex=None*)
Given a string representing the MicroPython runtime hex, will embed a string representing a hex encoded Python script into it.

Returns a string representation of the resulting combination.

Will raise a ValueError if the runtime_hex is missing.

If the python_hex is missing, it will return the unmodified runtime_hex.

uflash.**extract**(*path_to_hex*, *output_path=None*)
Given a path_to_hex file this function will attempt to extract the embedded script from it and save it either to output_path or stdout

uflash.**extract_script**(*embedded_hex*)
Given a hex file containing the MicroPython runtime and an embedded Python script, will extract the original Python script.

Returns a string containing the original embedded script.

uflash.**find_microbit**()
Returns a path on the filesystem that represents the plugged in BBC micro:bit that is to be flashed. If no micro:bit is found, it returns None.

Works on Linux, OSX and Windows. Will raise a NotImplementedError exception if run on any other operating system.

uflash.**flash**(*path_to_python=None*, *paths_to_microbits=None*, *path_to_runtime=None*, *python_script=None*, *minify=False*)

Given a path to or source of a Python file will attempt to create a hex file and then flash it onto the referenced BBC micro:bit.

If the path_to_python & python_script are unspecified it will simply flash the unmodified MicroPython runtime onto the device.

If used, the python_script argument should be a bytes object representing a UTF-8 encoded string. For example:

```
script = "from microbit import *\ndisplay.scroll('Hello, World!')"
uflash.flash(python_script=script.encode('utf-8'))
```

If paths_to_microbits is unspecified it will attempt to find the device's path on the filesystem automatically.

If the path_to_runtime is unspecified it will use the built in version of the MicroPython runtime. This feature is useful if a custom build of MicroPython is available.

If the automatic discovery fails, then it will raise an IOError.

uflash.**get_minifier**()

Report the minifier will be used when minify=True

uflash.**get_version**()

Returns a string representation of the version information of this project.

uflash.**hexlify**(*script*, *minify=False*)

Takes the byte content of a Python script and returns a hex encoded version of it.

Based on the hexlify script in the microbit-micropython repository.

uflash.**main**(*argv=None*)

Entry point for the command line tool 'uflash'.

Will print help text if the optional first argument is "help". Otherwise it will ensure the optional first argument ends in ".py" (the source Python script).

An optional second argument is used to reference the path to the micro:bit device. Any more arguments are ignored.

Exceptions are caught and printed for the user.

uflash.**save_hex**(*hex_file*, *path*)

Given a string representation of a hex file, this function copies it to the specified path thus causing the device mounted at that point to be flashed.

If the hex_file is empty it will raise a ValueError.

If the filename at the end of the path does not end in '.hex' it will raise a ValueError.

uflash.**strfunc**(*raw*)

Compatibility for 2 & 3 str()

uflash.**unhexlify**(*blob*)

Takes a hexlified script and turns it back into a string of Python code.

uflash.**watch_file**(*path*, *func*, **args*, ***kwargs*)

Watch a file for changes by polling its last modification time. Call the provided function with *args and **kwargs upon modification.

CHAPTER 6

Release History

6.1 1.2.4

- Updated to the latest version of MicroPython for micro:bit (1.0.1)
- This is the version of uflash to be used in Mu 1.0.2.

6.2 1.2.3

- Update to the latest version of MicroPython for micro:bit (1.0.0).
- This is the version of uflash to be used in Mu 1.0.1.

6.3 1.2.2

- Update to latest version of MicroPython for micro:bit (1.0.0-rc.3).

6.4 1.2.1

- Update to latest version of MicroPython. Thanks to Damien George and Carlos Pereira Atencio for their hard work.
- This is the version of uflash to be used in Mu 1.0.0 (final).

6.5 1.2.0

- Update to latest version of MicroPython. Thanks to Damien George.

- Add attribute called MICROPYTHON_VERSION to report the version of MicroPython bundled with uflash.

6.6 1.1.1

- Update to the latest version of MicroPython for the BBC micro:bit – fixes a bug relating to flooding and the radio module. As always, many thanks to Damien George for his work on MicroPython.

6.7 1.1.0

- Update to latest version of MicroPython for the BBC micro:bit (many thanks to Damien George for his amazing efforts!).
- Add a –version flag to uflash that causes it to print the current version number (many thanks to Lenz Grimmer for this work).
- Allow uflash to accept the content of a script as well as the path to a script (many thanks to Zander Brown for this work).
- Ensure uflash works nicely / better with external tools (many thanks to Lex Robinson for this work).
- Added copyright and license information to the start of the script.

6.8 1.0.8

- Refactor hex extraction to not depend on extended address record before script (thanks Carlos).
- Refactor tox tests to fix Windows related Gremlin (thanks again, Carlos).

6.9 1.0.7

- Watch for changes in a script. Automatically flash on save.

6.10 1.0.5

- Update runtime to include latest bug fixes and inclusion of input() builtin.
- Detecting drives on Windows 10 no longer causes pop-ups in certain situations.
- Documentation updates.

6.11 1.0.4

- Add support for flash multiple microbits.

6.12 1.0.3

- Update runtime to include audio and speech modules.

6.13 1.0.2

- Update runtime to include the new radio module.

6.14 1.0.1

- Update runtime to include file system related changes.

6.15 1.0.0.final.0

- Runtime updated to version 1.0 of MicroPython for the BBC micro:bit.

6.16 1.0.0.beta.7

- Runtime update to fix display related bug.

6.17 1.0.0.beta.6

- Runtime update to latest version of the DAL (swaps pins 4 and 5).

6.18 1.0.0.beta.5

- Runtime update to fix error reporting bug.

6.19 1.0.0.beta.4

- Documentation update.
- Help text update.

6.20 1.0.0.beta.3

- Add ability to specify a MicroPython runtime to use.
- Test fixes.

6.21 1.0.0.beta.2

- Updated to latest version of MicroPython runtime.

6.22 1.0.0.beta.1

- Works with Python 2.7 (thanks to @Funkyhat).
- Updated to the latest build of MicroPython for the BBC micro:bit.
- Minor refactoring and updates to the test suite due to MicroPython updates.

6.23 0.9.17

- Minor code refactor.
- Documentation update.

6.24 0.9.14

- Feature complete.
- Comprehensive test suite - 100% coverage.
- Tested on Linux and Windows.
- Documentation.
- Access via the "uflash" command.

6.25 0.0.1

- Initial release. Basic functionality.

CHAPTER 7

License

Python Module Index

u

Index