MicroFS Documentation

Release latest

Nicholas H.Tollervey

Sep 21, 2021

Contents

A simple command line tool and module for interacting with the limited file system provided by MicroPython on the BBC micro:bit.

CHAPTER 1

Installation

To install simply type:

```
$ pip install microfs
```

. . . 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 microfs
```

CHAPTER 2

Usage

There are two ways to use microfs - as a module in your Python code or as a stand-alone command to use from your shell (`ufs`).

2.1 In Code

In your Python script import the required functions like this:

```
from microfs import ls, rm, put, get, get_serial
```

Read the API documentation below to learn how each of the functions works.

2.2 Command Line

From the command line (but not the Python shell) use the "ufs" ("u" = micro) command.

To read the built-in help:

```
$ ufs --help
```

List the files on the device:

```
$ ufs ls
```

You can also specify a delimiter to separte file names displayed on the output (default is whitespace ' '):

```
# use ';' as a delimiter
$ ufs ls ';'
```

Delete a file on the device:

```
$ ufs rm foo.txt
```

Copy a file onto the device:

```
$ ufs put path/to/local.txt
```

Get a file from the device:

```
$ ufs get remote.txt
```

The `put` and `get` commands optionally take a further argument to specify the name of the target file:

```
$ ufs put /path/to/local.txt remote.txt
$ ufs get remote.txt local.txt
```

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/microfs.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.
```

3.1 Contributing to MicroFS

Hey! Many thanks for wanting to improve MicroFS.

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.

CHAPTER 4

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!

4.1 API

This module contains functions for running remote commands on the BBC micro:bit relating to file system based operations.

You may:

- ls - list files on the device. Based on the equivalent Unix command.
- rm - remove a named file on the device. Based on the Unix command.
- put - copy a named local file onto the device a la equivalent FTP command.
- get - copy a named file from the device to the local file system a la FTP.

`microfs.`**`ls`**(*serial=None*)
List the files on the micro:bit.

If no serial object is supplied, microfs will attempt to detect the connection itself.

Returns a list of the files on the connected device or raises an IOError if there's a problem.

`microfs.`**`rm`**(*filename*, *serial=None*)
Removes a referenced file on the micro:bit.

If no serial object is supplied, microfs will attempt to detect the connection itself.

Returns True for success or raises an IOError if there's a problem.

microfs.**put**(*filename*, *target=None*, *serial=None*)
Puts a referenced file on the LOCAL file system onto the file system on the BBC micro:bit.

If no serial object is supplied, microfs will attempt to detect the connection itself.

Returns True for success or raises an IOError if there's a problem.

microfs.**get**(*filename*, *target=None*, *serial=None*)
Gets a referenced file on the device's file system and copies it to the target (or current working directory if unspecified).

If no serial object is supplied, microfs will attempt to detect the connection itself.

Returns True for success or raises an IOError if there's a problem.

microfs.**get_serial**()
Detect if a micro:bit is connected and return a serial object to talk to it.

4.2 Release History

CHAPTER 5

1.4.4

- New feature. Thanks to @makinteract, it is possible to add an optional delimeter for the `ls` command. Please see PR #28 for more details.

CHAPTER 6

1.4.3

- Bug fix. See commentary in issue #22. Thanks again to alexandros769.

CHAPTER 7

1.4.2

- Update getting of data from micro:bit device to deal with control characters found within the file on the device. Thanks to Damien George for the fix and to GitHub user alexandros769 for reporting it.

CHAPTER 8

1.4.1

- Clamp PySerial version to use with microfs to a version known to work.

CHAPTER 9

1.4.0

- Updated and changed the `get` functionality to work on a wider range of supported boards. Many thanks to Carlos Pereira Atencio for putting in the effort to make this happen.

CHAPTER 10

1.3.1

- Fix bug in version parsing that was mangling the `machine` attribute.

CHAPTER 11

1.3.0

- Added a new function (not available via the command line) to get the version of MicroPython on the device.
- **API CHANGE** The find_microbit function now returns a tuple with position 0 as the port and position 1 as the serial number of the connected device.

CHAPTER 12

1.2.3

- Extensive debugging and a fix by Carlos Pereira Atencio to ensure that serial connections are opened, closed and made ready for microfs related commands in a robust manner.

CHAPTER 13

1.2.2

- The get and put commands optionally take a further argument to specify the name of the target file.

CHAPTER 14

1.2.1

- Made implicit string concatenation explicit.

CHAPTER 15

1.2.0

- **API CHANGE** the serial object passed into command functions is optional.
- **API CHANGE** call signature changes on command functions.

CHAPTER 16

1.1.2

- Allow external modules to use built-in device detection and connection.

CHAPTER 17

1.1.1

- Unlink command logic from device detection and serial connection.

CHAPTER 18

1.1.0

- Fix broken ‘put’ and ‘get’ commands to work with arbitrary file sizes.
- Fix error when working with binary data.
- Update execute function to work with lists of multiple commands.
- Minor refactor to extract raw mode related code.
- Updated tests to keep coverage at 100% on both Python 2 and Python 3.

CHAPTER 19

1.0.2

- Remove spare print call.

CHAPTER 20

1.0.1

- Fix broken setup.

CHAPTER 21

1.0.0

- Full implementation of all the expected features.
- 100% test coverage.
- Comprehensive documentation.

CHAPTER 22

0.0.1

- Initial release. Basic functionality.

Python Module Index

m

Index