# micropy-cli

*Release 4.2.2*

Braden Mars

Jan 12, 2024

# DOCUMENTATION

Micropy Cli is a project management/generation tool for writing Micropython code in modern IDEs such as VSCode. Its primary goal is to automate the process of creating a workspace complete with:

- **Linting** compatible with Micropython
- VSCode **Intellisense**
- **Autocompletion**
- Dependency Management
- VCS Compatibility

CHAPTER

ONE

# INSTALLATION

You can download and install the latest version of this software from the Python package index (PyPI) as follows:

```
pip install --upgrade micropy-cli
```

## 1.1 Installation

You can download and install the latest version of this software from the Python package index (PyPI) as follows:

```
pip install --upgrade micropy-cli
```

If applicable, you can test out a pre-release by executing:

```
pip install --upgrade --pre micropy-cli
```

## 1.2 Getting Started

### 1.2.1 Creating a Project

Creating a new project folder is as simple as:

1. Executing `micropy init <PROJECT NAME>`
2. Selecting which features to enable
3. Selecting your target device/firmware
4. Boom. Your workspace is ready.

### 1.2.2 Micropy Project Environment

When creating a project with `micropy-cli`, two special items are added:

- A `.micropy/` folder
- A `micropy.json` file

The `.micropy/` contains symlinks from your project to your `$HOME/.micropy/stubs` folder. By doing this, micropy can reference the required stub files for your project as relative to it, rather than using absolute paths to `$HOME/.micropy`. How does this benefit you? Thanks to this feature, you can feel free to push common setting files such as `settings.json` and `.pylint.rc` to your remote git repository. This way, others who clone your repo can achieve a matching workspace in their local environment.

Note: The generated `.micropy/` folder should be *IGNORED* by your VCS. It is created locally for each environment via the `micropy.json` file.

The `micropy.json` file contains information micropy needs in order to resolve your projects required files when other clone your repo. Think of it as a `package.json` for micropython.

### 1.2.3 Cloning a Micropy Environment

To setup a Micropy environment locally, simply:

- Install `micropy-cli`
- Navigate to the project directory
- Execute `micropy`

Micropy will automatically configure and install any stubs required by a project thanks to its `micropy.json` file.

### 1.2.4 Project Dependencies

While all modules that are included in your targeted micropython firmware are available with autocompletion, intellisense, and linting, most projects require external dependencies.

Currently, handling dependencies with micropython is a bit tricky. Maybe you can install a cpython version of your requirement? Maybe you could just copy and paste it? What if it needs to be frozen?

Micropy handles all these issues for you automatically. Not only does it track your project's dependencies, it keeps both `requirements.txt` and `dev-requirements.txt` updated, enables autocompletion/intellisense for each dep, and allows you to import them just as you would on your device.

This allows you to include your requirement however you want, whether that be as a frozen module in your custom built firmware, or simply in the `/lib` folder on your device.

#### Installing Packages

To add a package as a requirement for your project, run:

```
micropy install <PACKAGE_NAMES>
```

while in your project's root directory.

This will automatically execute the following:

- Source `PACKAGE_NAMES` from pypi, as a url, or a local path
- Retrieve the module/package and stub it, adding it to your local `.micropy` folder.
- Add requirement to your `micropy.json`
- Update `requirements.txt`

To install dev packages that are not needed on your device, but are needed for local development, add the `--dev` flag. This will do everything above **except** stub the requirement.

You can also install all requirements found in `micropy.json/requirements.txt/dev-requirements.txt` by executing `micropy install` without passing any packages. Micropy will automatically do this when setting up a local environment of an existing micropy project.

### Example

Lets say your new project will depend on picoweb and blynklib. Plus, you'd like to use rshell to communicate directly with your device. After creating your project via `micropy init`, you can install your requirements as so:

Now you or anybody cloning your project can import those requirements normally, and have the benefits of all the features micropy brings:

## 1.2.5 Stub Management

Stub files are the magic behind how micropy allows features such as linting, Intellisense, and autocompletion to work. To achieve the best results with MicropyCli, its important that you first add the appropriate stubs for the device/firmware your project uses.

> Note: When working in a micropy project, all stub related commands will also be executed on the active project. (i.e if in a project and you run `micropy stubs add <stub-name>`, then that stub retrieved AND added to the active project.)

### Adding Stubs

Adding stubs to Micropy is a breeze. Simply run: `micropy stubs add <STUB_NAME>` By sourcing micropy-stubs, MicroPy has several premade stub packages to choose from.

These packages generally use the following naming schema:

`<device>-<firmware>-<version>`

For example, running `micropy stubs add esp32-micropython-1.11.0` will install the following:

- Micropython Specific Stubs
- ESP32 Micropython v1.11 Device Specific Stubs
- Frozen Modules for both device and firmware

You can search stubs that are made available to Micropy via `micropy stubs search <QUERY>`

Alternatively, using `micropy stubs add <PATH>`, you can manually add stubs to Micropy. For manual stub generation, please see Josvel/micropython-stubber.

### Creating Stubs

Using `micropy stubs create <PORT/IP_ADDRESS>`, MicropyCli can automatically generate and add stubs from any Micropython device you have on hand. This can be done over both USB and WiFi.

> Note: For stub creation, micropy-cli has additional dependencies.
>
> These can be installed by executing: `pip install micropy-cli[create_stubs]`

#### Viewing Stubs

To list stubs you have installed, simply run `micropy stubs list`.

To search for stubs for your device, use `micropy stubs search <QUERY>`.

## 1.3 See Also

- VSCode IntelliSense, Autocompletion & Linting capabilities
  - An awesome article written by lemariva. It covers creating a micropython project environment from scratch using `micropy-cli` and pymakr-vsc. Great place to start if you're new to this!

## 1.4 Acknowledgements

### 1.4.1 Micropython-Stubber

Josvel/micropython-stubber

Josverl's Repo is full of information regarding Micropython compatibility with VSCode and more. To find out more about how this process works, take a look at it.

micropy-cli and micropy-stubs depend on micropython-stubber for its ability to generate frozen modules, create stubs on a pyboard, and more.

## 1.5 CLI Usage

## 1.6 API Reference

| | |
|---|---|
| *micropy* | Micropy Cli. |
| *micropy.main* | Main Module. |
| *micropy.exceptions* | Micropy Exceptions. |
| *micropy.stubs* | micropy.stubs |
| *micropy.stubs.source* | micropy.stubs.source<br>:2: (WARNING/2) Title underline too short.<br>micropy.stubs.source ~~~~~~~~~~~~~~ |
| *micropy.project* | Module for generating/managing projects. |
| *micropy.project.modules* | Project Modules. |
| *micropy.utils* | micropy.utils |
| *micropy.config* | Configuration files and interfaces for them. |
| *micropy.config.config_source* | Config Abstract. |
| *micropy.packages* | Packages Module. |

### 1.6.1 micropy

Micropy Cli.

Micropy Cli is a project management/generation tool for writing Micropython code in modern IDEs such as VSCode. Its primary goal is to automate the process of creating a workspace complete with:

Linting compatible with Micropython, VSCode Intellisense, Autocompletion, Dependency Management, VCS Compatibility and more.

#### Classes

| | |
|---|---|
| `MicroPy`(*[, options]) | Handles App State Management. |

### 1.6.2 micropy.main

Main Module.

#### Functions

| | |
|---|---|
| `parse_file_as`(type_, path, *[, ...]) | **rtype** `TypeVar(T)` |

#### Classes

| | |
|---|---|
| `Log`() | Borg for easy access to any Log from anywhere in the package. |
| `MicroPy`(*[, options]) | Handles App State Management. |
| `MicroPyOptions`(*[, root_dir, stubs_dir]) | |
| `Path`(*args, **kwargs) | PurePath subclass that can make system calls. |
| `Project`(path[, name]) | Micropy Project. |
| `RepositoryInfo`(**data) | |
| `StubManager`([resource, repos]) | Manages a collection of Stubs. |
| `StubRepository`([manifests, packages_index, ...]) | |

### 1.6.3 micropy.exceptions

Micropy Exceptions.

#### Exceptions

| | |
|---|---|
| `MicropyException` | Generic MicroPy Exception. |
| `PyDeviceConnectionError`(location) | |
| `PyDeviceError`([message]) | Generic PyDevice exception. |
| `PyDeviceFileIntegrityError`(device_path, ...) | |
| `RequirementException`(*args, **kwargs) | A Requirement Exception Occurred. |
| `RequirementNotFound`(*args, **kwargs) | A requirement could not be found. |
| `StubError`([message, stub]) | Exception for any errors raised by stubs. |
| `StubNotFound`([stub_name]) | Raised when a stub cannot be found. |
| `StubValidationError`(path, errors, *args, ...) | Raised when a stub fails validation. |

### 1.6.4 micropy.stubs

#### micropy.stubs

This module contains all functionality relating to stub files/frozen modules and their usage in MicropyCli

#### Classes

| | |
|---|---|
| `MicropyStubPackage`(**data) | |
| `MicropythonStubsManifest`(**data) | |
| `MicropythonStubsPackage`(**data) | |
| `RepositoryInfo`(**data) | |
| `StubManager`([resource, repos]) | Manages a collection of Stubs. |
| `StubPackage`(**data) | |
| `StubRepository`([manifests, packages_index, ...]) | |
| `StubRepositoryPackage`(manifest, package) | |
| `StubsManifest`(**data) | |

### 1.6.5 micropy.stubs.source

#### micropy.stubs.source

This module contains abstractions for handling stub sources and their location.

##### Functions

| | |
|---|---|
| `cast`(typ, val) | Cast a value to a type. |
| `contextmanager`(func) | @contextmanager decorator. |
| `get_source`(location, **kwargs) | Factory for StubSource Instance. |
| `reduce`(function, iterable[, initial]) | Apply a function of two arguments cumulatively to the items of a sequence or iterable, from left to right, so as to reduce the iterable to a single value. |

##### Classes

| | |
|---|---|
| `Any`(*args, **kwargs) | Special type indicating an unconstrained type. |
| `ExitStack`() | Context manager for dynamic management of a stack of exit callbacks. |
| `LocateStrategy`(*args, **kwargs) | |
| `Log`() | Borg for easy access to any Log from anywhere in the package. |
| `Path`(*args, **kwargs) | PurePath subclass that can make system calls. |
| `Protocol`() | Base class for protocol classes. |
| `RemoteStubLocator`() | Stub Source for remote locations. |
| `RepoStubLocator`(repo) | |
| `StubInfoSpecLocator`() | |
| `StubSource`([locators, location]) | Handles sourcing stubs. |
| `partial` | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### 1.6.6 micropy.project

Module for generating/managing projects.

#### Classes

| | |
|---|---|
| `Project`(path[, name]) | Micropy Project. |

### 1.6.7 micropy.project.modules

Project Modules.

#### Classes

| | |
|---|---|
| `DevPackagesModule`(path, **kwargs) | Project Module for Dev Packages. |
| `HookProxy`(name) | Proxy for Project Hooks. |
| `PackagesModule`(path, **kwargs) | Project Module for handling requirements. |
| `ProjectModule`([parent, log]) | Abstract Base Class for Project Modules. |
| `StubsModule`(stub_manager[, stubs]) | Project module for handling Stubs. |
| `TemplatesModule`([templates, run_checks]) | Project Templates Module. |

### 1.6.8 micropy.utils

#### micropy.utils

This module provides utility functions that are used within MicropyCli.

#### Functions

| | |
|---|---|
| `create_dir_link`(source, target) | Creates a platform appropriate directory link. |
| `ensure_existing_dir`(path) | Ensure path exists and is a directory. |
| `ensure_valid_url`(url) | Ensure a url is valid. |
| `extract_tarbytes`(file_bytes, path) | Extract tarfile as bytes. |
| `generate_stub`(path[, log_func]) | Create Stub from local .py file. |
| `get_cached_data`(url) | Wrap requests with a short cache. |
| `get_class_that_defined_method`(meth) | Determines Class that defined a given method. |
| `get_package_meta`(name, url) | Retrieve package metadata from PyPi. |
| `get_url_filename`(url) | Parse filename from url. |
| `is_dir_link`(path) | Test if path is either a symlink or directory junction. |
| `is_downloadable`(url) | Checks if the url can be downloaded from. |
| `is_existing_dir`(path) | Check if path is an existing directory. |
| `is_update_available`() | Check if micropy-cli update is available. |
| `is_url`(url) | Check if provided string is a url. |
| `iter_requirements`(path) | Iterate requirements from a requirements.txt file. |
| `lazy_property`(fn) | |
| `search_xml`(url, node) | Search xml from url by node. |
| `stream_download`(url, **kwargs) | Stream download with tqdm progress bar. |

#### Classes

| | |
|---|---|
| `Validator`(schema_path) | "jsonschema wrapper for file validation. |

### 1.6.9 micropy.config

Configuration files and interfaces for them.

#### Classes

| | |
|---|---|
| `Config`(*args[, source_format, default]) | Configuration File Interface. |
| `DictConfigSource`([config]) | |
| `JSONConfigSource`(path) | JSON Config File Source. |

### 1.6.10 micropy.config.config_source

Config Abstract.

#### Classes

| | |
|---|---|
| `Any`(*args, **kwargs) | Special type indicating an unconstrained type. |
| `ConfigSource`([initial_config]) | Abstract Base Class for Config Sources. |
| `Log`() | Borg for easy access to any Log from anywhere in the package. |
| `ServiceLog`([service_name, base_color]) | Handles logging to stdout and micropy.log. |

### 1.6.11 micropy.packages

Packages Module.

Allows user to address different dependency types (package, module, path, pypi, etc.) through a single uniform api.

#### Functions

| | |
|---|---|
| `create_dependency_source`(requirement[, name]) | Factory for creating a dependency source object. |

### Classes

| | |
|---|---|
| `Any`(*args, **kwargs) | Special type indicating an unconstrained type. |
| `LocalDependencySource`(package, path) | Dependency Source that is available locally. |
| `Package`(name, specs[, path, uri, vcs, ...]) | |
| `PackageDependencySource`(package[, format_desc]) | Dependency Source for pypi packages. |
| `Path`(*args, **kwargs) | PurePath subclass that can make system calls. |
| `VCSDependencySource`(package[, format_desc]) | Dependency Source for vcs packages. |

CHAPTER

# TWO

## INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

# INDEX