Micropython-Stubber

Release 1.17.2

Jos Verlinde

Feb 14, 2024

CONTENTS:

CHAPTER

ONE

BOOST MICROPYTHON PRODUCTIVITY IN VSCODE

The intellisense and code linting that is so prevalent in modern editors, does not work out-of-the-gate for MicroPython projects. While the language is Python, the modules used are different from CPython , and also different ports have different modules and classes , or the same class with different parameters.

Writing MicroPython code in a modern editor should not need to involve keeping a browser open to check for the exact parameters to read a sensor, light-up a led or send a network request.

Fortunately with some additional configuration and data, it is possible to make the editors understand your flavor of MicroPython. even if you run a on-off custom firmware version.

In order to achieve this a few things are needed:

1) Stub files for the native / enabled modules in the firmware using PEP 484 Type Hints
2) Specific configuration of the VSCode Python extensions
3) Specific configuration of Pylint
4) Suppression of warnings that collide with the MicroPython principals or code optimization.

With that in place, VSCode will understand MicroPython for the most part, and help you to write code, and catch more errors before deploying it to your board.

Note that the above is not limited to VSCode and pylint, but it happens to be the combination that I use.

A lot of subs have already been generated and are shared on github or other means, so it is quite likely that you can just grab a copy to be productive in a few minutes.

For now you will need to *configure this by hand*, or use the `micropy cli` *tool*

1. The sister-repo [**MicroPython-stubs**][stubs-repo] contains [all stubs][all-stubs] I have collected with the help of others, and which can be used directly. That repo also contains examples configuration files that can be easily adopted to your setup.
2. A second repo [micropy-stubs repo][stubs-repo2] maintained by BradenM, also contains stubs but in a structure used and distributed by the *micropy-cli* tool. you should use micropy-cli to consume stubs in this repo.

The (stretch) goal is to create a VSCode add-in to simplify the configuration, and allow easy switching between different firmwares and versions.

1.1 Licensing

MicroPython-Stubber is licensed under the MIT license, and all contributions should follow this LICENSE.

CHAPTER

TWO

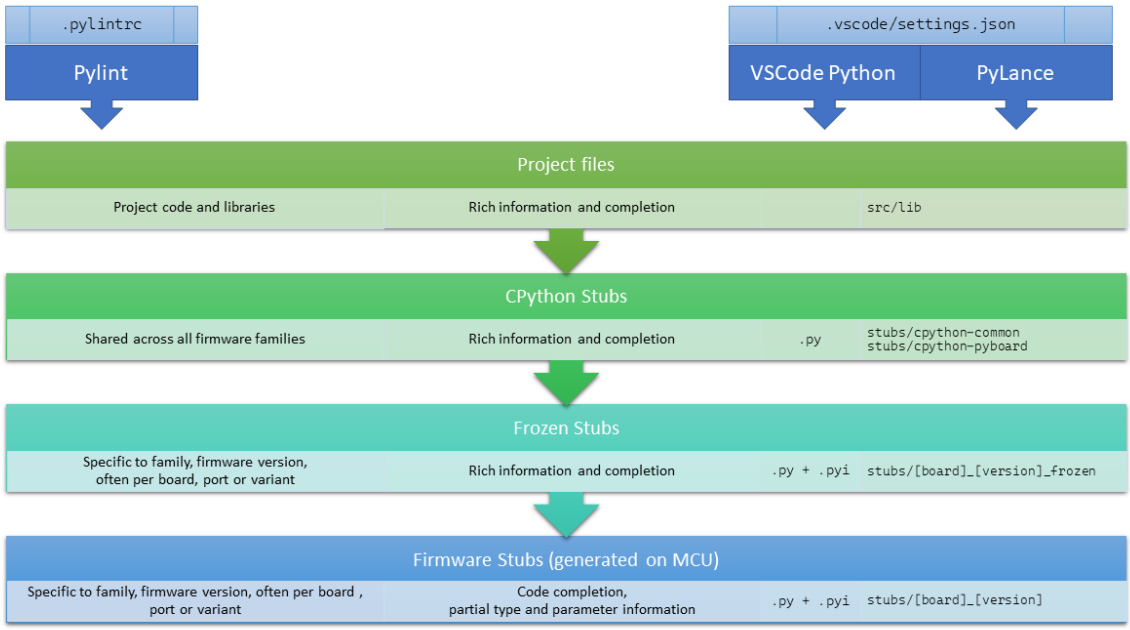
APPROACH TO COLLECTING STUB INFORMATION

The stubs are used by 3 components.

1. the VSCode Pylance Language Server
2. the VSCode Python add-in
3. a linter such as pylint

These 3 tools work together to provide code completion/prediction, type checking and all the other good things. For this the order in which these tools use, the stub folders is significant, and best results are when all use the same order.

In most cases the best results are achieved by the below setup:



[1]

1. **Your own source files**, including any libraries you add to your project. This can be a single libs folder or multiple directories. There is no need to run stubber on your source or libraries.
2. **The CPython common stubs**. These stubs are handcrafted to allow MicroPython script to run on a CPython system. There are only a limited number of these stubs and while they are not intended to be used to provide type hints, they do provide valuable information. Note that for some modules (such as the `gc`, `time` and `sys` modules) this approach does not work.

3. **Frozen stubs**. Most micropython firmwares include a number of python modules that have been included in the firmware as frozen modules in order to take up less memory. These modules have been extracted from the source code.
4. **Firmware Stubs**. For all other modules that are included on the board, [micropython-stubber] or [micropy-cli] has been used to extract as much information as available, and provide that as stubs. While there is a lot of relevant and useful information for code completion, it does unfortunately not provide all details regarding parameters that the above options may provide.

2.1 Stub collection process

- The **CPython common stubs** are periodically collected from the [micropython-lib][] or the [pycopy-lib][].
- The **Frozen stubs** are collected from the repos of [micropython][] + [micropython-lib][] and from the [loboris][] repo the methods to gather these differs per firmware family , and there are differences between versions how these are stored , and retrieved. where possible this is done per port and board, or if not possible the common configuration for has been included.
- the **Firmware stubs** are generated directly on a MicroPython board.

2.2 Firmware Stubs format and limitations

1. No function parameters are generated
2. No return types are generated
3. Instances of imported classes have no type (due to 2)
4. The stubs use the .py extension rather than .pyi (for autocomplete to work)
5. Due to the method of generation nested modules are included, rather than referenced. While this leads to somewhat larger stubs, this should not be limiting for using the stubs on a PC.

2.3 Stub naming convention

The firmware naming conventions is most relevant to provide clear folder names when selecting which stubs to use.

for stubfiles: {**firmware family**}-{version}-{port}

for frozen modules : {firmware}-{version}-frozen{port}{board}

- ***firmware family***: lowercase
  - micropython | loboris | pycopy | ...
- ***port***: lowercase , as reported by os.implementation.platform
  - stm32 | esp32 | linux | win32 | rp2 | samd | ...
- ***board***: used mainly for frozen stubs
  - GENERIC | RELEASE | UM_TINYPICO | GENERIC_512K | ARDUINO_NANO_RP2040_CONNECT | ...

Note: RELEASE appears to be used mainly for CI/CD purposes and is not commonly used on hardware.

- ***version*** : digits only , dots replaced by underscore, follow version in documentation rather than semver

- v1_13
- v1_9_4
- ***build***, only for nightly build, the build nr. extracted from the git tag
  - Nothing , for released versions
  - 103
  - Latest

CHAPTER

THREE

STUBBER

```
stubber [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...
```

Options

--version
Show the version and exit.

-v, --verbose
-v for DEBUG, -v -v for TRACE

3.1 build

Commandline interface to publish stubs.

```
stubber build [OPTIONS]
```

Options

--family <family>

Default
micropython

-V, --version, --Version <versions>
MicroPython version to build, or 'preview' for the latest preview version

Default
v1.22.1

-p, --port <ports>
multiple:

Default
all

-b, --board <boards>
multiple:

Default
GENERIC

--clean
clean folders after processing and publishing

--force
build package even if no changes detected

3.2 clone

Clone/fetch the micropython repos locally.

The local repos are used to generate frozen-stubs and doc-stubs.

```
stubber clone [OPTIONS]
```

Options

-p, --path <path>

--add-stubs, --no-stubs
Also clone the micropython-stubs repo

3.3 enrich

Enrich the stubs in stub_folder with the docstubs in docstubs_folder.

```
stubber enrich [OPTIONS]
```

Options

-s, --stubs <stubs_folder>
File or folder containing the firmware stubs to be updated

Default
repos/micropython-stubs/stubs

-ds, --docstubs <docstubs_folder>
File or folder containing the docstubs to be applied

Default
repos/micropython-stubs/stubs

--diff
Show diff

Default
False

--dry-run
Dry run does not write the files back

Default
False

-p, --package-name <package_name>

Package name to be enriched (Optional)

Default

3.4 get-core

Download core CPython stubs from PyPi.

Get the core (CPython compat) modules for both MicroPython and Pycopy.

```
stubber get-core [OPTIONS]
```

Options

-stubs, --stub-folder <stub_folder>

Default
repos/micropython-stubs/stubs

--stubgen, --no-stubgen

run stubgen to create .pyi files for the (new) frozen modules

Default
True

--black, --no-black

Run black on the (new) frozen modules

Default
True

3.5 get-docstubs

Build stubs from documentation.

Read the Micropython library documentation files and use them to build stubs that can be used for static typechecking.

```
stubber get-docstubs [OPTIONS]
```

Options

-p, --path <path>

Default
repos

--stub-path, --stub-folder <target>

Destination of the files to be generated.

Default
repos/micropython-stubs/stubs

--version, **--tag** <version>

Version number to use. [default: Git tag]

-b, **--black**, **-nb**, **--no-black**

Run black

Default
`True`

3.6 get-frozen

Get the frozen stubs for MicroPython.

Get the frozen modules for the checked out version of MicroPython

```
stubber get-frozen [OPTIONS]
```

Options

-stubs, **--stub-folder** <stub_folder>

Default
`repos/micropython-stubs/stubs`

-V, **--version**, **--Version** <version>

The version of MicroPython to get the frozen modules for. Use 'preview' to get the latest version from the micropython repo

Default

--stubgen, **--no-stubgen**

Run stubgen to create .pyi files for the (new) frozen modules

Default
`True`

--black, **--no-black**

Run black on the (new) frozen modules

Default
`True`

3.7 get-lobo

Get the frozen stubs for Lobo-esp32.

```
stubber get-lobo [OPTIONS]
```

Options

-stubs, **--stub-folder** <stub_folder>

Default
repos/micropython-stubs/stubs

--pyi, **--no-pyi**

Create .pyi files for the (new) frozen modules

Default
True

--black, **--no-black**

Run black on the (new) frozen modules

Default
True

3.8 make-variants

Update all variants of createstubs*.py.

```
stubber make-variants [OPTIONS]
```

Options

-t, **--target** <target_folder>

Target folder for the createstubs*.py/.mpy files

-V, **--version**, **--Version** <version>

The version of mpy-cross to use

Default
v1.22.1

3.9 merge

Enrich the stubs in stub_folder with the docstubs in docstubs_folder.

```
stubber merge [OPTIONS]
```

Options

--family <family>

Default
micropython

-V, --version, --Version <versions>

'latest', 'auto', or one or more versions

Default
all

-p, --port <ports>

multiple:

Default
all

-b, --board <boards>

multiple:

Default
generic

3.10 minify

Minify createstubs*.py.

Creates a minified version of the SOURCE micropython file in TARGET (file or folder). The goal is to use less memory / not to run out of memory, while generating Firmware stubs.

```
stubber minify [OPTIONS]
```

Options

-s, --source <source>

Default
createstubs.py

-d, --diff

Show the functional changes made to the source script.

-c, -xc, --compile

Cross compile after minification.

-a, --all

Minify all variants (normal, _mem and _db).

--report, --no-report

Keep or disable minimal progress reporting in the minified version.

Default
True

3.11 publish

Commandline interface to publish stubs.

```
stubber publish [OPTIONS]
```

Options

--family <family>

Default
micropython

-V, **--version**, **--Version** <versions>

multiple:

Default
v1.22.1

-p, **--port** <ports>

multiple:

Default
all

-b, **--board** <boards>

multiple:

Default
GENERIC

--pypi, **--test-pypi**

publish to PYPI or Test-PYPI

Default
False

--build

build before publish

--force

create new post release even if no changes detected

--dry-run

Do not actually publish, just show what would be done

--clean

clean folders after processing and publishing

3.12 show-config

Show the current configuration

```
stubber show-config [OPTIONS]
```

3.13 stub

Create or update .pyi type hint files.

```
stubber stub [OPTIONS]
```

Options

-s, --source <source>

3.14 switch

Switch to a specific version of the micropython repos.

The Micropython-lib repo will be checked out to a commit that corresponds in time to that version tag, in order to allow non-current versions to be stubbed correctly.

The repros must be cloned already

```
stubber switch [OPTIONS] [[v1.10|v1.11|v1.12|v1.13|v1.14|v1.15|v1.16|v1.17|v1.
               18|v1.19|v1.19.1|v1.20.0|v1.21.0|v1.22.0|v1.22.0-preview|v1.22.
               1|v1.23.0-preview|v1.9.3|v1.9.4|preview|latest]]
```

Options

-p, --path <path>

Arguments

TAG

Optional argument

3.15 update-fallback

Update the fallback stubs.

The fallback stubs are updated/collated from files of the firmware-stubs, doc-stubs and core-stubs.

```
stubber update-fallback [OPTIONS]
```

Options

--version <version>

Version number to use

Default

v1_18

--stub-folder <stub_folder>

Destination of the files to be generated.

Default

repos/micropython-stubs/stubs

3.16 update-module-list

Update the module list based on the information in the data folder

```
stubber update-module-list [OPTIONS]
```

CHAPTER

FOUR

USING MICROPYTHON STUBBER

This section describes ho to use micropython-stubber to create and maintain stubs for a MicroPython firmware or project.

If you just want to use the stubs, then you can skip this section and instead read the documentation in the sister-repo [micropython-stubs][] section [using-the-stubs][]

4.1 Manual configuration

The manual configuration, including sample configuration files is described in detail in the sister-repo [micropython-stubs][] section [using-the-stubs][]

4.2 using micropython-stubber

You can install micropython stubber from PyPi using `pip install micropython-stubber`.

4.3 Using micropy-cli

'micropy-cli' is command line tool for managing MicroPython projects with VSCode If you want a command line interface to setup a new project and configure the settings as described above for you, then take a look at : [micropy-cli]

```
pip install micropy-cli
micropy init
```

Braden has essentially created a front-end for using micropython-stubber, and the configuration of a project folder for pymakr.

micropy-cli maintains its own repository of stubs.

CHAPTER

FIVE

CREATE FIRMWARE STUBS

It is possible to create MicroPython stubs using the `createstubs.py` MicroPython script.

the script goes though the following stages

1. it determines the firmware family, the version and the port of the device, and based on that information it creates a firmware identifier (fwid) in the format : {family}-{port}-{version} the fwid is used to name the folder that stores the stubs for that device.
    - stubs/micropython-v1_10-stm32
    - stubs/micropython-v1_12-esp32
    - stubs/loboris-v3_2_4-esp32
2. it cleans the stub folder
3. it generates stubs, using a predetermined list of module names. for each found module or submodule a stub file is written to the device and progress is output to the console/repl.
4. a module manifest (`modules.json`) is created that contains the pertinent information determined from the board, the version of createstubs.py and a list of the successful generated stubs

Module duplication

Due to the module naming convention in micropython some modules will be duplicated , ie `uos` and `os` will both be included

5.1 Running the script

The `createstubs.py` script can either be run as a script or imported as a module depending on your preferences.

Running as a script is used on the linux or win32 platforms in order to pass a –path parameter to the script.

The steps are :

1. Connect to your board
2. Upload the script(s) to your board. All variants of the script are located in the `/board` folder of this repo
3. Run/import the `createstubs.py` script
4. Download the generated stubs to a folder on your PC
5. run the post-processor [optional, but recommended]

![createstubs-flow][]

Note: There is a memory allocation bug in MicroPython 1.30 that prevents createstubs.py to work. this was fixed in nightly build v1.13-103 and newer.

If you try to create stubs on this defective version, the stubber will raise *NotImplementedError*("MicroPython 1.13.0 cannot be stubbed")

5.2 Generating Stubs for a specific Firmware

The stub files are generated on a MicroPython board by running the script `createstubs.py`, this will generate the stubs on the board and store them, either on flash or on the SD card. If your firmware does not include the `logging` module, you will need to upload this to your board, or use the minified version.

```
import createstubs
```

The generation will take a few minutes (2-5 minutes) depending on the speed of the board and the number of included modules.

As the stubs are generated on the board, and as MicroPython is highly optimized to deal with the scarce resources, this unfortunately does mean that the stubs lacks parameters details. So for these you must still use the documentation provided for that firmware.

5.3 Downloading the files

After the sub files have been generated , you will need to download the generated stubs from the micropython board and most likely you will want to copy and save them on a folder on your computer. if you work with multiple firmwares, ports or version it is simple to keep the stub files in a common folder as the firmware id is used to generate unique names

- ./stubs
  - /micropython-v1_10-stm32
  - /micropython-v1_12-esp32
  - /micropython-v1_11-linux
  - /loboris-v3_1_20-esp32
  - /loboris-v3_2_24-esp32

5.4 Custom firmware

The script tries to determine a firmware ID and version from the information provided in `sys.implementation` , `sys.uname()` and the existence of specific modules..

This firmware ID is used in the stubs , and in the folder name to store the subs.

If you need, or prefer, to specify a firmware ID you can do so by setting the firmware_id variable before importing createstubs For this you will need to edit the createstubs.py file.

The recommendation is to keep the firmware id short, and add a version as in the below example.

```
# almost at the end of the file
def main():
    stubber = Stubber(firmware_id='HoverBot v1.2.1')
    # Add specific additional modules to be stubbed
```

(continues on next page)

(continued from previous page)

```
    stubber.add_modules(['hover','rudder'])
```

after this , upload the file and import it to generate the stubs using your custom firmware id.

5.5 The Unstubbables

There are a limited number of modules that cannot be stubbed by createstubs.py for a number of different reasons. Some simply raise errors , others my reboot the MCU, or require a specific configuration or state before they are loaded.

a few of the frozen modules are just included as a sample rather \t would not be very useful to generate stubs for these

the problematic category throw errors or lock up the stubbing process altogether:

```
self.problematic=["upysh","webrepl_setup","http_client","http_client_ssl","http_server",
↪"http_server_ssl"]
```

The excluded category provides no relevant stub information

```
self.excluded=["webrepl","_webrepl","port_diag","example_sub_led.py","example_pub_
↪button.py"]
```

`createstubs.py` will not process a module in either category.

Note: that some of these modules are also included in the frozen modules that are gathered for those ports or boards. For those modules it makes sense to use/prioritize the .pyi stubs for the forzen modules over the firmware stubs.

CHAPTER

SIX

USING CREATESTUB

6.1 Use MIP to install createstubs on a MCU board

It is possible to install the firmware stubber (createstubs.py or one of its variants) on a MicroPython board. This allows you to create the firmware stubs on the board itself, and then copy them to your PC.

mip is a package manager for MicroPython. It is a command line tool that allows you to install packages on a MicroPython board. In this case it is best to use `mpremote` that has a built-in `mip` command.

Make sure you have the latest version of `mpremote` installed.

```
pip install mpremote
```

6.2 Install createstubs

Connect your board to your PC and run the following command:

`mpremote mip install github:josverl/micropython-stubber`

```
Install github:josverl/micropython-stubber
Installing github:josverl/micropython-stubber/package.json to /lib
Installing: /lib/createstubs.py
Installing: /lib/createstubs_db.py
Installing: /lib/createstubs_mem.py
Installing: /lib/modules.txt
Done
```

6.3 run createstubs

A simple way to run createstubs is to use the `mpremote mount` command to allow the MCU board to directly access the PC's file system. Then you can run the createstubs.py script directly from the MCU board with outh the need to copy the created files back to the PC.

Navigate to the folder where you want to create the stubs and run the following command: `mpremote mount . exec "import createstubs"` or `mpremote mount . exec "import createstubs_mem"` or

6.4 low memory devices

If you have a low memory board, then you can install the cross-compiled variants to reduce the memory footprint durign compilation on the board:

MicroPython release	.mpy version	command
v1.19 and up	6	`mpremote mip install github:josverl/micropython-stubber/mpy_v6.json`
v1.12 - v1.18	5	`mpremote mip install github:josverl/micropython-stubber/mpy_v5.json`

```
Install github:josverl/micropython-stubber/mpy_v6.json
Installing github:josverl/micropython-stubber/mpy_v6.json to /lib
Installing: /lib/createstubs_mpy.mpy
Installing: /lib/createstubs_db_mpy.mpy
Installing: /lib/createstubs_mem_mpy.mpy
Installing: /lib/modulelist.txt
```

Note: The names of the scripts have changed to createstubs_mpy.py, createstubs_db_mpy.py and createstubs_mem_mpy.py

Navigate to the folder where you want to create the stubs and run the following command: `mpremote mount . exec "import createstubs_db_mpy"` or `mpremote mount . exec "import createstubs_mem_mpy"`

6.5 format the stubs and generate the .pyi files

The stubs are generated in a folder called `stubs` in the current folder. You can use stubber to run the `stubgen` tool to format the stubs and generate the .pyi files.

For example:

```
stubber stub -s ./stubs/micropython-v1_19_1-rp2
```

see MicroPython Package management

CHAPTER

SEVEN

CREATESTUB VARIANTS

There are multiple variants of the script available, in 3 levels of optimisation:

variant	full documented script	minified script (no logging)	cross-compiled script
full version	createstubs.py	createstubs_min.py	createstubs_mpy.mpy
memory optimized	createstubs_mem.py	createstubs_mem_min.py	createstubs_mem_mpy.mpy
restartable	createstubs_db.py	createstubs_db_min.py	createstubs_db_mpy.mpy
lvgl only	createstubs_lvgl.py	createstubs_lvgl_min.py	createstubs_lvgl_mpy.mpy

all file are located in the repo in 'src/stubber/board' and included in the stubber package as stubber.board.*

In all cases the generation will take a few minutes (2-5 minutes) depending on the speed of the board and the number of included modules. As the stubs are generated on the board, and as MicroPython is highly optimized to deal with the scarce resources, this unfortunately does mean that the stubs lacks parameters details. So for these you must still use the documentation provided for that firmware.

The variants can be updated / regenerated using the `stubber make-variants` command

```
stubber make-variants

Usage: stubber make-variants [OPTIONS]

  Update all variants of createstubs*.py.

Options:
  -V, --version, --Version TEXT  The version of mpy-cross to use  [default: v1.19.1]
```

7.1 board/createstubs.py

This is the core version of the script, and is fully self contained, but includes logging with requires the logging module to be avaialble on your device If your firmware does not include the `logging` module, you will need to upload this to your board, or use the minified version.

```
import createstubs
```

7.2 createstubs_mem.py

This variant of the createstubs.py script is optimised for use on low-memory devices, and reads the list of modules from a text file, rather than including it in the source file. as a result this requires an additional file `./modulelist.txt`, that should be uploaded to the device. If that cannot be found then only a single module (micropython) is stubbed.

```
import createstubs_mem
```

7.3 Optimisations

In order to run this on low-memory devices two additional steps are recommended:

- Minifification, using python-minifier to reduce overall size, and remove logging overhead. can be used on all devices
- Cross compilation, using mpy-cross, to avoid the compilation step on the micropython device. The cross-compiled version can only run on specific Micropython 1.12 or newer.

7.3.1 Minification

Minified versions, which requires less memory and only very basic logging. this removes the requirement for the logging module on the MCU.

Minifiacation helps reduce the seize of the script, and therefore of the memory usage. As a result the script becomes almost unreadable.

7.3.2 Cross compilation

This is specially suited for low memory devices such as the esp8622

CHAPTER

EIGHT

CPYTHON AND FROZEN MODULES

8.1 Frozen Modules

It is common for Firmwares to include a few (or many) python modules as 'frozen' modules. 'Freezing' modules is a way to pre-process .py modules so they're 'baked-in' to MicroPython' s firmware and use less memory. Once the code is frozen it can be quickly loaded and interpreted by MicroPython without as much memory and processing time.

Most OSS firmwares store these frozen modules as part of their repository, which allows us to:

1. Download the *.py from the (github) repo using `git clone` or a direct download
2. Extract and store the 'unfrozen' modules (ie the *.py files) in a _Frozen folder. if there are different port / boards or releases defined , there may be multiple folders such as:
   - stubs/micropython_1_12_frozen
     - /esp32
       - /GENERIC
       - /RELEASE
       - /TINYPICO
     - /stm32
       - /GENERIC
       - /PYBD_SF2
3. Generate typeshed stubs of these files. *const pre-processing:* As the mypy.stubgen tool is not able to incur the correct types from the MicroPython `foo = const(1)` syntax, the 'to be frozen' modules are pre-processed usig a regular expression to replace the `foo = const(1)` with `foo = 1`. If the `.py` files contain any docstrings, they are preserved. Howecer this is uncommon as most micropython-lib modules have not docstrings to save space.

   Addition of docstrings: Then the docstring to modules, classes and methods are added by merging the docstrings based on the docstubs generated from the MicroPython documentation.

   Finally the stubs are generated using the `stubgen` tool. The resulting .pyi files are stored alongside the .py files
4. Include/use them in the configuration

ref: https://learn.adafruit.com/micropython-basics-loading-modules/frozen-modules

8.2 Collect Frozen Stubs (micropython)

This is run daily though the github action workflow : get-all-frozen in the micropython-stubs repo.

If you want to run this manually

- Check out repos side-by-side:
  - micropython-stubs
  - micropython-stubber
  - micropython
  - micropython-lib
- link repos using all_stubs symlink
- checkout tag / version in the micropython folder
  (for most accurate results should checkout micropython-lib for the same date)
- run `get-frozen`
- run `update_stub`
- create a PR for changes to the stubs repo

8.3 Postprocessing

You can run postprocessing for all stubs by running either of the two scripts. There is an optional parameter to specify the location of the stub folder. The default path is `./all_stubs`

```
update_stubs [./mystubs]
```

This will generate or update the `.pyi` stubs for all new (and existing) stubs in the `./all_stubs` or specified folder.

From version '1.3.8' the `.pyi` stubs are generated using `stubgen`, before that the `make_stub_files.py` script was used.

Stubgen is run on each 'collected stub folder' (that contains a `modules.json` manifest) using the options : `--ignore-errors --include-private` and the resulting `.pyi` files are stored in the same folder (`foo.py` and `foo.pyi` are stored next to each other).

In some cases `stubgen` detects duplicate modules in a 'collected stub folder', and subsequently does not generate any stubs for any `.py` module or script. then **Plan B** is to run stubgen for each separate `*.py` file in that folder. While this is significantly slower and according to the stubgen documentation the resulting stubs may of lesser quality, but that is better than no stubs at all.

Note: In several cases `stubgen` creates folders in inappropriate locations (reason undetermined), which would cause issues when re-running `stubgen` at a later time. to compensate for this behaviour the known-incorrect .pyi files are removed before and after stubgen is run see: `cleanup(modules_folder)` in `utils.py`

CHAPTER

NINE

DOCUMENTATION STUBS

9.1 What / Why

Advantages : they bring the richness of the MicroPython documentation to Pylance. This includes function and method parameters and descriptions, the module and class constants for all documented library modules.

9.2 How docstubs are generated

The documentation stubs are generated using the `stubber get-docstubs` command.

1) Read the Micropython library documentation files and use them to build stubs that can be used for static type-checking using a custom-built parser to read and process the micropython RST files

- This will generate :
  - Python modules (`<module.py>`), one for each `<module>.rst` file
    - The module docstring is based on the module header in the .rst file
  - Function definitions
    - Function parameters and types based on documentation As the parameter documentaion is sometimes rather abigious or imprecise, the parameters definities are cleaned up based on a hand tuned algoritm
    - The function docstring is based on the function description in the .rst file
    - The return type of a function is based on phrases used in the documentation, with an override table for functions with insufficient documented information to determine the return type.
  - Classes
    - The class docstring is based on the Class description in the .rst file
    - **init** method
      - The init parameters are based on the documentation for the class As the parameter documentaion is sometimes rather abigious or imprecise, the parameters definities are cleaned up based on a hand tuned algoritm
      - **init** docstring is based on the Class description in the .rst file
    - Methods
      - Method parameters and types based are based on the documentation in the .rst file As the parameter documentaion is sometimes rather abigious or imprecise, the parameters definities are cleaned up based on a hand tuned algoritm

        - The method docstring is based on the method description in the .rst file
        - The return type of a method is based on phrases used in the documentation, with an override table for functions with insufficient documented information to determine the return type.
        - Method decorators @classmethod and @staticmethod are generated based on the use of `py:staticmethod` or `py:classmethod` in the documentation. ref: https://sphinx-tutorial.readthedocs.io/cheatsheet/
        - Method parameter names `self` and `cls` are used accordingly.
    - Exceptions

9.2.1 Return types

- Tries to determine the return type by parsing the docstring.
    - Imperative verbs used in docstrings have a strong correlation to return -> None
    - Recognizes documented Generators, Iterators, Callable
    - Coroutines are identified based tag "This is a Coroutine". Then if the return type was Foo, it will be transformed to : Coroutine[Foo]
    - A static Lookup list is used for a few methods/functions for which the return type cannot be determined from the docstring.
    - add NoReturn to a few functions that never return (stop / deepsleep / reset)
    - if no type can be detected the type `Any` is used

9.2.2 Lookup tables :

- *src/stubber/rst/lookup.py*
    - LOOKUP_LIST
        - contains return types for functions and methods
        - "module.[class.].function" : ("type", probability)
    - NONE_VERBS
        - if no type has been determined, and the docstring starts with one of these verbs, then assume the return type is None
    - MODULE_GLUE
        - Add additional imports to some modules to allow one module to import other supporting modules
        - currently only used for `lcd160cr` and `esp32`
    - PARAM_FIXES
        - manual fixes needed for parameters (micropython v.16 & v1.17)
        - used to clean up the parameter strings before further interpetation using search and replace
    - CHILD_PARENT_CLASS
        - List of classes and their parent classes that should be added to the class definition. The documentation contains no clear references of the parent classes so these can only be added based on a (manual) lookup table

* Note: The parent class **Exeptions** is determined based on the rst hint `py:exception` and the Class Name.

9.2.3 Code Formatting

The generated stub files (`.py`) are formatted using `black` and checked for validity using `pyright`

Note: black on python 3.7 does not like some function defs, this is not treated as an error. `def sizeof(struct, layout_type=NATIVE, /) -> int:`

9.2.4 Ordering of inter-dependent classes in the same module

Classes are frequently documented in a different order than thery need to be declared in a source file. To accomodate for this the source code is re-ordered to avoid forward references in the code. The code for this is located in :

- *src/stubber/rst/classsort.py*
- *src/stubber/rst/output_dict.py*

9.2.5 Add GLUE imports to allow specific modules to import specific others.

This is based on the `MODULE_GLUE` table to support some modules that need to import other modules or classes.

9.2.6 Literals / constants

```
- documentation contains repeated vars with the same indentation
- Module level:
.. code-block::

    .. data:: IPPROTO_UDP
              IPPROTO_TCP

- class level:
.. code-block::

    .. data:: Pin.IRQ_FALLING
              Pin.IRQ_RISING
              Pin.IRQ_LOW_LEVEL
              Pin.IRQ_HIGH_LEVEL

              Selects the IRQ trigger type.

- literals documented using a wildcard are added as comments only
```

- Repeats of definitions in the rst file for similar functions or literals
  - CONSTANTS (module and Class level)
  - functions
  - methods

CHAPTER

TEN

REPO STRUCTURE

- *This and sister repos*
- *Structure of this repo*
- *Naming Convention and Stub folder structure*
- 2 python versions

10.1 This and sister repos

repo	Why	Where	example
micropython-stubber	needed to make stubs	in your source folder	develop/micropython-stubber
micropython-stubs	stores collected stubs	next to the `stubber`	develop/micropython-stubs

Note:

- recommended is to create a symlink from `develop/micropython-stubber\all-stubs` to `develop/micropython-stubs`

10.2 Structure of this repo

The file structure is based on my personal windows environment, but you should be able to adapt that without much hardship to you own preference and OS.

What	Details	Where
stub root	symlink to connect the 2 sister-repos	all_stubs
firmware stubber	MicroPython	board/createstubs.py
minified firmware stubber	MicroPython	minified/createstubs.py
PC based scripts	CPython	src/*
PC based scripts	CPython	process.py
pytest tests		test/*

10.3 Naming Convention and Stub folder structure

What	Why	Where
stub root	connect the 2 repos	all_stubs
cpython stubs for micropython core	adapt for differences between CPython and MicroPython	stubs/cpython-core
generated stub files	needed to use stubs	stubs/{firmware}-{port}-{version}-frozen
Frozen stub files	better code intellisense	stubs/{firmware}-{version}-frozen

Note: I found that, for me, using submodules caused more problems than it solved. So instead I link the two main repo's using a *symlink*.

```
cd /develop

git clone  https://github.com/josverl/micropython-stubber.git
git clone  https://github.com/josverl/micropython-stubs.git

cd micropython-stubber
poetry install

stubber clone
```

10.4 Create a symbolic link

To create the symbolic link to the `../micropython-stubs/stubs` folder the instructions differ slightly for each OS/ The below examples assume that the micropython-stubs repo is cloned 'next-to' your project folder. please adjust as needed.

10.4.1 Windows 10

Requires `Developer enabled` or elevated powershell prompt.

```
# target must be an absolute path, resolve path is used to resolve the relative path to␣
↪absolute
New-Item -ItemType SymbolicLink -Path "all-stubs" -Target (Resolve-Path -Path ../
↪micropython-stubs/stubs)
```

or use mklink in an (elevated) command prompt

```
rem target must be an absolute path
mklink /d all-stubs c:\develop\micropython-stubs\stubs
```

10.4.2 Linux/Unix/Mac OS

```
# target must be an absolute path
ln -s /path/to/micropython-stubs/stubs all-stubs
```

CHAPTER

ELEVEN

CODEMODS

11.1 What are codemods

Codemods are a way to modify the codebase in a way that is not a breaking change. Codemods are built on [LibCST][libcst] and are written in [Python][python].

Simply said codemods allow one to manipulate python sourcecode (.py and .pyi) files in a structured way withouth the complexities and limitations of using regexes.

11.2 Provided codemods

You can list the codemods that are provided by micopython stubber using the `libcst.tool`'s `list` command.

```
python -m libcst.tool list
```

- add_comment.AddComment - Add comment(s) to each file
- merge_docstub.MergeCommand - Merge the type-rich information from a doc-stub into a firmware stub

To run a codemos use the `codemod` command:

```
python -m libcst.tool codemod <codemod.name> arguments ...
```

examples:

- `python -m libcst.tool codemod add_comment.AddComment --help`
  Get help on the add_comment codemod

11.3 add_comment.AddComment codemod

`Addcomment` is used to add comments to frozen modudels and modules that are copied from other sources in order to clarify their origin.

examples:

- `python -m libcst.tool codemod add_comment.AddComment --help`
  Get help on the add_comment codemod
- `python -m libcst.tool codemod add_comment.AddComment --comment="This is a comment" ./module.py`
  Add a comment to the module.py file.
  `--comment` can be specified multple times to add more comment lines.

The comment will be added below any existing comments at the top of the file, and will be prefixed with "# " if it is not already present.
If the first comment line already exists in the source code, no comments will be added

- `python -m libcst.tool codemod add_comment.AddComment --include-stubs --comment="MicroPython 1.18 frozen modules" ./stubs/micropython-v1_18-frozen`
  Add a comment to all the .py and .pyi files in the frozen module folder.

11.4 How to run a codemode from the commandline

merge_docstub.MergeCommand

examples:

- `python -m libcst.tool codemod merge_docstub.MergeCommand --help`
  Get help on the add_comment codemod

11.5 Where are codemods used

- To merge the type-rich information from a doc-stub into a firmware stub
- To create the different variants of `createstubs.py` `stubber make-variants` This will use moding the code to:
  - a memory efficient version (low memory)
  - a version that allows the MCU to restart without losing the progress (very low memory)

CHAPTER

TWELVE

POWERSHELL SCRIPTS

A number of scripts have been written in PowerShell as that is one of my preferred scripting languages. Possibly these scripts could be ported to python , at the cost of more complex handling of OS processes and paths and ports.

(a PR with a port to Python would be appreciated)

12.1 bulk_stubber.ps1

The goal of this script is to run create_stubs on a set of boards connected to my machine in order to generate new stubs for multiple micropython versions

high level operation:

- Scans the serial ports for connected esp32 and esp8266 devices using `get-serialport.ps1 -chip`
- Uses a (hardcoded) list of firmwares including version + chip type
- for each firmware in that list:
  - Selects the corresponding device and serialport
  - Flashes the micropython version to the device using `flash_MPY.ps1`
  - waits for the device to finish processing any initial tasks (file system creation etc)

```
rshell -p $serialport  --rts 1 repl "~ print('connected') ~"
```

> **Note:** This is quite sensitive to timing and requires some delays to allow the device to restart before the script continues.
>
> Also a bit of automated manipulation of the RTS (and DTR) signals is needed to avoid needing to press a device's reset button.

  - Starts the minified version of createstubs.py

```
$createstubs_py = join-path $WSRoot "minified/createstubs.py"
pyboard --device $serialport $createstubs_py | write-host
```

  - Downloads the generated machine-stubs

```
# reverse sync
# $dest = path relative to current directory
# $source = path on board ( all boards are called pyboard)
```

(continues on next page)

(continued from previous page)

```
$source = "/pyboard/stubs"
rshell -p $serialport --buffer-size 512 rsync $source $subfolder  | write-host
```

12.1.1 Minificantion and compilation

in order to allow createstubs to be run on low-memory devices there are a few steps needed to allow for sufficient memory

12.1.2 Requirements & dependencies

Python

- esptool - to flash new firmware to the esp32 and esp8266
- pyboard.py - to upload files and run commands (not the old version on PyPi)
- rshell - to download the folder with stubs

PowerShell ../../Firmware

- get-serialport.ps1
- flash_MPY.ps1

12.1.3 Hardware

- ESP32 board + SPIRAM on USB + Serial drivers
- ESP8266 board on USB + Serial drivers

Note: Multiple boards can be connected at the same time. The script will select the first board of the corresponding type. If a board-type is not present, then no stubs for that device type will be generated.

CHAPTER

THIRTEEN

OVERVIEW OF STUBS

Initially I also stored all the generated subs in the same repo. That turned out to be a bit of a hassle and since then I have moved all the stubs to the micropython-stubs repo

Below are the most relevant stub sources referenced in this project.

13.1 Firmware and libraries

13.1.1 MicroPython firmware and frozen modules *[MIT]*

https://github.com/micropython/micropython

https://github.com/micropython/micropython-lib

13.1.2 Pycopy firmware and frozen modules *[MIT]*

https://github.com/pfalcon/pycopy

https://github.com/pfalcon/pycopy-lib

13.1.3 LoBoris ESP32 firmware and frozen modules *[MIT, Apache 2]*

https://github.com/loboris/MicroPython_ESP32_psRAM_LoBo

13.2 Included custom stubs

Github repo	Contributions	License
pfalcon/micropython-lib	CPython backports	MIT
dastultz/micropython-pyb	a pyb.py file for use with IDEs in developing a project for the Pyboard	Apache 2

13.2.1 Stub source: MicroPython-lib > CPython backports *[MIT, Python]*

While micropython-lib focuses on MicroPython, sometimes it may be beneficial to run MicroPython code using CPython, e.g. to use code coverage, debugging, etc. tools available for it. To facilitate such usage, micropython-lib also provides re-implementations ("backports") of MicroPython modules which run on CPython. https://github.com/pfalcon/micropython-lib#cpython-backports

13.2.2 micropython_pyb *[Apache 2]*

This project provides a pyb.py file for use with IDEs in developing a project for the Pyboard. https://github.com/dastultz/micropython-pyb

CHAPTER

FOURTEEN

REFERENCES

14.1 Inspiration

14.1.1 Thonny - MicroPython _cmd_dump_api_info *[MIT License]*

The `createstubs.py` script to create the stubs is based on the work of Aivar Annamaa and the Thonny crew. It is somewhere deep in the code and is apparently only used during the development cycle but it showed a way how to extract/generate a representation of the MicroPython modules written in C

While the concepts remain, the code has been rewritten to run on a micropython board, rather than on a connected PC running CPython. Please refer to : Thonny code sample

14.1.2 MyPy Stubgen

MyPy stubgen is used to generate stubs for the frozen modules and for the `*.py` stubs that were generated on a board.

14.1.3 make_stub_files *[Public Domain]*

https://github.com/edreamleo/make-stub-files

This script `make_stub_files.py` makes a stub (.pyi) file in the output directory for each source file listed on the command line (wildcard file names are supported).

The script does no type inference. Instead, the user supplies patterns in a configuration file. The script matches these patterns to: The names of arguments in functions and methods and The text of return expressions. Return expressions are the actual text of whatever follows the "return" keyword. The script removes all comments in return expressions and converts all strings to "str". This preprocessing greatly simplifies pattern matching.

Note: It was found that the stubs / prototypes of some functions with complex arguments were not handled correctly, resulting in incorrectly formatted stubs (.pyi)
Therefore this functionality has been replaced by MyPy `stubgen`

14.2 Documentation on Type hints

- Type hints cheat sheet
- PEP 3107 – Function Annotations
- PEP 484 – Type Hints
- Optional Static Typing for Python
- TypeShed
- SO question

CHAPTER

FIFTEEN

DOCUMENTATION

This documentation is built using Sphinxs with the bulk of the documents written in markdown and hosted on read the docs.

The markdown files are processed using Myst

Some diagrams have been generated using Mermaid and integrated using the Mermaid plugin

Documentation for the scripts is created using the Spinxs AutoApi plugin

CHAPTER

SIXTEEN

DEVELOPING

16.1 Cloning the repo

```
git clone https://github.com/Josverl/micropython-stubber.git
cd micropython-stubber

poetry install --with dev --with docs
stubber clone
```

16.2 Windows 10

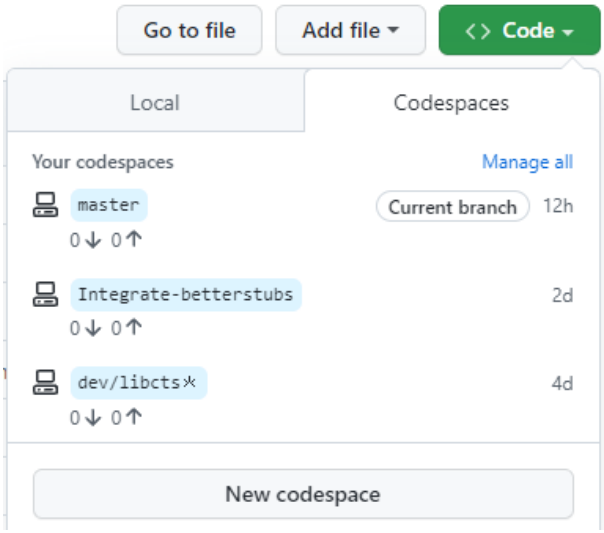
I use Windows 10/11 and use WSL2 to run the linux based parts. if you develop on other platform, it is quite likely that you may need to change some details. if that is needed , please update/add to the documentation and send a documentation PR.

- clone
- create python virtual environment (optional)
- install requirements-dev
- setup sister repos
- run test to verify setup

16.3 Github codespaces

Is is also possible to start a pre-configure development environment in GitHub Codespaces this is probably the fastest and quickest way to start developing.

Note that Codespaces is currently in an extended beta.



16.4 Wrestling with two pythons

This project combines CPython and MicroPython in one project. As a result you may/will need to switch the configuration of pylint and VSCode to match the section of code that you are working on. This is caused by the fact that pylint does not support per-folder configuration

to help switching there are 2 different .pylintrc files stored in the root of the project to simplify switching.

Similar changes will need to be done to the .vscode/settings.json

If / when we can get pylance to work with the micropython stubs , this may become simpler as Pylance natively supports multi-root workspaces, meaning that you can open multiple folders in the same Visual Studio Code session and have Pylance functionality in each folder.

16.5 Minification

If you make changes to the createstubs.py script , you should also update the minified version by running `python process.py minify` at some point.

If you forget to do this there is a github action that should do this for you and create a PR for your branch.

16.6 Testing

MicroPython-Stubber has a number of tests written in Pytest

see below overview

folder	what	how	used where
board	createstubs.pynormal & minified	runs createstubs.py on micropython-linux ports	WSL2 and github actions
check-out_repo	simple_git mod-uleretrieval of frozen modules	does not use mocking but actually retrieves different firmware versions locally using git or dowNloads modules for online	local win-dows
com-mon	all other tests	common	local + github action

Note: Also see *test documentation*

Platform detection to support pytest In order to allow both simple usability om MicroPython and testability on Full Python, createstubs does a runtime test to determine the actual platform it is running on while importing the module This is similar to using the `if __name__ == "__main__":` preamble If running on MicroPython, then it starts stubbing

```
if isMicroPython():
    main()
```

Testing on micropython linux port(s) In order to be able to test `createstubs.py`, it has been updated to run on linux, and accept a –path parameter to indicate the path where the stubs should be stored.

16.7 Debugging Cpython code that run Micropython

Some of the test code run the micropython executable using `subprocess.run()`. When you try to debug these tests the VSCode debugger (debugpy](https://github.com/microsoft/debugpy) then tries to attach to that micropython subprocess in order to facilitate debugging. This will fail as reported in this issue.

The solution to this problem is to disable subprocess debugging using the `"subProcess": false` switch.

```
// launch.json
        {
            // disable pytest coverage report as it conflicts with debugging tests
            "name": "Debug pytest tests",
            "type": "python",
            "purpose": [
                "debug-test"
            ],
            "console": "integratedTerminal",
            "justMyCode": false,
            "stopOnEntry": false,
            "subProcess": false, // Avoid debugpy trying to debug micropython
            "env": {
                "PYTEST_ADDOPTS": "--no-cov"
            }
        },
```

16.8 github actions

16.8.1 pytests.yml

This workflow will :

- test the workstation scripts
- test the createstubs.py script on multiple micropython linux versions
- test the minified createstubs.py script on multiple micropython linux versions

16.8.2 run minify-pr.yml

This workflow will :

- create a minified version of createstubs.py
- run a quick test on that
- and submit a PR to the branch -minify

CHAPTER

SEVENTEEN

TESTING

A significant number of tests have been created in pytest.

- The tests are located in the `tests` folder.
- The `tests/data` folder contains folders with subs that are used to verify the correct working of the minification modules
- debugging the tests only works if –no-cov is specified for pytest

17.1 testing & debugging createstubs.py

- the `tests\mocks` folder contains mock-modules that allow the micropython code to be run in CPython. This is used by the unit tests that verify `createstubs.py` and it minified version.
- in order to load / debug the test the python path needs to include the cpython_core modules (Q&D)
- mocking cpython_core/os is missing the implementation attribute so that has been added (Q&D)

17.2 platform detection

In order to allow both simple usability om MicroPython and testability on *full* Python, createstubs does a runtime test to determine the actual platform it is running on while importing the module

This is similar to using the `if __name__ == "__main__":` preamble

```
if isMicroPython():
    main()
```

This allows pytest test running on full Python to import `createstubs.py` and run tests against individual methods, while allowing the script to run directly on import on a MicroPython board.

Note: Some test are platform dependent and have been marked to only run on linux or windows

17.3 Code Coverage

Code coverage is measured and reported in the `coverage/index.html` report. This report is not checked in to the repo, and therefore is only

CHAPTER

EIGHTEEN

PUBLISHING

Publishing is done using poetry

reqs:

- poetry installed and on path `pip install poetry`
  - poetry config
  - `poetry config repositories.test-pypi https://test.pypi.org/legacy/`
  - poetry config repositories.pypi https://upload.pypi.org/legacy/
- poetry api key stored in systems secure store or in environment variable `PYPI_API_KEY`
  - **PYPI test**
    - get token from https://test.pypi.org/manage/account/token/
    - store token using `poetry config pypi-token.pypi pypi-YYYYYYYY`
  - **PYPI Production**
    - get token from https://pypi.org/manage/account/token/
    - store token using `poetry config pypi-token.pypi pypi-XXXXXXXX`
- bump version

  `poetry version prerelease` `poetry version patch`
- poetry publish `poetry publish --build -r test-pypi` `poetry publish --build`

CHAPTER

NINETEEN

API REFERENCE

This page contains auto-generated API reference documentation[1].

19.1 stubber

get the version

19.1.1 Subpackages

stubber.codemod

Submodules

stubber.codemod.add_comment

Add comment(s) to each file

Module Contents

Classes

AddComment	A command that acts identically to a visitor-based transform, but also has

class stubber.codemod.add_comment.**AddComment**(*context: libcst.codemod.CodemodContext*, *comments: List[str]*)

Bases: libcst.codemod.VisitorBasedCodemodCommand

A command that acts identically to a visitor-based transform, but also has the support of add_args() and running supported helper transforms after execution. See CodemodCommand and ContextAwareTransformer for additional documentation.

Parameters

- **context** (*libcst.codemod.CodemodContext*) –

[1] Created with sphinx-autoapi

- **comments** (*List[str]*) –

DESCRIPTION: str = 'Add comment(s) to each file'

static add_args(*arg_parser: argparse.ArgumentParser*) → None

Add command-line args that a user can specify for running this codemod.

Parameters
arg_parser (*argparse.ArgumentParser*) –

Return type
None

visit_Comment(*node: libcst.Comment*) → None

connect comments from the source

Parameters
node (*libcst.Comment*) –

Return type
None

leave_Module(*original_node: libcst.Module*, *updated_node: libcst.Module*) → libcst.Module

If the tag already exists, don't modify the file.

Parameters

- **original_node** (*libcst.Module*) –
- **updated_node** (*libcst.Module*) –

Return type
libcst.Module

stubber.codemod.add_method

Codemod to add methods to a classes.

Used to add methods that are documented, but are not reported by the firmware, so they are also not present in the board stubs.

Module Contents

Classes

CallFinder	Find the Pin.__call__ method and extract it from a (machine) module.
CallAdder	Add a __call__ method to a class if it is missing.

class stubber.codemod.add_method.**CallFinder**

Bases: libcst.matchers.MatcherDecoratableTransformer

Find the Pin.__call__ method and extract it from a (machine) module.

class_name: str = 'Pin'

method_name: *str* = '__call__'

detect_call(*node: libcst.FunctionDef*) → None

find the __call__ method and store it.

Parameters
node (*libcst.FunctionDef*) –

Return type
None

class stubber.codemod.add_method.**CallAdder**(*call_meth: libcst.FunctionDef*)

Bases: libcst.matchers.MatcherDecoratableTransformer

Add a __call__ method to a class if it is missing.

Parameters
call_meth (*libcst.FunctionDef*) –

class_name = 'Pin'

has_call = 0

detect_call(*node: libcst.FunctionDef*) → None

Detect if the class already has a __call__ method.

Parameters
node (*libcst.FunctionDef*) –

Return type
None

add_call(*original_node: libcst.ClassDef*, *updated_node: libcst.ClassDef*) → libcst.ClassDef

Add the __call__ method to the class if it is not already there.

Parameters

- **original_node** (*libcst.ClassDef*) –
- **updated_node** (*libcst.ClassDef*) –

Return type
libcst.ClassDef

stubber.codemod.board

" Codemods to create the different variants of createstubs.py

Module Contents

Classes

CreateStubsVariant	Dictates create stubs target variant.
ReadModulesCodemod	Replaces static modules list with file-load method.
ModuleDocCodemod	Replaces a module's docstring.
ModulesUpdateCodemod	Update or replace the static module list(s) with the provided changes.
LVGLCodemod	Generates createstubs.py LVGL variant.
LowMemoryCodemod	Generates createstubs.py low-memory variant.
DBCodemod	Generates createstubs.py db variant.
CreateStubsCodemod	Generates createstubs.py variant based on provided variant.

Attributes

_STUBBER_MATCHER
_MODULES_MATCHER
_DEF_MAIN_MATCHER
_PROBLEMATIC_MATCHER
_EXCLUDED_MATCHER
_LOW_MEM_MODULE_DOC
_DB_MODULE_DOC
_LVGL_MODULE_DOC

stubber.codemod.board.**_STUBBER_MATCHER**

stubber.codemod.board.**_MODULES_MATCHER**

stubber.codemod.board.**_DEF_MAIN_MATCHER**

stubber.codemod.board.**_PROBLEMATIC_MATCHER**

stubber.codemod.board.**_EXCLUDED_MATCHER**

stubber.codemod.board.**_LOW_MEM_MODULE_DOC = Multiline-String**

```
"""
"""Create stubs for (all) modules on a MicroPython board.

    This variant of the createstubs.py script is optimised for use on low-
↪memory devices, and reads the list of modules from a text file
    `modulelist.txt` in the root or `libs` folder that should be uploaded␣
```

(continues on next page)

(continued from previous page)

```
↪to the device.
    If that cannot be found then only a single module (micropython) is␣
↪stubbed.
    In order to run this on low-memory devices two additional steps are␣
↪recommended:
    - minification, using python-minifier
      to reduce overall size, and remove logging overhead.
    - cross compilation, using mpy-cross,
      to avoid the compilation step on the micropython device
"""
"""
```

stubber.codemod.board.**_DB_MODULE_DOC = Multiline-String**

```
"""
"""
Create stubs for (all) modules on a MicroPython board.

    This variant of the createstubs.py script is optimized for use on very-
↪low-memory devices.
    Note: this version has undergone limited testing.

    1) reads the list of modules from a text file `modulelist.txt` that␣
↪should be uploaded to the device.
    2) stored the already processed modules in a text file `modulelist.done`
    3) process the modules in the database:
        - stub the module
        - update the modulelist.done file
        - reboots the device if it runs out of memory
    4) creates the modules.json

    If that cannot be found then only a single module (micropython) is␣
↪stubbed.
    In order to run this on low-memory devices two additional steps are␣
↪recommended:
    - minification, using python-minifierto reduce overall size, and remove␣
↪logging overhead.
    - cross compilation, using mpy-cross, to avoid the compilation step on␣
↪the micropython device

"""
"""
```

stubber.codemod.board.**_LVGL_MODULE_DOC = Multiline-String**

```
"""
"""
Create stubs for the lvgl modules on a MicroPython board.

Note that the stubs can be very large, and it may be best to directly store␣
↪them on an SD card if your device supports this.
"""
```

(continues on next page)

(continued from previous page)

```
"""
```

class stubber.codemod.board.**CreateStubsVariant**

Bases: str, enum.Enum

Dictates create stubs target variant.

BASE = 'base'

MEM = 'mem'

DB = 'db'

LVGL = 'lvgl'

class stubber.codemod.board.**ReadModulesCodemod**(*context: libcst.codemod.CodemodContext*, *reader_node: libcst.Module | None = None*)

Bases: libcst.codemod.Codemod

Replaces static modules list with file-load method.

Parameters

- **context** (*libcst.codemod.CodemodContext*) –
- **reader_node** (*Optional[libcst.Module]*) –

modules_reader_node: libcst.Module

transform_module_impl(*tree: libcst.Module*) → libcst.Module

Replaces static modules list with file-load method.

Parameters
tree (*libcst.Module*) –

Return type
libcst.Module

class stubber.codemod.board.**ModuleDocCodemod**(*context: libcst.codemod.CodemodContext*, *module_doc: str*)

Bases: libcst.codemod.Codemod

Replaces a module's docstring.

Parameters

- **context** (*libcst.codemod.CodemodContext*) –
- **module_doc** (*str*) –

module_doc: str

transform_module_impl(*tree: libcst.Module*) → libcst.Module

Replaces a module's docstring.

Parameters
tree (*libcst.Module*) –

Return type
libcst.Module

class stubber.codemod.board.**ModulesUpdateCodemod**(*context: libcst.codemod.CodemodContext*, *, *modules:* stubber.codemod.modify_list.ListChangeSet | *None* = *None*, *problematic:* stubber.codemod.modify_list.ListChangeSet | *None* = *None*, *excluded:* stubber.codemod.modify_list.ListChangeSet | *None* = *None*)

Bases: libcst.codemod.Codemod

Update or replace the static module list(s) with the provided changes.

Parameters

- **context** (*libcst.codemod.CodemodContext*) –
- **modules** (*Optional[stubber.codemod.modify_list.ListChangeSet]*) –
- **problematic** (*Optional[stubber.codemod.modify_list.ListChangeSet]*) –
- **excluded** (*Optional[stubber.codemod.modify_list.ListChangeSet]*) –

modules_changeset: *stubber.codemod.modify_list.ListChangeSet* | None

problematic_changeset: *stubber.codemod.modify_list.ListChangeSet* | None

excluded_changeset: *stubber.codemod.modify_list.ListChangeSet* | None

modules_scope: libcst.matchers.BaseMatcherNode

problematic_scope: libcst.matchers.BaseMatcherNode

excluded_scope: libcst.matchers.BaseMatcherNode

iter_transforms() → Iterator[libcst.matchers.MatcherDecoratableTransformer]

Return type
Iterator[libcst.matchers.MatcherDecoratableTransformer]

transform_module_impl(*tree: libcst.Module*) → libcst.Module

Update or replace the static module list(s) with the provided changes.

Parameters
tree (*libcst.Module*) –

Return type
libcst.Module

class stubber.codemod.board.**LVGLCodemod**(*context: libcst.codemod.CodemodContext*)

Bases: libcst.codemod.Codemod

Generates createstubs.py LVGL variant.

Parameters
context (*libcst.codemod.CodemodContext*) –

modules_transform: *ModulesUpdateCodemod*

init_node: libcst.Module

transform_module_impl(*tree: libcst.Module*) → libcst.Module

Generates createstubs.py LVGL variant.

Parameters
tree (*libcst.Module*) –

Return type
libcst.Module

class stubber.codemod.board.**LowMemoryCodemod**(*context: libcst.codemod.CodemodContext*)

Bases: libcst.codemod.Codemod

Generates createstubs.py low-memory variant.

Parameters
context (*libcst.codemod.CodemodContext*) –

transform_module_impl(*tree: libcst.Module*) → libcst.Module

Generates createstubs.py low-memory variant. - replace the static module list with the low-memory variant (read from file)

Parameters
tree (*libcst.Module*) –

Return type
libcst.Module

class stubber.codemod.board.**DBCodemod**(*context: libcst.codemod.CodemodContext*)

Bases: libcst.codemod.Codemod

Generates createstubs.py db variant.

Parameters
context (*libcst.codemod.CodemodContext*) –

transform_module_impl(*tree: libcst.Module*) → libcst.Module

Generates createstubs.py db variant.

Parameters
tree (*libcst.Module*) –

Return type
libcst.Module

class stubber.codemod.board.**CreateStubsCodemod**(*context: libcst.codemod.CodemodContext*, *variant:* CreateStubsVariant = *CreateStubsVariant.BASE*, *, *modules:* stubber.codemod.modify_list.ListChangeSet | *None* = *None*, *problematic:* stubber.codemod.modify_list.ListChangeSet | *None* = *None*, *excluded:* stubber.codemod.modify_list.ListChangeSet | *None* = *None*)

Bases: libcst.codemod.Codemod

Generates createstubs.py variant based on provided variant.

Parameters

- **context** (*libcst.codemod.CodemodContext*) –
- **variant** (CreateStubsVariant) –

- **modules** (*Optional[stubber.codemod.modify_list.ListChangeSet]*) –
- **problematic** (*Optional[stubber.codemod.modify_list.ListChangeSet]*) –
- **excluded** (*Optional[stubber.codemod.modify_list.ListChangeSet]*) –

variant: *CreateStubsVariant*

modules_transform: *ModulesUpdateCodemod*

transform_module_impl(*tree: libcst.Module*) → libcst.Module

Generates a createstubs.py variant based on provided flavor. Transform it to emit the appropriate variant of createstubs.py, Optionally allows to replace the - list of modules to stub. (if relevant for the flavour) - list of problematic modules. - list of excluded modules.

Parameters
tree (*libcst.Module*) –

Return type
libcst.Module

stubber.codemod.enrich

Enrich firmware stubs by copying docstrings and parameter information from doc-stubs or python source code. Both (.py or .pyi) files are supported.

Module Contents

Functions

enrich_file(→ Optional[str])	Enrich a firmware stubs using the doc-stubs in another folder.
enrich_folder(→ int)	Enrich a folder with containing firmware stubs using the doc-stubs in another folder.

stubber.codemod.enrich.**enrich_file**(*target_path: pathlib.Path*, *docstub_path: pathlib.Path*, *diff: bool = False*, *write_back: bool = False*, *package_name=''*) → str | None

Enrich a firmware stubs using the doc-stubs in another folder. Both (.py or .pyi) files are supported.

Parameters

- **source_path** – the path to the firmware stub to enrich
- **docstub_path** (*pathlib.Path*) – the path to the folder containing the doc-stubs
- **diff** (*bool*) – if True, return the diff between the original and the enriched source file
- **write_back** (*bool*) – if True, write the enriched source file back to the source_path
- **target_path** (*pathlib.Path*) –

Return type
Optional[str]

Returns: - None or a string containing the diff between the original and the enriched source file

stubber.codemod.enrich.**enrich_folder**(*source_path: pathlib.Path*, *docstub_path: pathlib.Path*, *show_diff: bool = False*, *write_back: bool = False*, *require_docstub: bool = False*, *package_name: str = ''*) → int

Enrich a folder with containing firmware stubs using the doc-stubs in another folder.

Returns the number of files enriched.

Parameters

- **source_path** (*pathlib.Path*) –
- **docstub_path** (*pathlib.Path*) –
- **show_diff** (*bool*) –
- **write_back** (*bool*) –
- **require_docstub** (*bool*) –
- **package_name** (*str*) –

Return type
int

stubber.codemod.merge_docstub

Merge documentation and type information from from the docstubs into a board stub

Module Contents

Classes

MergeCommand	A libcst transformer that merges the type-rich information from a doc-stub into

Attributes

empty_module

stubber.codemod.merge_docstub.**empty_module**

class stubber.codemod.merge_docstub.**MergeCommand**(*context: libcst.codemod.CodemodContext*, *docstub_file: pathlib.Path | str*)

Bases: `libcst.codemod.VisitorBasedCodemodCommand`

A libcst transformer that merges the type-rich information from a doc-stub into a firmware stub. The resulting file will contain information from both sources.

- module docstring - from source
- function parameters and types - from docstubs
- function return types - from docstubs

- function docstrings - from source

Parameters

- **context** (*libcst.codemod.CodemodContext*) –
- **docstub_file** (*Union[pathlib.Path, str]*) –

DESCRIPTION: str = 'Merge the type-rich information from a doc-stub into a firmware stub'

static add_args(*arg_parser: argparse.ArgumentParser*) → None

Add command-line args that a user can specify for running this codemod.

Parameters
arg_parser (*argparse.ArgumentParser*) –

Return type
None

leave_Module(*original_node: libcst.Module*, *updated_node: libcst.Module*) → libcst.Module

This method is responsible for updating the module node after processing it in the codemod. It performs the following tasks: 1. Adds any needed imports from the doc-stub. 2. Adds *from module import* * from the doc-stub. 3. Updates the module docstring. 4. Updates the comments in the module.

Parameters

- **original_node** (*libcst.Module*) – The original module node.
- **updated_node** (*libcst.Module*) – The updated module node after processing.

Returns
The updated module node.

Return type
libcst.Module

visit_ClassDef(*node: libcst.ClassDef*) → bool | None

keep track of the the (class, method) names to the stack

Parameters
node (*libcst.ClassDef*) –

Return type
Optional[bool]

leave_ClassDef(*original_node: libcst.ClassDef*, *updated_node: libcst.ClassDef*) → libcst.ClassDef

Parameters

- **original_node** (*libcst.ClassDef*) –
- **updated_node** (*libcst.ClassDef*) –

Return type
libcst.ClassDef

visit_FunctionDef(*node: libcst.FunctionDef*) → bool | None

Parameters
node (*libcst.FunctionDef*) –

Return type
Optional[bool]

leave_FunctionDef(*original_node: libcst.FunctionDef*, *updated_node: libcst.FunctionDef*) → libcst.FunctionDef | libcst.ClassDef

Update the function Parameters and return type, decorators and docstring

Parameters

- **original_node** (*libcst.FunctionDef*) –
- **updated_node** (*libcst.FunctionDef*) –

Return type
Union[libcst.FunctionDef, libcst.ClassDef]

stubber.codemod.modify_list

Module Contents

Classes

ListChangeSet	Describes a set of changes to be made to a list.
ModifyListElements	Modifies the elements of a list (i.e, of modules to stub or exclude),

class stubber.codemod.modify_list.**ListChangeSet**

Describes a set of changes to be made to a list. - add: a list of elements to add to the list - remove: a list of elements to remove from the list - replace: if True, the list will be replaced with the elements in add

add: Sequence[libcst.BaseExpression]

remove: Sequence[libcst.matchers.BaseMatcherNode]

replace: bool = False

classmethod from_strings(*, *add: Sequence[str] | None = None*, *remove: Sequence[str] | None = None*, *replace: bool = False*) → *ListChangeSet*

Parameters

- **add** (*Optional[Sequence[str]]*) –
- **remove** (*Optional[Sequence[str]]*) –
- **replace** (*bool*) –

Return type
ListChangeSet

class stubber.codemod.modify_list.**ModifyListElements**(*, *change_set:* ListChangeSet)

Bases: *stubber.codemod.utils.ScopeableMatcherTransformer*

Modifies the elements of a list (i.e, of modules to stub or exclude), adding and removing elements as specified in the change_set.

Parameters
change_set (ListChangeSet) –

change_set: *ListChangeSet*

modify_list_elements(*original_node: libcst.List*, *updated_node: libcst.List*) → libcst.List

Parameters

- **original_node** (*libcst.List*) –
- **updated_node** (*libcst.List*) –

Return type
libcst.List

stubber.codemod.utils

Module Contents

Classes

ScopeableMatcherTransformer	MatcherDecoratableTransformer that can be reused with different scopes.

Functions

shallow_copy_function(→ types.FunctionType)	Create a shallow copy of the given function.

stubber.codemod.utils.**shallow_copy_function**(*func: Any*) → types.FunctionType

Create a shallow copy of the given function.

The returned function is unbound and does not copy attributes defined on the function.

Parameters
func (*Any*) –

Return type
types.FunctionType

class stubber.codemod.utils.**ScopeableMatcherTransformer**

Bases: libcst.matchers.MatcherDecoratableTransformer

MatcherDecoratableTransformer that can be reused with different scopes.

scope_matcher: libcst.matchers.BaseMatcherNode | None

_build_scoped_meth(*method_name: str*, *scope_matcher: libcst.matchers.BaseMatcherNode*)

Build unbound scoped method from parent class.

Parameters

- **method_name** (*str*) –
- **scope_matcher** (*libcst.matchers.BaseMatcherNode*) –

with_scope(*scope_matcher: libcst.matchers.BaseMatcherNode*) → libcst.matchers.MatcherDecoratableTransformer

Construct a copy of this matcher with visitors scoped to *scope_matcher*.

Parameters
scope_matcher (*libcst.matchers.BaseMatcherNode*) –

Return type
libcst.matchers.MatcherDecoratableTransformer

stubber.commands

Submodules

stubber.commands.build_cmd

Build stub packages - is a Light version of Publish command

Module Contents

Functions

cli_build(family, versions, ports, boards, clean, force)	Commandline interface to publish stubs.

stubber.commands.build_cmd.**cli_build**(*family: str*, *versions: str | List[str]*, *ports: str | List[str]*, *boards: str | List[str]*, *clean: bool*, *force: bool*)

Commandline interface to publish stubs.

Parameters

- **family** (*str*) –
- **versions** (*Union[str, List[str]]*) –
- **ports** (*Union[str, List[str]]*) –
- **boards** (*Union[str, List[str]]*) –
- **clean** (*bool*) –
- **force** (*bool*) –

stubber.commands.cli

command line interface - main group

Module Contents

Functions

stubber_cli(→ None)	
set_loglevel(→ str)	Set log level based on verbose level

stubber.commands.cli.**stubber_cli**(*ctx: click.Context*, *verbose: int = 0*) → None

Parameters

- **ctx** (*click.Context*) –
- **verbose** (*int*) –

Return type
None

stubber.commands.cli.**set_loglevel**(*verbose: int*) → str

Set log level based on verbose level Get the level from the verbose setting (0=INFO, 1=DEBUG, 2=TRACE) Set the format string, based on the level. Add the handler to the logger, with the level and format string. Return the level

Parameters
verbose (*int*) –

Return type
str

stubber.commands.clone_cmd

Clone/fetch the micropython repos locally.

Module Contents

Functions

cli_clone(path[, stubs])	Clone/fetch the micropython repos locally.

stubber.commands.clone_cmd.**cli_clone**(*path: str | pathlib.Path*, *stubs: bool = False*)

Clone/fetch the micropython repos locally.

The local repos are used to generate frozen-stubs and doc-stubs.

Parameters

- **path** (*Union[str, pathlib.Path]*) –
- **stubs** (*bool*) –

stubber.commands.config_cmd

Show the current configuration

Module Contents

Functions

cli_config()	Show the current configuration

stubber.commands.config_cmd.**cli_config**()
Show the current configuration

stubber.commands.enrich_folder_cmd

enrich machinestubs with docstubs

Module Contents

Functions

cli_enrich_folder(stubs_folder, docstubs_folder[, ...])	Enrich the stubs in stub_folder with the docstubs in docstubs_folder.

stubber.commands.enrich_folder_cmd.**cli_enrich_folder**(*stubs_folder: str | pathlib.Path*, *docstubs_folder: str | pathlib.Path*, *diff: bool = False*, *dry_run: bool = False*, *package_name: str = ''*)

Enrich the stubs in stub_folder with the docstubs in docstubs_folder.

Parameters

- **stubs_folder** (*Union[str, pathlib.Path]*) –
- **docstubs_folder** (*Union[str, pathlib.Path]*) –
- **diff** (*bool*) –
- **dry_run** (*bool*) –
- **package_name** (*str*) –

stubber.commands.get_core_cmd

Get core CPython stubs from PyPi.

Module Contents

Functions

cli_get_core([stub_folder, stubgen, black])	Download core CPython stubs from PyPi.

stubber.commands.get_core_cmd.**cli_get_core**(*stub_folder: str = CONFIG.stub_path.as_posix()*, *stubgen: bool = True*, *black: bool = True*)

Download core CPython stubs from PyPi.

Get the core (CPython compat) modules for both MicroPython and Pycopy.

Parameters

- **stub_folder** (*str*) –
- **stubgen** (*bool*) –
- **black** (*bool*) –

stubber.commands.get_docstubs_cmd

get-docstubs

Module Contents

Functions

cli_docstubs(ctx[, path, target, black, basename, version])	Build stubs from documentation.

stubber.commands.get_docstubs_cmd.**cli_docstubs**(*ctx: click.Context*, *path: str = CONFIG.repo_path.as_posix()*, *target: str = CONFIG.stub_path.as_posix()*, *black: bool = True*, *basename: str = 'micropython'*, *version: str = ''*)

Build stubs from documentation.

Read the Micropython library documentation files and use them to build stubs that can be used for static type-checking.

Parameters

- **ctx** (*click.Context*) –
- **path** (*str*) –
- **target** (*str*) –

- **black** (*bool*) –
- **basename** (*str*) –
- **version** (*str*) –

stubber.commands.get_frozen_cmd

Get the frozen stubs for MicroPython.

Module Contents

Functions

cli_get_frozen([stub_folder, version, stubgen, black, ...])	Get the frozen stubs for MicroPython.

stubber.commands.get_frozen_cmd.**cli_get_frozen**(*stub_folder: str = CONFIG.stub_path.as_posix()*, *version: str = ''*, *stubgen: bool = True*, *black: bool = True*, *autoflake: bool = True*)

Get the frozen stubs for MicroPython.

Get the frozen modules for the checked out version of MicroPython

Parameters

- **stub_folder** (*str*) –
- **version** (*str*) –
- **stubgen** (*bool*) –
- **black** (*bool*) –
- **autoflake** (*bool*) –

stubber.commands.get_lobo_cmd

get-lobo (frozen)

Module Contents

Functions

cli_get_lobo([stub_folder, pyi, black])	Get the frozen stubs for Lobo-esp32.

stubber.commands.get_lobo_cmd.**cli_get_lobo**(*stub_folder: str = CONFIG.stub_path.as_posix()*, *pyi: bool = True*, *black: bool = True*)

Get the frozen stubs for Lobo-esp32.

Parameters

- **stub_folder** (*str*) –
- **pyi** (*bool*) –
- **black** (*bool*) –

stubber.commands.merge_cmd

enrich machinestubs with docstubs

Module Contents

Functions

cli_merge_docstubs(versions, boards, ports, family)	Enrich the stubs in stub_folder with the docstubs in docstubs_folder.

stubber.commands.merge_cmd.**cli_merge_docstubs**(*versions: str | List[str]*, *boards: str | List[str]*, *ports: str | List[str]*, *family: str*)

Enrich the stubs in stub_folder with the docstubs in docstubs_folder.

Parameters

- **versions** (*Union[str, List[str]]*) –
- **boards** (*Union[str, List[str]]*) –
- **ports** (*Union[str, List[str]]*) –
- **family** (*str*) –

stubber.commands.minify_cmd

Minify createstubs*.py.

Module Contents

Functions

cli_minify(→ int)	Minify createstubs*.py.

stubber.commands.minify_cmd.**cli_minify**(*ctx: click.Context*, *source: str | pathlib.Path*, *keep_report: bool*, *diff: bool*, *compile: bool*, *all: bool*) → int

Minify createstubs*.py.

Creates a minified version of the SOURCE micropython file in TARGET (file or folder). The goal is to use less memory / not to run out of memory, while generating Firmware stubs.

Parameters

- **ctx** (*click.Context*) –

- **source** (*Union[str, pathlib.Path]*) –
- **keep_report** (*bool*) –
- **diff** (*bool*) –
- **compile** (*bool*) –
- **all** (*bool*) –

Return type
int

stubber.commands.publish_cmd

Commandline interface to publish stubs.

Module Contents

Functions

cli_publish(family, versions, ports, boards[, ...])	Commandline interface to publish stubs.

stubber.commands.publish_cmd.**cli_publish**(*family: str*, *versions: str | List[str]*, *ports: str | List[str]*, *boards: str | List[str]*, *production: bool = True*, *build: bool = False*, *force: bool = False*, *dry_run: bool = False*, *clean: bool = False*)

Commandline interface to publish stubs.

Parameters

- **family** (*str*) –
- **versions** (*Union[str, List[str]]*) –
- **ports** (*Union[str, List[str]]*) –
- **boards** (*Union[str, List[str]]*) –
- **production** (*bool*) –
- **build** (*bool*) –
- **force** (*bool*) –
- **dry_run** (*bool*) –
- **clean** (*bool*) –

stubber.commands.stub_cmd

Create or update .pyi type hint files.

Module Contents

Functions

cli_stub(source)	Create or update .pyi type hint files.

stubber.commands.stub_cmd.**cli_stub**(*source: str | pathlib.Path*)

Create or update .pyi type hint files.

Parameters

source (*Union[str, pathlib.Path]*) –

stubber.commands.switch_cmd

switch to a specific version of the micropython repos

Module Contents

Functions

cli_switch(path[, tag])	Switch to a specific version of the micropython repos.

Attributes

VERSION_LIST

stubber.commands.switch_cmd.**VERSION_LIST**

stubber.commands.switch_cmd.**cli_switch**(*path: str | pathlib.Path*, *tag: str | None = None*)

Switch to a specific version of the micropython repos.

The Micropython-lib repo will be checked out to a commit that corresponds in time to that version tag, in order to allow non-current versions to be stubbed correctly.

The repros must be cloned already

Parameters

- **path** (*Union[str, pathlib.Path]*) –
- **tag** (*Optional[str]*) –

stubber.commands.upd_fallback_cmd

update-fallback folder with common set of stubs that cater for most of the devices

Module Contents

Functions

cli_update_fallback(version[, stub_folder])	Update the fallback stubs.

stubber.commands.upd_fallback_cmd.**cli_update_fallback**(*version: str*, *stub_folder: str = CONFIG.stub_path.as_posix()*)

Update the fallback stubs.

The fallback stubs are updated/collated from files of the firmware-stubs, doc-stubs and core-stubs.

Parameters

- **version** (*str*) –
- **stub_folder** (*str*) –

stubber.commands.upd_module_list_cmd

update-fallback folder with common set of stubs that cater for most of the devices

Module Contents

Functions

cli_update_module_list()	Update the module list based on the information in the data folder.

stubber.commands.upd_module_list_cmd.**cli_update_module_list**()

Update the module list based on the information in the data folder.

stubber.commands.variants_cmd

Create all variant of createstubs*.py.

Module Contents

Functions

cli_variants(→ int)	Update all variants of createstubs*.py.

stubber.commands.variants_cmd.**cli_variants**(*ctx: click.Context*, *target_folder: str = ''*, *version: str = CONFIG.stable_version*) → int

Update all variants of createstubs*.py.

Parameters

- **ctx** (*click.Context*) –
- **target_folder** (*str*) –
- **version** (*str*) –

Return type
int

stubber.freeze

Submodules

stubber.freeze.common

common functions for frozen stub generation

Module Contents

Functions

get_portboard(manifest_path)	returns a 2-tuple of the port and board in the provided manifest path
get_freeze_path(→ Tuple[pathlib.Path, str])	get path to a folder to store the frozen stubs for the given port/board
apply_frozen_module_fixes(freeze_path, mpy_path)	apply common fixes to the fozen modules to improve stub generation

stubber.freeze.common.**get_portboard**(*manifest_path: pathlib.Path*)

returns a 2-tuple of the port and board in the provided manifest path

raises an ValueError if neither a port or board can be found

Parameters
manifest_path (*pathlib.Path*) –

stubber.freeze.common.**get_freeze_path**(*stub_path: pathlib.Path*, *port: str*, *board: str*) → Tuple[pathlib.Path, str]

get path to a folder to store the frozen stubs for the given port/board

Parameters

- **stub_path** (*pathlib.Path*) –
- **port** (*str*) –
- **board** (*str*) –

Return type
Tuple[pathlib.Path, str]

stubber.freeze.common.**apply_frozen_module_fixes**(*freeze_path: pathlib.Path*, *mpy_path: pathlib.Path*)

apply common fixes to the fozen modules to improve stub generation

Parameters

- **freeze_path** (*pathlib.Path*) –
- **mpy_path** (*pathlib.Path*) –

stubber.freeze.freeze_folder

get and parse the to-be-frozen .py modules for micropython to extract the static type information.

Module Contents

Functions

freeze_folders(stub_folder, mpy_folder, lib_folder, ...)	get and parse the to-be-frozen .py modules for micropython to extract the static type information

Attributes

FAMILY

stubber.freeze.freeze_folder.**FAMILY = 'micropython'**

stubber.freeze.freeze_folder.**freeze_folders**(*stub_folder: str*, *mpy_folder: str*, *lib_folder: str*, *version: str*)

get and parse the to-be-frozen .py modules for micropython to extract the static type information locates the to-be-frozen files in modules folders - 'ports/<port>/modules/*.py' - 'ports/<port>/boards/<board>/modules/*.py'

Parameters

- **stub_folder** (*str*) –
- **mpy_folder** (*str*) –
- **lib_folder** (*str*) –

- **version** (*str*) –

stubber.freeze.freeze_manifest_2

Freeze manifest files for micropython 1.16 and later uses the manifest file to generate frozen stubs

Module Contents

Functions

make_path_vars(*[, mpy_path, mpy_lib_path, port, board])	
freeze_one_manifest_2(manifest, frozen_stub_path, ...)	
copy_frozen_to_stubs(stub_path, port, board, files, ...)	copy the frozen files from the manifest to the stubs folder

stubber.freeze.freeze_manifest_2.**make_path_vars**(*, *mpy_path: pathlib.Path = CONFIG.mpy_path*, *mpy_lib_path: pathlib.Path = CONFIG.mpy_lib_path*, *port: str | None = None*, *board: str | None = None*)

Parameters

- **mpy_path** (*pathlib.Path*) –
- **mpy_lib_path** (*pathlib.Path*) –
- **port** (*Optional[str]*) –
- **board** (*Optional[str]*) –

stubber.freeze.freeze_manifest_2.**freeze_one_manifest_2**(*manifest: pathlib.Path*, *frozen_stub_path: pathlib.Path*, *mpy_path: pathlib.Path*, *mpy_lib_path: pathlib.Path*, *version: str*)

Parameters

- **manifest** (*pathlib.Path*) –
- **frozen_stub_path** (*pathlib.Path*) –
- **mpy_path** (*pathlib.Path*) –
- **mpy_lib_path** (*pathlib.Path*) –
- **version** (*str*) –

stubber.freeze.freeze_manifest_2.**copy_frozen_to_stubs**(*stub_path: pathlib.Path*, *port: str*, *board: str*, *files: List[stubber.tools.manifestfile.ManifestOutput]*, *version: str*, *mpy_path: pathlib.Path*)

copy the frozen files from the manifest to the stubs folder

stubpath = the destination : # stubs/{family}-{version}-frozen

Parameters

- **stub_path** (*pathlib.Path*) –
- **port** (*str*) –
- **board** (*str*) –
- **files** (*List[stubber.tools.manifestfile.ManifestOutput]*) –
- **version** (*str*) –
- **mpy_path** (*pathlib.Path*) –

stubber.freeze.get_frozen

Collect modules and python stubs from MicroPython source projects (v1.12 +) and stores them in the all_stubs folder
The all_stubs folder should be mapped/symlinked to the micropython_stubs/stubs repo/folder

Module Contents

Functions

get_manifests(→ List[pathlib.Path])	Returns a list of all manifests.py files found in the ports folder of the MicroPython repo
add_comment_to_path(→ None)	Add a comment to the top of each file in the path
freeze_any(→ pathlib.Path)	Get and parse the to-be-frozen .py modules for micropython to extract the static type information
get_fsp(→ pathlib.Path)	

Attributes

FAMILY

stubber.freeze.get_frozen.**FAMILY = 'micropython'**

stubber.freeze.get_frozen.**get_manifests**(*mpy_path: pathlib.Path*) → List[pathlib.Path]

Returns a list of all manifests.py files found in the ports folder of the MicroPython repo

Parameters

mpy_path (*pathlib.Path*) –

Return type

List[pathlib.Path]

stubber.freeze.get_frozen.**add_comment_to_path**(*path: pathlib.Path*, *comment: str*) → None

Add a comment to the top of each file in the path using a codemod

Parameters

- **path** (*pathlib.Path*) –
- **comment** (*str*) –

Return type
None

stubber.freeze.get_frozen.**freeze_any**(*stub_folder: pathlib.Path | None = None*, *version: str = V_PREVIEW*, *mpy_path: pathlib.Path | None = None*, *mpy_lib_path: pathlib.Path | None = None*) → pathlib.Path

Get and parse the to-be-frozen .py modules for micropython to extract the static type information

- requires that the MicroPython and Micropython-lib repos are checked out and available on a local path
- repos should be cloned side-by-side as some of the manifests refer to micropython-lib scripts using a relative path

The micropython-* repos must be checked out to the required version/tag.

Parameters

- **stub_folder** (*Optional[pathlib.Path]*) –
- **version** (*str*) –
- **mpy_path** (*Optional[pathlib.Path]*) –
- **mpy_lib_path** (*Optional[pathlib.Path]*) –

Return type
pathlib.Path

stubber.freeze.get_frozen.**get_fsp**(*version: str*, *stub_folder: pathlib.Path | None = None*) → pathlib.Path

Parameters

- **version** (*str*) –
- **stub_folder** (*Optional[pathlib.Path]*) –

Return type
pathlib.Path

stubber.publish

Submodules

stubber.publish.bump

Bump a version number to a next version

Module Contents

Functions

bump_version(→ packaging.version.Version)	Increases a version number

stubber.publish.bump.**bump_version**(*current: packaging.version.Version*, *, *major_bump: bool = False*, *minor_bump: bool = False*, *micro_bump: bool = False*, *version_bump: bool = False*, *post_bump: bool = False*, *rc: int = 0*) → packaging.version.Version

Increases a version number

This allows for a new stub-release to be published while still using the Major.Minor.Patch version numbers of Micropython if rc = 0(default) : bump post release

format: x.y.z.post1, x.y.z.post2 ...

if rec specified:
drop the post release and set the release candidate number

ref: https://peps.python.org/pep-0440/

Parameters

- **current** (*packaging.version.Version*) –
- **major_bump** (*bool*) –
- **minor_bump** (*bool*) –
- **micro_bump** (*bool*) –
- **version_bump** (*bool*) –
- **post_bump** (*bool*) –
- **rc** (*int*) –

Return type
packaging.version.Version

stubber.publish.candidates

In order to generate stups for all ports and boars and versions of micropython we need to know what versions are available. This module provides functions to :

- get a list of all ports for a given version of micropython (list micropython ports)
- get a list of all ports and board for a given version of micropython (list micropython ports boards)
  - get a list of versions for micropython (version candidates)
- get the frozen stubs for a given version of micropython (frozen candidates)
- get a list of all the docstubs (docstub candidates)
- get a list of the firmware/board stubs (firmware candidates)

Module Contents

Functions

subfolder_names(path)	returns a list of names of the subfolders of the given path
version_candidates(→ Generator[str, None, None])	get a list of versions for the given family and suffix
list_frozen_ports([family, version, path])	get list of ports with frozen stubs for a given family and version
list_micropython_ports([family, mpy_path])	get list of micropython ports for a given family and version
list_micropython_port_boards(port[, family, mpy_path])	get list of micropython boards for a given family version and board
frozen_candidates(→ Generator[Dict[str, Any], None, None])	generate a list of possible firmware stubs for the given family (, version port and board) ?
is_auto(thing)	Is this version/port/board specified as 'auto' ?
docstub_candidates([family, versions, path])	Generate a list of possible documentation stub candidates for the given family and version.
board_candidates([family, versions, mpy_path, pt])	generate a list of possible board stub candidates for the given family and version.
filter_list(worklist[, ports, boards])	filter a list of candidates down to the ones we want, based on the ports and boards specified (case insensitive)

stubber.publish.candidates.**subfolder_names**(*path: pathlib.Path*)

returns a list of names of the subfolders of the given path

Parameters
path (*pathlib.Path*) –

stubber.publish.candidates.**version_candidates**(*suffix: str*, *prefix: str = '.*'*, ***, *path: pathlib.Path = CONFIG.stub_path*, *oldest: str = OLDEST_VERSION*) → Generator[str, None, None]

get a list of versions for the given family and suffix

Parameters

- **suffix** (*str*) –
- **prefix** (*str*) –
- **path** (*pathlib.Path*) –
- **oldest** (*str*) –

Return type
Generator[str, None, None]

stubber.publish.candidates.**list_frozen_ports**(*family: str = 'micropython'*, *version: str = V_PREVIEW*, *path: pathlib.Path = CONFIG.stub_path*)

get list of ports with frozen stubs for a given family and version

Parameters

- **family** (*str*) –
- **version** (*str*) –
- **path** (*pathlib.Path*) –

stubber.publish.candidates.**list_micropython_ports**(*family: str = 'micropython'*, *mpy_path: pathlib.Path = CONFIG.mpy_path*)

get list of micropython ports for a given family and version

Parameters

- **family** (*str*) –
- **mpy_path** (*pathlib.Path*) –

stubber.publish.candidates.**list_micropython_port_boards**(*port: str*, *family: str = 'micropython'*, *mpy_path: pathlib.Path = CONFIG.mpy_path*)

get list of micropython boards for a given family version and board

Parameters

- **port** (*str*) –
- **family** (*str*) –
- **mpy_path** (*pathlib.Path*) –

stubber.publish.candidates.**frozen_candidates**(*family: str = 'micropython'*, *versions: str | List[str] = V_PREVIEW*, *ports: str | List[str] = 'all'*, *boards: str | List[str] = 'all'*, ***, *path: pathlib.Path = CONFIG.stub_path*) → Generator[Dict[str, Any], None, None]

generate a list of possible firmware stubs for the given family (, version port and board) ? - family = micropython

board and port are ignored, they are looked up from the available frozen stubs

- versions = 'latest' , 'auto' or a list of versions
- port = 'auto' or a specific port
- board = 'auto' or a specific board, 'generic' must be specified in lowercare

Parameters

- **family** (*str*) –
- **versions** (*Union[str, List[str]]*) –
- **ports** (*Union[str, List[str]]*) –
- **boards** (*Union[str, List[str]]*) –
- **path** (*pathlib.Path*) –

Return type
Generator[Dict[str, Any], None, None]

stubber.publish.candidates.**is_auto**(*thing: None | str | List[str]*)

Is this version/port/board specified as 'auto' ?

Parameters
thing (*Union[None, str, List[str]]*) –

stubber.publish.candidates.**docstub_candidates**(*family: str = 'micropython'*, *versions: str | List[str] = V_PREVIEW*, *path: pathlib.Path = CONFIG.stub_path*)

Generate a list of possible documentation stub candidates for the given family and version.

Note that the folders do not need to exist, with the exception of auto which will scan the stubs folder for versions of docstubs

Parameters

- **family** (*str*) –
- **versions** (*Union[str, List[str]]*) –
- **path** (*pathlib.Path*) –

stubber.publish.candidates.**board_candidates**(*family: str = 'micropython'*, *versions: str | List[str] = V_PREVIEW*, ***, *mpy_path: pathlib.Path = CONFIG.mpy_path*, *pt: str = FIRMWARE_STUBS*)

generate a list of possible board stub candidates for the given family and version. list is based on the micropython repo: /ports/<list of ports>/boards/<list of boards>

Parameters

- **family** (*str*) –
- **versions** (*Union[str, List[str]]*) –
- **mpy_path** (*pathlib.Path*) –
- **pt** (*str*) –

stubber.publish.candidates.**filter_list**(*worklist: List[Dict[str, str]]*, *ports: List[str] | str | None = None*, *boards: List[str] | str | None = None*)

filter a list of candidates down to the ones we want, based on the ports and boards specified (case insensitive) for board also match using a '**GENERIC_**' prefix, so board 's3' will match candidate 'GENERIC_S3'

Parameters

- **worklist** (*List[Dict[str, str]]*) –
- **ports** (*Optional[Union[List[str], str]]*) –
- **boards** (*Optional[Union[List[str], str]]*) –

stubber.publish.database

basic interface to the json database

Module Contents

Functions

get_database(→ pysondb.PysonDB)	Open the json database at the given path.

stubber.publish.database.**get_database**(*publish_path: pathlib.Path | str*, *production: bool = False*) → pysondb.PysonDB

Open the json database at the given path.

The database should be located in a subfolder */publish* of the root path. The database name is determined by the production flag as *package_data[_test].jsondb*

Parameters

- **publish_path** (*Union[pathlib.Path, str]*) –
- **production** (*bool*) –

Return type
pysondb.PysonDB

stubber.publish.defaults

Build and packaging defaults for stubber

Module Contents

Functions

default_board(→ str)	Return the default board for the given version and port

Attributes

DEFAULT_BOARDS	
GENERIC_L	generic lowercase
GENERIC_U	GENERIC uppercase
GENERIC	GENERIC eithercase

stubber.publish.defaults.**DEFAULT_BOARDS: Dict[str, List[str]]**

stubber.publish.defaults.**GENERIC_L = 'generic'**

generic lowercase

stubber.publish.defaults.**GENERIC_U = 'GENERIC'**

GENERIC uppercase

stubber.publish.defaults.**GENERIC**

GENERIC eithercase

stubber.publish.defaults.**default_board**(*port: str*, *version=V_PREVIEW*) → str

Return the default board for the given version and port

Parameters
port (*str*) –

Return type
str

stubber.publish.enums

Enumerations for the stubber package.

Module Contents

Classes

StubSource	str(object='') -> str

Attributes

ALL_TYPES
COMBO_STUBS
DOC_STUBS
CORE_STUBS
FIRMWARE_STUBS

class stubber.publish.enums.**StubSource**

Bases: str, enum.Enum

str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

FIRMWARE = 'Firmware stubs'

stubs built by combining the firmware, frozen and core stubs

FROZEN = 'Frozen stubs'

stubs of python modules that are frozen as part of the firmware image

CORE = 'Core stubs'

stubs that allow (some) MicroPython code to be run by CPython

DOC = 'Doc stubs'

stubs built by parsing the micropython RST documentation files

MERGED = 'Merged stubs'

stubs built by merging the information from doc-stubs and firmware-stubs

__str__()

Return str(self).

__repr__()

Return repr(self).

stubber.publish.enums.**ALL_TYPES = ['combo', 'doc', 'core', 'firmware']**

stubber.publish.enums.**COMBO_STUBS**

stubber.publish.enums.**DOC_STUBS**

stubber.publish.enums.**CORE_STUBS**

stubber.publish.enums.**FIRMWARE_STUBS**

stubber.publish.helpers

This module provides a function to retrieve the docstring of a Python module.

Module Contents

Functions

get_module_docstring(→ Union[str, Any])	Retrieve the docstring of a Python module.

stubber.publish.helpers.**get_module_docstring**(*fname: pathlib.Path*) → str | Any

Retrieve the docstring of a Python module.

Parameters

fname (*Path*) – The path to the Python module file.

Returns

The docstring of the Python module, or None if no docstring is found.

Return type

Union[str, Any]

stubber.publish.merge_docstubs

Merge firmware stubs and docstubs into a single folder

Module Contents

Functions

merge_all_docstubs([versions, family, ports, boards, ...])	merge docstubs and board stubs to merged stubs
copy_and_merge_docstubs(fw_path, dest_path, docstub_path)	**param fw_path** Path to firmware stubs (absolute path)

stubber.publish.merge_docstubs.**merge_all_docstubs**(*versions: List[str] | str | None = None*, *family: str = 'micropython'*, *ports: List[str] | str | None = None*, *boards: List[str] | str | None = None*, ***, *mpy_path: pathlib.Path = CONFIG.mpy_path*)

merge docstubs and board stubs to merged stubs

Parameters

- **versions** (*Optional[Union[List[str], str]]*) –
- **family** (*str*) –
- **ports** (*Optional[Union[List[str], str]]*) –
- **boards** (*Optional[Union[List[str], str]]*) –
- **mpy_path** (*pathlib.Path*) –

stubber.publish.merge_docstubs.**copy_and_merge_docstubs**(*fw_path: pathlib.Path*, *dest_path: pathlib.Path*, *docstub_path: pathlib.Path*)

Parameters

- **fw_path** (*pathlib.Path*) – Path to firmware stubs (absolute path)
- **dest_path** (*pathlib.Path*) – Path to destination (absolute path)
- **mpy_version** – micropython version ('1.18')
- **docstub_path** (*pathlib.Path*) –

Copy files from the firmware stub folders to the merged - 1 - Copy all firmware stubs to the package folder - 1.B - clean up a little bit - 2 - Enrich the firmware stubs with the document stubs

stubber.publish.missing_class_methods

Add missing methods to classes in the stubs that are documented in the docstubs

Module Contents

Functions

add_machine_pin_call(merged_path, version)	Add the __call__ method to the machine.Pin and pyb.Pin class

stubber.publish.missing_class_methods.**add_machine_pin_call**(*merged_path: pathlib.Path*, *version: str*)

Add the __call__ method to the machine.Pin and pyb.Pin class in all pyb and machine/umachine stubs

Parameters

- **merged_path** (*pathlib.Path*) –
- **version** (*str*) –

stubber.publish.package

prepare a set of stub files for publishing to PyPi

Module Contents

Functions

package_name(→ str)	generate a package name for the given package type
get_package(→ stubber.publish.stubpackage.StubPackage)	Get the package from the database or create a new one if it does not exist.
get_package_info(→ Union[Dict, None])	get a package's record from the json db if it can be found
create_package(→ stubber.publish.stubpackage.StubPackage)	create and initialize a package with the correct sources
combo_sources(→ stubber.publish.stubpackage.StubSources)	Build a source set for combo stubs

stubber.publish.package.**package_name**(*pkg_type: str*, ***, *port: str = ''*, *board: str = ''*, *family: str = 'micropython'*, ***kwargs*) → str

generate a package name for the given package type

Parameters

- **pkg_type** (*str*) –
- **port** (*str*) –
- **board** (*str*) –
- **family** (*str*) –

Return type

str

stubber.publish.package.**get_package**(*db: pysondb.PysonDB*, *, *pkg_type: str*, *version: str*, *port: str*, *board: str = GENERIC_L*, *family: str = 'micropython'*) → *stubber.publish.stubpackage.StubPackage*

Get the package from the database or create a new one if it does not exist.

Parameters

- **db** (*pysondb.PysonDB*) –
- **pkg_type** (*str*) –
- **version** (*str*) –
- **port** (*str*) –
- **board** (*str*) –
- **family** (*str*) –

Return type
stubber.publish.stubpackage.StubPackage

stubber.publish.package.**get_package_info**(*db: pysondb.PysonDB*, *pub_path: pathlib.Path*, *, *pkg_name: str*, *mpy_version: str*) → Dict | None

get a package's record from the json db if it can be found matches om the package name and version

pkg_name: package name (micropython-esp32-stubs) mpy_version: micropython/firmware version (1.18)

Parameters

- **db** (*pysondb.PysonDB*) –
- **pub_path** (*pathlib.Path*) –
- **pkg_name** (*str*) –
- **mpy_version** (*str*) –

Return type
Union[Dict, None]

stubber.publish.package.**create_package**(*pkg_name: str*, *mpy_version: str*, *, *port: str = ''*, *board: str = ''*, *family: str = 'micropython'*, *pkg_type: str = COMBO_STUBS*) → *stubber.publish.stubpackage.StubPackage*

create and initialize a package with the correct sources

Parameters

- **pkg_name** (*str*) –
- **mpy_version** (*str*) –
- **port** (*str*) –
- **board** (*str*) –
- **family** (*str*) –
- **pkg_type** (*str*) –

Return type
stubber.publish.stubpackage.StubPackage

stubber.publish.package.**combo_sources**(*family: str*, *port: str*, *board: str*, *ver_flat: str*) → stubber.publish.stubpackage.StubSources

Build a source set for combo stubs

Parameters

- **family** (*str*) –
- **port** (*str*) –
- **board** (*str*) –
- **ver_flat** (*str*) –

Return type
stubber.publish.stubpackage.StubSources

stubber.publish.pathnames

Helper functions to deal with path names and filenames for the folders in the stubs repo

Module Contents

Functions

get_base(candidate[, version])	
board_folder_name(→ str)	Return the name of the firmware folder. Can be in AnyCase.
get_board_path(→ pathlib.Path)	
get_merged_path(→ pathlib.Path)	

stubber.publish.pathnames.**get_base**(*candidate: Dict[str, str]*, *version: str | None = None*)

Parameters

- **candidate** (*Dict[str, str]*) –
- **version** (*Optional[str]*) –

stubber.publish.pathnames.**board_folder_name**(*fw: Dict*, ***, *version: str | None = None*) → str

Return the name of the firmware folder. Can be in AnyCase.

Parameters

- **fw** (*Dict*) –
- **version** (*Optional[str]*) –

Return type
str

stubber.publish.pathnames.**get_board_path**(*candidate: Dict*) → pathlib.Path

Parameters
candidate (*Dict*) –

Return type
pathlib.Path

stubber.publish.pathnames.**get_merged_path**(*fw: Dict*) → pathlib.Path

Parameters
fw (*Dict*) –

Return type
pathlib.Path

stubber.publish.publish

prepare a set of stub files for publishing to PyPi

!!Note: anything excluded in .gitignore is not packaged by poetry

Module Contents

Functions

build_multiple(→ List[Dict[str, Any]])	Build a bunch of stub packages
publish_multiple(→ List[Dict[str, Any]])	Publish a bunch of stub packages
build_worklist(family, versions, ports, boards)	Build a worklist of packages to build or publish, and filter to only the requested ports and boards

stubber.publish.publish.**build_multiple**(*family: str = 'micropython'*, *versions: List[str] = [V_PREVIEW]*, *ports: List[str] = ['all']*, *boards: List[str] = [GENERIC_U]*, *production: bool = False*, *clean: bool = False*, *force: bool = False*) → List[Dict[str, Any]]

Build a bunch of stub packages

Parameters

- **family** (*str*) –
- **versions** (*List[str]*) –
- **ports** (*List[str]*) –
- **boards** (*List[str]*) –
- **production** (*bool*) –
- **clean** (*bool*) –
- **force** (*bool*) –

Return type
List[Dict[str, Any]]

stubber.publish.publish.**publish_multiple**(*family: str = 'micropython'*, *versions: List[str] = ['v1.19.1']*, *ports: List[str] = ['all']*, *boards: List[str] = [GENERIC_U]*, *production: bool = False*, *clean: bool = False*, *build: bool = False*, *force: bool = False*, *dry_run: bool = False*) → List[Dict[str, Any]]

Publish a bunch of stub packages

Parameters

- **family** (*str*) –
- **versions** (*List[str]*) –
- **ports** (*List[str]*) –
- **boards** (*List[str]*) –
- **production** (*bool*) –
- **clean** (*bool*) –
- **build** (*bool*) –
- **force** (*bool*) –
- **dry_run** (*bool*) –

Return type
List[Dict[str, Any]]

stubber.publish.publish.**build_worklist**(*family: str*, *versions: List[str] | str*, *ports: List[str] | str*, *boards: List[str] | str*)

Build a worklist of packages to build or publish, and filter to only the requested ports and boards

Parameters

- **family** (*str*) –
- **versions** (*Union[List[str], str]*) –
- **ports** (*Union[List[str], str]*) –
- **boards** (*Union[List[str], str]*) –

stubber.publish.pypi

Read versions published to PyPi or test.PyPi. uses the pypi-simple package to get the versions from the simple index.

Module Contents

Functions

get_pypi_versions(package_name[, base, production])	Get all versions of a package from a PyPI endpoint.

stubber.publish.pypi.**get_pypi_versions**(*package_name: str*, *base: packaging.version.Version | None = None*, *production: bool = True*)

Get all versions of a package from a PyPI endpoint.

Parameters

- **package_name** (*str*) –
- **base** (*Optional[packaging.version.Version]*) –
- **production** (*bool*) –

stubber.publish.stubpackage

Create a stub-only package for a specific version of micropython

Module Contents

Classes

VersionedPackage	Represents a versioned package.
Builder	Builder class for creating and updating MicroPython stub packages.
PoetryBuilder	Build a package using Poetry
StubPackage	Create a stub-only package for a specific version , port and board of micropython

Attributes

Status
StubSources
STUBS_COPY_FILTER
STDLIB_UMODULES

stubber.publish.stubpackage.**Status**

stubber.publish.stubpackage.**StubSources**

stubber.publish.stubpackage.**STUBS_COPY_FILTER**

stubber.publish.stubpackage.**STDLIB_UMODULES = ['ucollections']**

class stubber.publish.stubpackage.**VersionedPackage**(*package_name: str*, *, *mpy_version: str*)

Bases: `object`

Represents a versioned package.

Parameters

- **package_name** (*str*) –
- **mpy_version** (*str*) –

package_name

The name of the package.

Type
str

mpy_version

The MicroPython version.

Type
str

__init__(self, package_name

str, mpy_version: str): Initializes a new instance of the VersionedPackage class.

is_preview(*self*)

Checks if the package is a preview version.

pkg_version(*self*) → str

Returns the version of the package.

Return type
str

pkg_version(self, version

str) -> None: Sets the version of the package.

get_prerelease_package_version(self, production

bool = False) -> str: Gets the next prerelease version for the package.

get_next_package_version(self, prod

bool = False, rc=False) -> str: Gets the next version for the package.

next_pkg_version(self, production

bool) -> str: Gets the next version for the package.

bump(self, *, rc

int = 0) -> str: Bumps the postrelease version of the package.

property pkg_version: str

return the version of the package

Return type
str

__str__() → str

Return str(self).

Return type
str

__repr__() → str

Return repr(self).

Return type
str

__eq__(*o: object*) → bool

Return self==value.

Parameters
o (*object*) –

Return type
bool

__hash__() → int

Return hash(self).

Return type
int

next_package_version(*production: bool*) → str

Get the next version for the package

Parameters
production (*bool*) –

Return type
str

is_preview()

_get_next_preview_package_version(*production: bool = False*) → str

Get the next prerelease version for the package. this is used for preview versions of micropython (-preview, formerly known as 'latest')

Parameters
production (*bool*) –

Return type
str

_get_next_package_version(*prod: bool = False*, *rc=False*) → str

Get the next version for the package.

Parameters
prod (*bool*) –

Return type
str

bump(*, *rc: int = 0*) → str

bump the postrelease version of the package, and write the change to disk if rc >= 1, the version is bumped to the specified release candidate

Parameters
rc (*int*) –

Return type
str

class stubber.publish.stubpackage.**Builder**(*package_name: str*, *, *mpy_version: str = '0.0.1'*, *port: str*, *board: str = GENERIC_U*, *description: str = 'MicroPython stubs'*, *stubs: StubSources | None = None*)

Bases: *VersionedPackage*

Builder class for creating and updating MicroPython stub packages.

Parameters

- **package_name** (*str*) – The name of the package.
- **mpy_version** (*str, optional*) – The version of MicroPython. Defaults to "0.0.1".
- **port** (*str*) – The port for the package.
- **board** (*str, optional*) – The board for the package. Defaults to GENERIC_U.
- **description** (*str, optional*) – The description of the package. Defaults to "MicroPython stubs".
- **stubs** (*Optional[StubSources], optional*) – The stub sources for the package. Defaults to None.

package_name

The name of the package.

Type
str

mpy_version

The version of MicroPython.

Type
str

port

The port for the package.

Type
str

board

The board for the package.

Type
str

description

The description of the package.

Type
str

stub_sources

The stub sources for the package.

Type
Optional[StubSources]

hash

The hash of all the files in the package.

Type
None

stub_hash

The hash of the stub files.

Type
None

Properties:

package_path (Path): The package path based on the package name and version, relative to the publish folder. toml_path (Path): The path to the *pyproject.toml* file. pyproject (Union[Dict[str, Any], None]): The parsed pyproject.toml or None.

`create_update_pyproject_toml()`

Create or update/overwrite a *pyproject.toml* file.

Return type
None

`check()`

Check if the package is valid.

Return type
bool

`clean()`

Remove the stub files from the package folder.

Return type
None

`copy_stubs()`

Copy files from all listed stub folders to the package folder.

Return type
None

`update_package_files()`

Update the stub-only package for a specific version of MicroPython.

Return type
None

`write_package_json()`

Write the package.json file to disk.

Return type
None

`to_dict()`

Return the package as a dict to store in the jsondb.

Return type
dict

`from_dict(json_data`

Dict): Load the package from a dict (from the jsondb).

`calculate_hash(include_md`

bool = True): Create a SHA1 hash of all files in the package.

`update_hashes()`

Update the package hashes.

Return type
None

is_changed(include_md

bool = True): Check if the package has changed.

property package_path: pathlib.Path

package path based on the package name and version and relative to the publish folder

Return type
pathlib.Path

property toml_path: pathlib.Path

the path to the *pyproject.toml* file

Return type
pathlib.Path

property pyproject: Dict[str, Any] | None

parsed pyproject.toml or None

Return type
Union[Dict[str, Any], None]

BUF_SIZE

hash

Hash of all the files in the package

stub_hash

Hash of all .pyi files

abstract create_update_pyproject_toml() → None

create or update/overwrite a *pyproject.toml* file by combining a template file with the given parameters.

Return type
None

check() → bool

Check if the package is valid, to be implemented by the subclass

Return type
bool

clean() → None

Remove the stub files from the package folder

This is used before update the stub package, to avoid lingering stub files, and after the package has been built, to avoid needing to store files multiple times.

.gitignore cannot be used as this will prevent poetry from processing the files.

Return type
None

copy_stubs() → None

Copy files from all listed stub folders to the package folder the order of the stub folders is relevant as "last copy wins"

- 1 - Copy all firmware stubs/merged to the package folder
- 2 - copy the remaining stubs to the package folder
- 3 - remove *.py files from the package folder

Return type
None

`update_umodules`()

Replace the STDLIB umodules with a simple import statement in order to allow the typecheckers to resove the stdlib modules in the usual stdlib location.

`copy_folder`(*stub_type:* stubber.publish.enums.StubSource, *src_path:* *pathlib.Path*)

Parameters

- **stub_type** (*stubber.publish.enums.StubSource*) –
- **src_path** (*pathlib.Path*) –

`update_package_files`() → None

Update the stub-only package for a specific version of micropython

- cleans the package folder
- copies the stubs from the list of stubs.
- creates/updates the readme and the license file

Return type
None

`write_package_json`() → None

write the package.json file to disk

Return type
None

`to_dict`() → dict

return the package as a dict to store in the jsondb

need to simplify some of the Objects to allow serialization to json - the paths to posix paths - the version (semver) to a string - toml file to list of lines

Return type
dict

`from_dict`(*json_data: Dict*) → None

load the package from a dict (from the jsondb)

Parameters
json_data (*Dict*) –

Return type
None

`calculate_hash`(*include_md: bool = True*) → str

Create a SHA1 hash of all files in the package, excluding the pyproject.toml file itself. the hash is based on the content of the .py/.pyi and .md files in the package. if include_md is False , the .md files are not hased, allowing the files in the packeges to be compared simply As a single hash is created across all files, the files are sorted prior to hashing to ensure that the hash is stable.

Note: A changed hash will not indicate which of the files in the package have been changed.

Parameters
include_md (*bool*) –

Return type
str

`add_file_hash`(*file*, *file_hash*)

Adds the hash of a file to the given file hash object. If an error occurs, the file is retried up to 3 times with a 0.2 second delay

Parameters

- **file** (*str*) – The path to the file.
- **file_hash** (*hashlib._Hash*) – The file hash object to update.

Returns
None

`update_hashes`(*ret=False*) → None

Update the package hashes. Resets is_changed() to False

Return type
None

`is_changed`(*include_md: bool = True*) → bool

Check if the package has changed, based on the current and the stored hash. The default checks the hash of all files, including the .md files.

Parameters
include_md (*bool*) –

Return type
bool

`create_license`() → None

Create a license file for the package

- copied from the template license file

Return type
None

`create_readme`() → None

Create a readme file for the package

- based on the template readme file
- with a list of all included stub folders added to it (not the individual stub-files)

Return type
None

`class stubber.publish.stubpackage.PoetryBuilder`(*package_name: str*, ***, *port: str*, *mpy_version: str = '0.0.1'*, *board: str = GENERIC_U*, *description: str = 'MicroPython stubs'*, *stubs: StubSources | None = None*, *json_data: Dict[str, Any] | None = None*)

Bases: *Builder*

Build a package using Poetry

Parameters

- **package_name** (*str*) –
- **port** (*str*) –
- **mpy_version** (*str*) –
- **board** (*str*) –
- **description** (*str*) –
- **stubs** (*Optional[StubSources]*) –
- **json_data** (*Optional[Dict[str, Any]]*) –

property pkg_version: str

return the version of the package

Return type
str

poetry_build() → bool

build the package by running *poetry build*

Return type
bool

poetry_publish(*production: bool = False*) → bool

Parameters
production (*bool*) –

Return type
bool

run_poetry(*parameters: List[str]*) → bool

Run a poetry commandline in the package folder. Note: this may write some output to the console ('All set!')

Parameters
parameters (*List[str]*) –

Return type
bool

check() → bool

check if the package is valid by running *poetry check* Note: this will write some output to the console ('All set!')

Return type
bool

create_update_pyproject_toml() → None

create or update/overwrite a *pyproject.toml* file by combining a template file with the given parameters. and updating it with the pyi files included

Return type
None

update_pyproject_stubs() → int

Add the stub files to the pyproject.toml file

Return type
int

class stubber.publish.stubpackage.**StubPackage**(*package_name: str*, *port: str*, *, *board: str = GENERIC_U*, *version: str = '0.0.1'*, *description: str = 'MicroPython stubs'*, *stubs: StubSources | None = None*, *json_data: Dict[str, Any] | None = None*)

Bases: *PoetryBuilder*

Create a stub-only package for a specific version , port and board of micropython

properties:

- toml_path - the path to the *pyproject.toml* file
- package_path - the path to the folder where the package info will be stored ('./publish').
- pkg_version - the version of the package as used on PyPi (semver). Is stored directly in the *pyproject.toml* file
- pyproject - the contents of the *pyproject.toml* file

Parameters

- **package_name** (*str*) –
- **port** (*str*) –
- **board** (*str*) –
- **version** (*str*) –
- **description** (*str*) –
- **stubs** (*Optional[StubSources]*) –
- **json_data** (*Optional[Dict[str, Any]]*) –

- from_json - load the package from json

- to_json - return the package as json

- create_update_pyproject_toml - create or update the `pyproject.toml` file

- create_readme - create the readme file

- create_license - create the license file

- copy_stubs - copy the stubs to the package folder

- update_included_stubs - update the included stubs in the `pyproject.toml` file

- create_hash - create a hash of the package files

- update_package_files - combines clean, copy, and create reeadme & updates

update_sources() → StubSources

Update the stub sources to: - FIRMWARE: prefer -merged stubs over bare firmware stubs - FROZEN: fallback to use the GENERIC folder for the frozen sources if no board specific folder exists

Return type
StubSources

update_distribution(*production: bool*) → bool

Update the package .pyi files, if all the sources are available

Parameters
production (*bool*) –

Return type
bool

build_distribution(*production: bool*, *force=False*) → bool

Build a package look up the previous package version in the dabase

- update package files
- build the wheels and sdist

Parameters

- **production** (*bool*) – PyPI or Test-PyPi -
- **force** – BUILD even if no changes

Returns
True if the package was built

Return type
bool

publish_distribution_ifchanged(*db: pysondb.PysonDB*, ***, *production: bool*, *build=False*, *force=False*, *dry_run=False*, *clean: bool = False*) → bool

Publish a package to PyPi look up the previous package version in the dabase, and only publish if there are changes to the package - change determied by hash across all files

Build

- update package files
- build the wheels and sdist

Publish

- publish to PyPi
- update database with new hash

Parameters

- **db** (*pysondb.PysonDB*) –
- **production** (*bool*) –
- **clean** (*bool*) –

Return type
bool

publish_distribution(*dry_run*, *production*, *db*)

Publishes the package to PyPi or Test-PyPi.

Parameters

- **dry_run** (*bool*) – If True, performs a dry run without actually publishing.
- **production** (*bool*) – If True, publishes to PyPi. If False, publishes to Test-PyPi.

- **db** – The database object to save the package state.

Returns
True if the publish was successful, False otherwise.

Return type
bool

are_package_sources_available() → bool

Check if (all) the packages sources exist.

Return type
bool

stubber.rst

.rst processing

Submodules

stubber.rst.classsort

Sort list of classess in parent-child order note that this does not take multiple inheritance into account ref : https://stackoverflow.com/questions/34964878/python-generate-a-dictionarytree-from-a-list-of-tuples/35049729#35049729 with modification

Module Contents

Functions

sort_classes(classes)	sort a list of classes to respect the parent-child order

stubber.rst.classsort.**sort_classes**(*classes: List[str]*)

sort a list of classes to respect the parent-child order

Parameters
classes (*List[str]*) –

stubber.rst.lookup

Lookup tables for the rst documentation stubber

Module Contents

stubber.rst.lookup.**TYPING_IMPORT: List[str] = ['from __future__ import annotations', 'from typing import IO, Any, Callable, Coroutine, Dict,...**

stubber.rst.lookup.**U_MODULES = ['array', 'binascii', 'io', 'json', 'os', 'select', 'ssl', 'struct', 'socket', 'time', 'zlib']**

List of modules that are documented with the base name only, but can also be imported with a *u* prefix

stubber.rst.lookup.**RST_DOC_FIXES: List[Tuple[str, str]] = [(':class: attention\n', ''), ('.. method:: match.', '.. method:: Match.'), (' ...**

stubber.rst.lookup.**DOCSTUB_SKIP = ['uasyncio.rst', 'builtins.rst', 're.rst']**

this is an list with manual overrides for function returns that could not efficiently be determined from their docstring description Format: a dictionary with : - key = module.[class.]function name - value : two-tuple with (return type , priority)

stubber.rst.lookup.**LOOKUP_LIST**

stubber.rst.lookup.**NONE_VERBS = ['Activate ', 'Build a ', 'Cancel ', 'Clear ', 'Close ', 'cancel ', 'Configure ', 'Connect ',...**

stubber.rst.lookup.**MODULE_GLUE**

stubber.rst.lookup.**PARAM_FIXES**

stubber.rst.lookup.**CHILD_PARENT_CLASS**

stubber.rst.output_dict

ModuleSourceDict represents a source file with the following components

- docstr
- version
- comment
- typing
- Optional: list of constants
- optional: ClassSourcedicts
- optional: FunctionSourcedicts
- optional: individual lines of code

ClassSourceDict represents a source file with the following components

- comment
- class
- docstr
- Optional: list of constants
- __init__ : class signature
- optional: FunctionSourcedicts

- optional: individual lines of code

FunctionSourceDict represents a source file with the following components

- # comments - todo
- optional: decorator
- def - function definition
- docstr
- constants
- body - ...
- optional: individual lines of code

SourceDict is the 'base class'

Module Contents

Classes

SourceDict	(abstract) dict to store source components respecting parent child dependencies and proper definition order
ModuleSourceDict	(abstract) dict to store source components respecting parent child dependencies and proper definition order
ClassSourceDict	(abstract) dict to store source components respecting parent child dependencies and proper definition order
FunctionSourceDict	(abstract) dict to store source components respecting parent child dependencies and proper definition order

class stubber.rst.output_dict.**SourceDict**(*base: List*, *indent: int = 0*, *body: int = 0*, *lf: str = '\n'*, *name=''*)

Bases: OrderedDict

(abstract) dict to store source components respecting parent child dependencies and proper definition order

Parameters

- **base** (*List*) –
- **indent** (*int*) –
- **body** (*int*) –
- **lf** (*str*) –

__str__() → str

convert the OD into a string (the to be generated source code)

Return type
str

__add__(*other:* SourceDict)

Aallows instances of the SourceDict class to be added together using the + operator or the += operator.

Parameters
other (SourceDict) –

add_docstr(*docstr: str | List[str]*, *extra: int = 0*)

Parameters

- **docstr** (*Union[str, List[str]]*) –
- **extra** (*int*) –

add_comment(*line: str | List[str]*)

Add a comment, or list of comments, to this block.

Parameters
line (*Union[str, List[str]]*) –

add_constant(*line: str*, *autoindent: bool = True*)

add constant to the constant scope of this block

Parameters

- **line** (*str*) –
- **autoindent** (*bool*) –

add_constant_smart(*name: str*, *type: str = ''*, *docstr: List[str] | None = None*, *autoindent: bool = True*)

add literal / constant to the constant scope of this block, or a class in this block

Parameters

- **name** (*str*) –
- **type** (*str*) –
- **docstr** (*Optional[List[str]]*) –
- **autoindent** (*bool*) –

abstract find(*name: str*) → str | None

Parameters
name (*str*) –

Return type
Union[str, None]

add_line(*line: str*, *autoindent: bool = True*)

Parameters

- **line** (*str*) –
- **autoindent** (*bool*) –

index(*key: str*)

Parameters
key (*str*) –

class stubber.rst.output_dict.**ModuleSourceDict**(*name: str*, *indent=0*, *lf: str = '\n'*)

Bases: *SourceDict*

(abstract) dict to store source components respecting parent child dependencies and proper definition order

Parameters

- **name** (*str*) –
- **lf** (*str*) –

sort()

make sure all classdefs are in order

__str__()

sort in the correct parent-child order, then convert to string (the code)

find(*name: str*) → str | None

find a classnode based on the name with or without the superclass

Parameters
name (*str*) –

Return type
Union[str, None]

classes()

get a list of the class names in parent-child order

add_import(*imports: str | List[str]*)

add a [list of] imports this module

Parameters
imports (*Union[str, List[str]]*) –

class stubber.rst.output_dict.**ClassSourceDict**(*name: str*, ***, *docstr: List[str] | None = None*, *init: str = ''*, *indent: int = 0*, *lf='\n'*)

Bases: *SourceDict*

(abstract) dict to store source components respecting parent child dependencies and proper definition order

Parameters

- **name** (*str*) –
- **docstr** (*Optional[List[str]]*) –
- **init** (*str*) –
- **indent** (*int*) –

class stubber.rst.output_dict.**FunctionSourceDict**(*name: str*, ***, *definition: List[str] | None = None*, *docstr: List[str] | None = None*, *indent: int = 0*, *decorators: List[str] | None = None*, *lf='\n'*, *is_async: bool = False*)

Bases: *SourceDict*

(abstract) dict to store source components respecting parent child dependencies and proper definition order

Parameters

- **name** (*str*) –
- **definition** (*Optional[List[str]]*) –
- **docstr** (*Optional[List[str]]*) –
- **indent** (*int*) –
- **decorators** (*Optional[List[str]]*) –
- **is_async** (*bool*) –

stubber.rst.reader

Read the Micropython library documentation files and use them to build stubs that can be used for static typechecking using a custom-built parser to read and process the micropython RST files - generates:

- **modules**
  - docstrings
- **function definitions**
  - function parameters based on documentation
  - docstrings
- **classes**
  - docstrings
  - __init__ method
  - parameters based on documentation for class
  - **methods**
    - parameters based on documentation for the method
    - docstrings
- exceptions
- **Tries to determine the return type by parsing the docstring.**
  - Imperative verbs used in docstrings have a strong correlation to return -> None
  - recognizes documented Generators, Iterators, Callable
  - Coroutines are identified based tag "This is a Coroutine". Then if the return type was Foo, it will be transformed to : Coroutine[Foo]
  - a static Lookup list is used for a few methods/functions for which the return type cannot be determined from the docstring.
  - add NoReturn to a few functions that never return (stop / deepsleep / reset)
  - if no type can be detected the type *Any* or *Incomplete* is used

The generated stub files are formatted using *black* and checked for validity using *pyright* Note: black on python 3.7 does not like some function defs *def sizeof(struct, layout_type=NATIVE, /) -> int:*

- ordering of inter-dependent classes in the same module
- **Literals / constants**
  - documentation contains repeated vars with the same indentation
  - Module level:

```
.. data:: IPPROTO_UDP
          IPPROTO_TCP
```

  - class level:

```
.. data:: Pin.IRQ_FALLING
          Pin.IRQ_RISING
          Pin.IRQ_LOW_LEVEL
          Pin.IRQ_HIGH_LEVEL

          Selects the IRQ trigger type.
```

  - literals documented using a wildcard are added as comments only

- Add GLUE imports to allow specific modules to import specific others.
- **Repeats of definitions in the rst file for similar functions or literals**
  - CONSTANTS (module and Class level)
  - functions
  - methods
- **Child/ Parent classes**
  are added based on a (manual) lookup table CHILD_PARENT_CLASS

Module Contents

Classes

FileReadWriter	base class for reading rst files
RSTReader	base class for reading rst files
RSTParser	Parse the RST file and create a ModuleSourceDict
RSTWriter	Reads, parses and writes

Attributes

SEPERATOR

stubber.rst.reader.**SEPERATOR** = '::'

class stubber.rst.reader.**FileReadWriter**

base class for reading rst files

property line: str

get the current line from input, also stores this as last_line to allow for inspection and dumping the json file

Return type
str

read_file(*filename: pathlib.Path*)

Parameters
filename (*pathlib.Path*) –

write_file(*filename: pathlib.Path*) → bool

Parameters
filename (*pathlib.Path*) –

Return type
bool

static is_balanced(*s: str*) → bool

Check if a string has balanced parentheses

Parameters
s (*str*) –

Return type
bool

extend_and_balance_line() → str

Append the current line + next line in order to try to balance the parentheses in order to do this the rst_test array is changed by the function and max_line is adjusted

Return type
str

class stubber.rst.reader.**RSTReader**

Bases: *FileReadWriter*

base class for reading rst files

property module_names: List[str]

list of possible module names [uname , name] (longest first)

Return type
List[str]

property at_anchor: bool

and ..data:: should be added)

Type
Stop at anchor '..something' (however .. note

Return type
bool

docstring_anchors = ['.. note::', '.. data:: Arguments:', '.. data:: Options:', '.. data:: Returns:', '.. data::...

read_file(*filename: pathlib.Path*)

Parameters
filename (*pathlib.Path*) –

at_heading(*large=False*) → bool

stop at heading

Return type
bool

read_docstring(*large: bool = False*) → List[str]

Read a textblock that will be used as a docstring, or used to process a toc tree The textblock is terminated at the following RST line structures/tags

– Heading == Heading ~~ Heading

The blank lines at the start and end are removed to limit the space the docstring takes up.

Parameters
large (*bool*) –

Return type
List[str]

static clean_docstr(*block: List[str]*)

Clean up a docstring

Parameters
block (*List[str]*) –

static add_link_to_docsstr(*block: List[str]*)

Add clickable hyperlinks to CPython docpages

Parameters
block (*List[str]*) –

get_rst_hint()

parse the '.. <rst hint>:: ' from the current line

strip_prefixes(*name: str*, *strip_mod: bool = True*, *strip_class: bool = False*)

Remove the modulename. and or the classname. from the begining of a name

Parameters

- **name** (*str*) –
- **strip_mod** (*bool*) –
- **strip_class** (*bool*) –

class stubber.rst.reader.**RSTParser**(*v_tag: str*)

Bases: *RSTReader*

Parse the RST file and create a ModuleSourceDict most methods have side effects

Parameters
v_tag (*str*) –

target = '.py'

PARAM_RE_FIXES

leave_class()

fix_parameters(*params: str*, *name: str = ''*) → str

Patch / correct the documentation parameter notation to a supported format that works for linting. - params is the string containing the parameters as documented in the rst file - name is the name of the function or method or Class

Parameters

- **params** (*str*) –
- **name** (*str*) –

Return type
str

static apply_fix(*fix: stubber.rst.lookup.Fix*, *params: str*, *name: str = ''*)

Parameters

- **fix** (*stubber.rst.lookup.Fix*) –
- **params** (*str*) –
- **name** (*str*) –

create_update_class(*name: str*, *params: str*, *docstr: List[str]*)

Parameters

- **name** (*str*) –
- **params** (*str*) –
- **docstr** (*List[str]*) –

parse_toc()

process table of content with additional rst files, and add / include them in the current module

parse_module()

parse a module tag and set the module's docstring

parse_current_module()

parse_function()

parse_class()

parse_method()

parse_exception()

parse_name(*line: str | None = None*)

get the constant/function/class name from a line with an identifier

Parameters
line (*Optional[str]*) –

parse_names(*oneliner: bool = True*)

get a list of constant/function/class names from and following a line with an identifier advances the linecounter

oneliner : treat a line with commas as multiple names (used for constants)

Parameters
oneliner (*bool*) –

parse_data()

process ..data:: lines (one or more) Note: some data islands are included in the docstring of the module/class/function as the ESPNow documentation started to use this pattern.

parse()

class stubber.rst.reader.**RSTWriter**(*v_tag='v1.xx'*)

Bases: *RSTParser*

Reads, parses and writes

write_file(*filename: pathlib.Path*) → bool

Parameters
filename (*pathlib.Path*) –

Return type
bool

prepare_output()

Remove trailing spaces and commas from the output.

stubber.rst.report_return

Work in Progress

build test and % report Will need to be updated after new_output has been implemented.

Module Contents

Functions

process(folder, pattern)

stubber.rst.report_return.**process**(*folder: pathlib.Path*, *pattern: str*)

Parameters

- **folder** (*pathlib.Path*) –
- **pattern** (*str*) –

stubber.rst.rst_utils

Tries to determine the return type by parsing the docstring and the function signature

- if the signature contains a return type –> <something> then that is returned
- **check a lookup dictionary of type overrides,**
  if the functionnae is listed, then use the override
- **use re to find phrases such as:**
  - 'Returns ….. '
  - 'Gets ….. '
- docstring is joined without newlines to simplify parsing
- then parses the docstring to find references to known types and give then a rating though a hand coded model ()
- builds a list return type candidates
- selects the highest ranking candidate
- the default Type is 'Any'

to do:

- **regex :**
  - 'With no arguments the frequency in Hz is returned.'
  - 'Get or set' –> indicates overloaded/optional return Union[None|...]
  - add regex for 'Query' ` Otherwise, query current state if no argument is provided. `
- **regex :**
  - 'With no arguments the frequency in Hz is returned.'
  - 'Get or set' –> indicates overloaded/optional return Union[None|...]
  - add regex for 'Query' ` Otherwise, query current state if no argument is provided. `
- **try if an Azure Machine Learning works as well**
  https://docs.microsoft.com/en-us/azure/machine-learning/quickstart-create-resources
-

Module Contents

Functions

simple_candidates(type, match_string, keywords[, ...])	find and rate possible types and confidence weighting for simple types.
compound_candidates(type, match_string, keywords[, ...])	find and rate possible types and confidence weighting for compound types that can have a subscription.
object_candidates(match_string[, rate, exclude])	find and rate possible types and confidence weighting for Object types.
distill_return(→ List[Dict])	Find return type and confidence.
return_type_from_context(*, docstring, signature, module)	
_type_from_context(*, docstring, signature, module[, ...])	Determine the return type of a function or method based on:

Attributes

TYPING_IMPORT

stubber.rst.rst_utils.**TYPING_IMPORT: List[str] = ['from __future__ import annotations', 'from typing import IO, Any, Callable, Coroutine, Dict,...**

stubber.rst.rst_utils.**simple_candidates**(*type: str*, *match_string: str*, *keywords: List[str]*, *rate: float = 0.5*, *exclude: List[str] | None = None*)

find and rate possible types and confidence weighting for simple types. Case sensitive

Parameters

- **type** (*str*) –

- **match_string** (*str*) –
- **keywords** (*List[str]*) –
- **rate** (*float*) –
- **exclude** (*Optional[List[str]]*) –

stubber.rst.rst_utils.**compound_candidates**(*type: str*, *match_string: str*, *keywords: List[str]*, *rate: float = 0.85*, *exclude: List[str] | None = None*)

find and rate possible types and confidence weighting for compound types that can have a subscription. Case sensitive

Parameters

- **type** (*str*) –
- **match_string** (*str*) –
- **keywords** (*List[str]*) –
- **rate** (*float*) –
- **exclude** (*Optional[List[str]]*) –

stubber.rst.rst_utils.**object_candidates**(*match_string: str*, *rate: float = 0.81*, *exclude: List[str] | None = None*)

find and rate possible types and confidence weighting for Object types. Case sensitive Exclude defaults to ["IRQ"]

Parameters

- **match_string** (*str*) –
- **rate** (*float*) –
- **exclude** (*Optional[List[str]]*) –

stubber.rst.rst_utils.**distill_return**(*return_text: str*) → List[Dict]

Find return type and confidence. Returns a list of possible types and confidence weighting. {

type :str # the return type confidence: float # the confidence between 0.0 and 1 match: Optional[str] # for debugging : the reason the match was made

}

Parameters
return_text (*str*) –

Return type
List[Dict]

stubber.rst.rst_utils.**return_type_from_context**(***, *docstring: str | List[str]*, *signature: str*, *module: str*, *literal: bool = False*)

Parameters

- **docstring** (*Union[str, List[str]]*) –
- **signature** (*str*) –
- **module** (*str*) –
- **literal** (*bool*) –

stubber.rst.rst_utils.**_type_from_context**(*, *docstring: str | List[str]*, *signature: str*, *module: str*, *literal: bool = False*)

Determine the return type of a function or method based on:

- the function signature
- the terminology used in the docstring

Logic: - if the signature contains a return type –> <something> then that is returned - use re to find phrases such as:

- 'Returns ….. '
- 'Gets ….. '
- docstring is joined without newlines to simplify parsing
- then parses the docstring to find references to known types and give then a rating though a hand coded model ()
- builds a list return type candidates
- selects the highest ranking candidate
- the default Type is 'Any'

Parameters

- **docstring** (*Union[str, List[str]]*) –
- **signature** (*str*) –
- **module** (*str*) –
- **literal** (*bool*) –

Package Contents

Classes

SourceDict	(abstract) dict to store source components respecting parent child dependencies and proper definition order
ModuleSourceDict	(abstract) dict to store source components respecting parent child dependencies and proper definition order
ClassSourceDict	(abstract) dict to store source components respecting parent child dependencies and proper definition order
FunctionSourceDict	(abstract) dict to store source components respecting parent child dependencies and proper definition order

Functions

sort_classes(classes)	sort a list of classes to respect the parent-child order
simple_candidates(type, match_string, keywords[, ...])	find and rate possible types and confidence weighting for simple types.
compound_candidates(type, match_string, keywords[, ...])	find and rate possible types and confidence weighting for compound types that can have a subscription.
object_candidates(match_string[, rate, exclude])	find and rate possible types and confidence weighting for Object types.
distill_return(→ List[Dict])	Find return type and confidence.
return_type_from_context(*, docstring, signature, module)	
_type_from_context(*, docstring, signature, module[, ...])	Determine the return type of a function or method based on:

Attributes

TYPING_IMPORT	
LOOKUP_LIST	
NONE_VERBS	
CHILD_PARENT_CLASS	
PARAM_FIXES	
MODULE_GLUE	
RST_DOC_FIXES	
DOCSTUB_SKIP	this is an list with manual overrides for function returns that could not efficiently be determined
U_MODULES	List of modules that are documented with the base name only,
TYPING_IMPORT	
__all__	

stubber.rst.**sort_classes**(*classes: List[str]*)

sort a list of classes to respect the parent-child order

Parameters
classes (*List[str]*) –

stubber.rst.**TYPING_IMPORT: List[str] = ['from __future__ import annotations', 'from typing import IO, Any, Callable, Coroutine, Dict,...**

stubber.rst.**LOOKUP_LIST**

stubber.rst.**NONE_VERBS = ['Activate ', 'Build a ', 'Cancel ', 'Clear ', 'Close ', 'cancel ', 'Configure ', 'Connect ',...**

stubber.rst.**CHILD_PARENT_CLASS**

stubber.rst.**PARAM_FIXES**

stubber.rst.**MODULE_GLUE**

stubber.rst.**RST_DOC_FIXES: List[Tuple[str, str]] = [(':class: attention\n', ''), ('.. method:: match.', '.. method:: Match.'), (' ...**

stubber.rst.**DOCSTUB_SKIP = ['uasyncio.rst', 'builtins.rst', 're.rst']**

this is an list with manual overrides for function returns that could not efficiently be determined from their docstring description Format: a dictionary with : - key = module.[class.]function name - value : two-tuple with (return type , priority)

stubber.rst.**U_MODULES = ['array', 'binascii', 'io', 'json', 'os', 'select', 'ssl', 'struct', 'socket', 'time', 'zlib']**

List of modules that are documented with the base name only, but can also be imported with a *u* prefix

class stubber.rst.**SourceDict**(*base: List*, *indent: int = 0*, *body: int = 0*, *lf: str = '\n'*, *name=''*)

Bases: `OrderedDict`

(abstract) dict to store source components respecting parent child dependencies and proper definition order

Parameters

- **base** (*List*) –
- **indent** (*int*) –
- **body** (*int*) –
- **lf** (*str*) –

__str__() → str

convert the OD into a string (the to be generated source code)

Return type
str

__add__(*other:* SourceDict)

Aallows instances of the SourceDict class to be added together using the + operator or the += operator.

Parameters
other (SourceDict) –

add_docstr(*docstr: str | List[str]*, *extra: int = 0*)

Parameters

- **docstr** (*Union[str, List[str]]*) –
- **extra** (*int*) –

add_comment(*line: str | List[str]*)

Add a comment, or list of comments, to this block.

Parameters
line (*Union[str, List[str]]*) –

add_constant(*line: str*, *autoindent: bool = True*)

add constant to the constant scope of this block

Parameters

- **line** (*str*) –
- **autoindent** (*bool*) –

add_constant_smart(*name: str*, *type: str = ''*, *docstr: List[str] | None = None*, *autoindent: bool = True*)

add literal / constant to the constant scope of this block, or a class in this block

Parameters

- **name** (*str*) –
- **type** (*str*) –
- **docstr** (*Optional[List[str]]*) –
- **autoindent** (*bool*) –

abstract find(*name: str*) → str | None

Parameters
name (*str*) –

Return type
Union[str, None]

add_line(*line: str*, *autoindent: bool = True*)

Parameters

- **line** (*str*) –
- **autoindent** (*bool*) –

index(*key: str*)

Parameters
key (*str*) –

class stubber.rst.**ModuleSourceDict**(*name: str*, *indent=0*, *lf: str = '\n'*)

Bases: *SourceDict*

(abstract) dict to store source components respecting parent child dependencies and proper definition order

Parameters

- **name** (*str*) –
- **lf** (*str*) –

sort()

make sure all classdefs are in order

__str__()

sort in the correct parent-child order, then convert to string (the code)

find(*name: str*) → str | None

find a classnode based on the name with or without the superclass

Parameters
name (*str*) –

Return type
Union[str, None]

`classes`()

get a list of the class names in parent-child order

`add_import`(*imports: str | List[str]*)

add a [list of] imports this module

Parameters
imports (*Union[str, List[str]]*) –

class stubber.rst.**ClassSourceDict**(*name: str*, *, *docstr: List[str] | None = None*, *init: str = ''*, *indent: int = 0*, *lf='\n'*)

Bases: *SourceDict*

(abstract) dict to store source components respecting parent child dependencies and proper definition order

Parameters

- **name** (*str*) –
- **docstr** (*Optional[List[str]]*) –
- **init** (*str*) –
- **indent** (*int*) –

class stubber.rst.**FunctionSourceDict**(*name: str*, *, *definition: List[str] | None = None*, *docstr: List[str] | None = None*, *indent: int = 0*, *decorators: List[str] | None = None*, *lf='\n'*, *is_async: bool = False*)

Bases: *SourceDict*

(abstract) dict to store source components respecting parent child dependencies and proper definition order

Parameters

- **name** (*str*) –
- **definition** (*Optional[List[str]]*) –
- **docstr** (*Optional[List[str]]*) –
- **indent** (*int*) –
- **decorators** (*Optional[List[str]]*) –
- **is_async** (*bool*) –

stubber.rst.**simple_candidates**(*type: str*, *match_string: str*, *keywords: List[str]*, *rate: float = 0.5*, *exclude: List[str] | None = None*)

find and rate possible types and confidence weighting for simple types. Case sensitive

Parameters

- **type** (*str*) –
- **match_string** (*str*) –
- **keywords** (*List[str]*) –
- **rate** (*float*) –
- **exclude** (*Optional[List[str]]*) –

stubber.rst.**compound_candidates**(*type: str*, *match_string: str*, *keywords: List[str]*, *rate: float = 0.85*, *exclude: List[str] | None = None*)

find and rate possible types and confidence weighting for compound types that can have a subscription. Case sensitive

Parameters

- **type** (*str*) –
- **match_string** (*str*) –
- **keywords** (*List[str]*) –
- **rate** (*float*) –
- **exclude** (*Optional[List[str]]*) –

stubber.rst.**object_candidates**(*match_string: str*, *rate: float = 0.81*, *exclude: List[str] | None = None*)

find and rate possible types and confidence weighting for Object types. Case sensitive Exclude defaults to ["IRQ"]

Parameters

- **match_string** (*str*) –
- **rate** (*float*) –
- **exclude** (*Optional[List[str]]*) –

stubber.rst.**distill_return**(*return_text: str*) → List[Dict]

Find return type and confidence. Returns a list of possible types and confidence weighting. {

type :str # the return type confidence: float # the confidence between 0.0 and 1 match: Optional[str] # for debugging : the reason the match was made

}

Parameters
return_text (*str*) –

Return type
List[Dict]

stubber.rst.**return_type_from_context**(*, *docstring: str | List[str]*, *signature: str*, *module: str*, *literal: bool = False*)

Parameters

- **docstring** (*Union[str, List[str]]*) –
- **signature** (*str*) –
- **module** (*str*) –
- **literal** (*bool*) –

stubber.rst.**_type_from_context**(*, *docstring: str | List[str]*, *signature: str*, *module: str*, *literal: bool = False*)

Determine the return type of a function or method based on:

- the function signature
- the terminology used in the docstring

Logic: - if the signature contains a return type –> <something> then that is returned - use re to find phrases such as:

- 'Returns ….. '

- 'Gets ..... '
- docstring is joined without newlines to simplify parsing
- then parses the docstring to find references to known types and give then a rating though a hand coded model ()
- builds a list return type candidates
- selects the highest ranking candidate
- the default Type is 'Any'

Parameters

- **docstring** (*Union[str, List[str]]*) –
- **signature** (*str*) –
- **module** (*str*) –
- **literal** (*bool*) –

stubber.rst.**TYPING_IMPORT: List[str] = ['from __future__ import annotations', 'from typing import IO, Any, Callable, Coroutine, Dict,...**

stubber.rst.**__all__**

stubber.tools

Submodules

stubber.tools.manifestfile

Module Contents

Classes

ManifestFile

exception stubber.tools.manifestfile.**ManifestFileError**

Bases: Exception

Common base class for all non-exit exceptions.

class stubber.tools.manifestfile.**ManifestFile**(*mode*, *path_vars=None*)

_resolve_path(*path*)

_manifest_globals(*kwargs*)

files()

pypi_dependencies()

execute(*manifest_file*)

_add_file(*full_path*, *target_path*, *kind=KIND_AUTO*, *opt=None*)

_search(*base_path*, *package_path*, *files*, *exts*, *kind*, *opt=None*, *strict=False*)

metadata(***kwargs*)

From within a manifest file, use this to set the metadata for the package described by current manifest.

After executing a manifest file (via execute()), call this to obtain the metadata for the top-level manifest file.

See ManifestPackageMetadata.update() for valid kwargs.

include(*manifest_path*, *is_require=False*, ***kwargs*)

Include another manifest.

The manifest argument can be a string (filename) or an iterable of strings.

Relative paths are resolved with respect to the current manifest file.

If the path is to a directory, then it implicitly includes the manifest.py file inside that directory.

Optional kwargs can be provided which will be available to the included script via the *options* variable.

e.g. include("path.py", extra_features=True)

in path.py:
options.defaults(standard_features=True)

freeze minimal modules. if options.standard_features:

freeze standard modules.

if options.extra_features:
freeze extra modules.

require(*name*, *version=None*, *unix_ffi=False*, *pypi=None*, ***kwargs*)

Require a module by name from micropython-lib.

Optionally specify unix_ffi=True to use a module from the unix-ffi directory.

Optionally specify pipy="package-name" to indicate that this should use the named package from PyPI when building for CPython.

package(*package_path*, *files=None*, *base_path='.'*, *opt=None*)

Define a package, optionally restricting to a set of files.

Simple case, a package in the current directory:
package("foo")

will include all .py files in foo, and will be stored as foo/bar/baz.py.

If the package isn't in the current directory, use base_path:
package("foo", base_path="src")

To restrict to certain files in the package use files (note: paths should be relative to the package):
package("foo", files=["bar/baz.py"])

module(*module_path*, *base_path='.'*, *opt=None*)

Include a single Python file as a module.

If the file is in the current directory:
module("foo.py")

Otherwise use base_path to locate the file:
module("foo.py", "src/drivers")

_freeze_internal(*path*, *script*, *exts*, *kind*, *opt*)

freeze(*path*, *script=None*, *opt=None*)

Freeze the input, automatically determining its type. A .py script will be compiled to a .mpy first then frozen, and a .mpy file will be frozen directly.

path must be a directory, which is the base directory to _search for files from. When importing the resulting frozen modules, the name of the module will start after *path*, ie *path* is excluded from the module name.

If *path* is relative, it is resolved to the current manifest.py. Use $(MPY_DIR), $(MPY_LIB_DIR), $(PORT_DIR), $(BOARD_DIR) if you need to access specific paths.

If *script* is None all files in *path* will be frozen.

If *script* is an iterable then freeze() is called on all items of the iterable (with the same *path* and *opt* passed through).

If *script* is a string then it specifies the file or directory to freeze, and can include extra directories before the file or last directory. The file or directory will be _searched for in *path*. If *script* is a directory then all files in that directory will be frozen.

opt is the optimisation level to pass to mpy-cross when compiling .py to .mpy.

freeze_as_str(*path*)

Freeze the given *path* and all .py scripts within it as a string, which will be compiled upon import.

freeze_as_mpy(*path*, *script=None*, *opt=None*)

Freeze the input (see above) by first compiling the .py scripts to .mpy files, then freezing the resulting .mpy files.

freeze_mpy(*path*, *script=None*, *opt=None*)

Freeze the input (see above), which must be .mpy files that are frozen directly.

stubber.utils

Submodules

stubber.utils.config

stubber configuration

Module Contents

Classes

StubberConfig	stubber configuration class

Functions

readconfig([filename, prefix, must_exist])	read the configuration from the pyproject.toml file

Attributes

CONFIG	stubber configuration singleton

class stubber.utils.config.**StubberConfig**(*section_name: str | None = None*, *sources: List[typedconfig.source.ConfigSource] | None = None*, *provider: typedconfig.provider.ConfigProvider | None = None*)

Bases: typedconfig.config.Config

stubber configuration class

Parameters

- **section_name** (*Optional[str]*) –
- **sources** (*Optional[List[typedconfig.source.ConfigSource]]*) –
- **provider** (*Optional[typedconfig.provider.ConfigProvider]*) –

stub_path
: a Path to the stubs directory

fallback_path
: a Path to the fallback stubs directory

repo_path
: a Path to the repo directory

mpy_path
: a Path to the micropython folder in the repos directory

mpy_lib_path
: a Path to the micropython-lib folder in the repos directory

publish_path
: a Path to the folder where all stub publication artefacts are stored

template_path
: a Path to the publication folder that has the template files

stable_version
: last published stable

all_versions
: list of recent versions

BLOCKED_PORTS = ['minimal', 'bare-arm']
: ports that should be ignored as a source of stubs

post_read_hook() → dict

This method can be overridden to modify config values after read() is called. :rtype: A dict of key-value pairs containing new configuration values for key() items in this Config class

Return type
dict

stubber.utils.config.**readconfig**(*filename: str = 'pyproject.toml'*, *prefix: str = 'tool.'*, *must_exist: bool = True*)

read the configuration from the pyproject.toml file

Parameters

- **filename** (*str*) –
- **prefix** (*str*) –
- **must_exist** (*bool*) –

stubber.utils.config.**CONFIG**

stubber configuration singleton

stubber.utils.makeversionhdr

Code from micropyton project and adapted to use the same versioning scheme

Module Contents

Functions

get_version_info_from_git() → Tuple[Union[str, None], ...)	return the version info from the git repository specified.

stubber.utils.makeversionhdr.**get_version_info_from_git**(*path: pathlib.Path = Path.cwd()*) → Tuple[str | None, str | None]

return the version info from the git repository specified. returns: a 2-tuple containing git_tag, short_hash

Parameters
path (*pathlib.Path*) –

Return type
Tuple[Union[str, None], Union[str, None]]

stubber.utils.manifest

Create a *module.json* manifest listing all files/stubs in this folder and subfolders.

Module Contents

Functions

manifest(→ dict)	create a new empty manifest dict
make_manifest(→ bool)	Create a *module.json* manifest listing all files/stubs in this folder and subfolders.

stubber.utils.manifest.**manifest**(*family: str = 'micropython'*, *stubtype: str = 'frozen'*, *machine: str | None = None*, *port: str | None = None*, *platform: str | None = None*, *sysname: str | None = None*, *nodename: str | None = None*, *version: str | None = None*, *release: str | None = None*, *firmware: str | None = None*) → dict

create a new empty manifest dict

Parameters

- **family** (*str*) –
- **stubtype** (*str*) –
- **machine** (*Optional[str]*) –
- **port** (*Optional[str]*) –
- **platform** (*Optional[str]*) –
- **sysname** (*Optional[str]*) –
- **nodename** (*Optional[str]*) –
- **version** (*Optional[str]*) –
- **release** (*Optional[str]*) –
- **firmware** (*Optional[str]*) –

Return type
dict

stubber.utils.manifest.**make_manifest**(*folder: pathlib.Path*, *family: str*, *port: str*, *version: str*, *release: str = ''*, *stubtype: str = ''*, *board: str = ''*) → bool

Create a *module.json* manifest listing all files/stubs in this folder and subfolders.

Parameters

- **folder** (*pathlib.Path*) –
- **family** (*str*) –
- **port** (*str*) –
- **version** (*str*) –
- **release** (*str*) –
- **stubtype** (*str*) –
- **board** (*str*) –

Return type
bool

stubber.utils.post

Pre/Post Processing for createstubs.py

Module Contents

Functions

do_post_processing(stub_paths, stubgen, black, autoflake)	Common post processing
run_black(path[, capture_output])	run black to format the code / stubs
run_autoflake(path[, capture_output, process_pyi])	run autoflake to remove unused imports

stubber.utils.post.**do_post_processing**(*stub_paths: List[pathlib.Path]*, *stubgen: bool*, *black: bool*, *autoflake: bool*)

Common post processing

Parameters

- **stub_paths** (*List[pathlib.Path]*) –
- **stubgen** (*bool*) –
- **black** (*bool*) –
- **autoflake** (*bool*) –

stubber.utils.post.**run_black**(*path: pathlib.Path*, *capture_output: bool = False*)

run black to format the code / stubs

Parameters

- **path** (*pathlib.Path*) –
- **capture_output** (*bool*) –

stubber.utils.post.**run_autoflake**(*path: pathlib.Path*, *capture_output: bool = False*, *process_pyi: bool = False*)

run autoflake to remove unused imports needs to be run BEFORE black otherwise it does not recognize long import from`s. note: is run file-by-file to include processing .pyi files

Parameters

- **path** (*pathlib.Path*) –
- **capture_output** (*bool*) –
- **process_pyi** (*bool*) –

stubber.utils.repos

utility functions to handle to cloned repos needed for stubbing.

Module Contents

Functions

switch(tag, *, mpy_path, mpy_lib_path)	Switch to a specific version of the micropython repos.
read_micropython_lib_commits([filename])	Read a csv with the micropython version and matching micropython-lib commit-hashes
match_lib_with_mpy(→ bool)	
fetch_repos(tag, mpy_path, mpy_lib_path)	Fetch updates, then switch to the provided tag
repo_paths(→ Tuple[pathlib.Path, pathlib.Path])	Return the paths to the micropython and micropython-lib repos, given a path to the repos.'

stubber.utils.repos.**switch**(*tag: str*, *, *mpy_path: pathlib.Path*, *mpy_lib_path: pathlib.Path*)

Switch to a specific version of the micropython repos.

Specify the version with –tag or –version to specify the version tag of the MicroPython repo. The Micropython-lib repo will be checked out to a commit that corresponds in time to that version tag, in order to allow non-current versions to be stubbed correctly.

The repros must be cloned already

Parameters

- **tag** (*str*) –
- **mpy_path** (*pathlib.Path*) –
- **mpy_lib_path** (*pathlib.Path*) –

stubber.utils.repos.**read_micropython_lib_commits**(*filename: str = 'data/micropython_tags.csv'*)

Read a csv with the micropython version and matching micropython-lib commit-hashes these can be used to make sure that the correct micropython-lib version is checked out.

filename is relative to the 'stubber' package

git for-each-ref –sort=creatordate –format '%(refname) %(creatordate)' refs/tags

Parameters

filename (*str*) –

stubber.utils.repos.**match_lib_with_mpy**(*version_tag: str*, *mpy_path: pathlib.Path*, *lib_path: pathlib.Path*) → bool

Parameters

- **version_tag** (*str*) –
- **mpy_path** (*pathlib.Path*) –
- **lib_path** (*pathlib.Path*) –

Return type
bool

stubber.utils.repos.**fetch_repos**(*tag: str*, *mpy_path: pathlib.Path*, *mpy_lib_path: pathlib.Path*)

Fetch updates, then switch to the provided tag

Parameters

- **tag** (*str*) –
- **mpy_path** (*pathlib.Path*) –
- **mpy_lib_path** (*pathlib.Path*) –

stubber.utils.repos.**repo_paths**(*dest_path: pathlib.Path*) → Tuple[pathlib.Path, pathlib.Path]

Return the paths to the micropython and micropython-lib repos, given a path to the repos.'

Parameters
dest_path (*pathlib.Path*) –

Return type
Tuple[pathlib.Path, pathlib.Path]

stubber.utils.stubmaker

Generate stub files for micropython modules using mypy/stubgen

Module Contents

Functions

generate_pyi_from_file(→ bool)	Generate a .pyi stubfile from a single .py module using mypy/stubgen
generate_pyi_files(→ bool)	Generate typeshed files for all scripts in a folder using mypy/stubgen

Attributes

STUBGEN_OPT

stubber.utils.stubmaker.**STUBGEN_OPT**

stubber.utils.stubmaker.**generate_pyi_from_file**(*file: pathlib.Path*) → bool

Generate a .pyi stubfile from a single .py module using mypy/stubgen

Parameters
file (*pathlib.Path*) –

Return type
bool

stubber.utils.stubmaker.**generate_pyi_files**(*modules_folder: pathlib.Path*) → bool

Generate typeshed files for all scripts in a folder using mypy/stubgen

Returns: False if one or more files had an issue generating a stub

Parameters
modules_folder (*pathlib.Path*) –

Return type
bool

stubber.utils.typed_config_toml

typed-config-toml

Extend typed-config to read configuration from .toml files

Module Contents

Classes

TomlConfigSource	Read configuration from a .toml file

class stubber.utils.typed_config_toml.**TomlConfigSource**(*filename: str*, *prefix: str | None = None*, *must_exist: bool = True*)

Bases: typedconfig.source.ConfigSource

Read configuration from a .toml file

prefix is used to allow for toml nested configuration a common prefix = "tool."

` #pyproject.toml [tool.deadparrot] species = "Norwegian Blue" state = "resting" details = ["pinging","Lovely plumage","3"] ` Use the below code to retrieve: ` # TODO sample code `

Parameters

- **filename** (*str*) –
- **prefix** (*Optional[str]*) –
- **must_exist** (*bool*) –

get_config_value(*section_name: str*, *key_name: str*) → str | None

Parameters

- **section_name** (*str*) –
- **key_name** (*str*) –

Return type
Optional[str]

stubber.utils.versions

Handle versions of micropython based on the git tags in the repo

Module Contents

Functions

clean_version(version, *[, build, patch, commit, ...])	Clean up and transform the many flavours of versions
checkedout_version(→ str)	Get the checked-out version of the repo
micropython_versions([start])	Get the list of micropython versions from github tags

Attributes

OLDEST_VERSION	This is the oldest MicroPython version to build the stubs on
V_PREVIEW	Latest preview version
SET_PREVIEW	

stubber.utils.versions.**OLDEST_VERSION = '1.16'**

This is the oldest MicroPython version to build the stubs on

stubber.utils.versions.**V_PREVIEW = 'preview'**

Latest preview version

stubber.utils.versions.**SET_PREVIEW**

stubber.utils.versions.**clean_version**(*version: str*, *, *build: bool = False*, *patch: bool = False*, *commit: bool = False*, *drop_v: bool = False*, *flat: bool = False*)

Clean up and transform the many flavours of versions

Parameters

- **version** (*str*) –
- **build** (*bool*) –
- **patch** (*bool*) –
- **commit** (*bool*) –
- **drop_v** (*bool*) –
- **flat** (*bool*) –

stubber.utils.versions.**checkedout_version**(*path: pathlib.Path*, *flat: bool = False*) → str

Get the checked-out version of the repo

Parameters

- **path** (*pathlib.Path*) –
- **flat** (*bool*) –

Return type
str

stubber.utils.versions.**micropython_versions**(*start: str = 'v1.9.2'*)

Get the list of micropython versions from github tags

Parameters
start (*str*) –

Package Contents

Functions

make_manifest(→ bool)	Create a *module.json* manifest listing all files/stubs in this folder and subfolders.
manifest	Create a *module.json* manifest listing all files/stubs in this folder and subfolders.
do_post_processing(stub_paths, stubgen, black, autoflake)	Common post processing
generate_pyi_files(→ bool)	Generate typeshed files for all scripts in a folder using mypy/stubgen
generate_pyi_from_file(→ bool)	Generate a .pyi stubfile from a single .py module using mypy/stubgen
checkedout_version(→ str)	Get the checked-out version of the repo
clean_version(version, *[, build, patch, commit, ...])	Clean up and transform the many flavours of versions

stubber.utils.**make_manifest**(*folder: pathlib.Path*, *family: str*, *port: str*, *version: str*, *release: str = ''*, *stubtype: str = ''*, *board: str = ''*) → bool

Create a *module.json* manifest listing all files/stubs in this folder and subfolders.

Parameters

- **folder** (*pathlib.Path*) –
- **family** (*str*) –
- **port** (*str*) –
- **version** (*str*) –
- **release** (*str*) –
- **stubtype** (*str*) –
- **board** (*str*) –

Return type
bool

stubber.utils.**manifest**(*family: str = 'micropython'*, *stubtype: str = 'frozen'*, *machine: str | None = None*, *port: str | None = None*, *platform: str | None = None*, *sysname: str | None = None*, *nodename: str | None = None*, *version: str | None = None*, *release: str | None = None*, *firmware: str | None = None*) → dict

create a new empty manifest dict

Parameters

- **family** (*str*) –
- **stubtype** (*str*) –
- **machine** (*Optional[str]*) –
- **port** (*Optional[str]*) –
- **platform** (*Optional[str]*) –
- **sysname** (*Optional[str]*) –
- **nodename** (*Optional[str]*) –
- **version** (*Optional[str]*) –
- **release** (*Optional[str]*) –
- **firmware** (*Optional[str]*) –

Return type
dict

stubber.utils.**do_post_processing**(*stub_paths: List[pathlib.Path]*, *stubgen: bool*, *black: bool*, *autoflake: bool*)

Common post processing

Parameters

- **stub_paths** (*List[pathlib.Path]*) –
- **stubgen** (*bool*) –
- **black** (*bool*) –
- **autoflake** (*bool*) –

stubber.utils.**generate_pyi_files**(*modules_folder: pathlib.Path*) → bool

Generate typeshed files for all scripts in a folder using mypy/stubgen

Returns: False if one or more files had an issue generating a stub

Parameters
modules_folder (*pathlib.Path*) –

Return type
bool

stubber.utils.**generate_pyi_from_file**(*file: pathlib.Path*) → bool

Generate a .pyi stubfile from a single .py module using mypy/stubgen

Parameters
file (*pathlib.Path*) –

Return type
bool

stubber.utils.**checkedout_version**(*path: pathlib.Path*, *flat: bool = False*) → str

Get the checked-out version of the repo

Parameters

- **path** (*pathlib.Path*) –
- **flat** (*bool*) –

Return type
str

stubber.utils.**clean_version**(*version: str*, *, *build: bool = False*, *patch: bool = False*, *commit: bool = False*, *drop_v: bool = False*, *flat: bool = False*)

Clean up and transform the many flavours of versions

Parameters

- **version** (*str*) –
- **build** (*bool*) –
- **patch** (*bool*) –
- **commit** (*bool*) –
- **drop_v** (*bool*) –
- **flat** (*bool*) –

19.1.2 Submodules

stubber.basicgit

Simple Git module, where needed via powershell

Some of the functions are based on the PyGithub module

Module Contents

Functions

_run_local_git(cmd[, repo, expect_stderr, ...])	run a external (git) command in the repo's folder and deal with some of the errors
clone(→ bool)	git clone [--depth 1] [--branch <tag_name>] <remote> <directory>
get_local_tag(→ Union[str, None])	get the most recent git version tag of a local repo
get_local_tags(→ List[str])	get list of all tags of a local repo
get_tags(→ List[str])	Get list of tag of a repote github repo
checkout_tag(→ bool)	checkout a specific git tag
sync_submodules(→ bool)	make sure any submodules are in sync
checkout_commit(→ bool)	Checkout a specific commit
switch_tag(→ bool)	switch to the specified version tag of a local repo
switch_branch(→ bool)	switch to the specified branch in a local repo"
fetch(→ bool)	fetches a repo
pull(→ bool)	pull a repo origin into main
get_git_describe([folder])	Based on MicroPython makeversionhdr.py

Attributes

PAT_NO_ACCESS
PAT
GH_CLIENT

stubber.basicgit.**PAT_NO_ACCESS**

stubber.basicgit.**PAT**

stubber.basicgit.**GH_CLIENT**

stubber.basicgit.**_run_local_git**(*cmd: List[str]*, *repo: pathlib.Path | str | None = None*, *expect_stderr=False*, *capture_output=True*, *echo_output=True*)

run a external (git) command in the repo's folder and deal with some of the errors

Parameters

- **cmd** (*List[str]*) –
- **repo** (*Optional[Union[pathlib.Path, str]]*) –

stubber.basicgit.**clone**(*remote_repo: str*, *path: pathlib.Path*, *shallow: bool = False*, *tag: str | None = None*) → bool

git clone [–depth 1] [–branch <tag_name>] <remote> <directory>

Parameters

- **remote_repo** (*str*) –
- **path** (*pathlib.Path*) –
- **shallow** (*bool*) –
- **tag** (*Optional[str]*) –

Return type
bool

stubber.basicgit.**get_local_tag**(*repo: str | pathlib.Path | None = None*, *abbreviate: bool = True*) → str | None

get the most recent git version tag of a local repo repo Path should be in the form of : repo = "./repo/micropython"

returns the tag or None

Parameters

- **repo** (*Optional[Union[str, pathlib.Path]]*) –
- **abbreviate** (*bool*) –

Return type
Union[str, None]

stubber.basicgit.**get_local_tags**(*repo: pathlib.Path | None = None*, *minver: str | None = None*) → List[str]

get list of all tags of a local repo

Parameters

- **repo** (*Optional[pathlib.Path]*) –
- **minver** (*Optional[str]*) –

Return type
List[str]

stubber.basicgit.**get_tags**(*repo: str*, *minver: str | None = None*) → List[str]

Get list of tag of a repote github repo

Parameters

- **repo** (*str*) –
- **minver** (*Optional[str]*) –

Return type
List[str]

stubber.basicgit.**checkout_tag**(*tag: str*, *repo: str | pathlib.Path | None = None*) → bool

checkout a specific git tag

Parameters

- **tag** (*str*) –
- **repo** (*Optional[Union[str, pathlib.Path]]*) –

Return type
bool

stubber.basicgit.**sync_submodules**(*repo: pathlib.Path | str | None = None*) → bool

make sure any submodules are in sync

Parameters
repo (*Optional[Union[pathlib.Path, str]]*) –

Return type
bool

stubber.basicgit.**checkout_commit**(*commit_hash: str*, *repo: pathlib.Path | str | None = None*) → bool

Checkout a specific commit

Parameters

- **commit_hash** (*str*) –
- **repo** (*Optional[Union[pathlib.Path, str]]*) –

Return type
bool

stubber.basicgit.**switch_tag**(*tag: str | pathlib.Path*, *repo: pathlib.Path | str | None = None*) → bool

switch to the specified version tag of a local repo repo should be in the form of : path/.git repo = '../micropython/.git' returns None

Parameters

- **tag** (*Union[str, pathlib.Path]*) –
- **repo** (*Optional[Union[pathlib.Path, str]]*) –

Return type
bool

stubber.basicgit.**switch_branch**(*branch: str*, *repo: pathlib.Path | str | None = None*) → bool

switch to the specified branch in a local repo" repo should be in the form of : path/.git repo = '../micropython/.git' returns None

Parameters

- **branch** (*str*) –
- **repo** (*Optional[Union[pathlib.Path, str]]*) –

Return type
bool

stubber.basicgit.**fetch**(*repo: pathlib.Path | str*) → bool

fetches a repo repo should be in the form of : path/.git repo = '../micropython/.git' returns True on success

Parameters
repo (*Union[pathlib.Path, str]*) –

Return type
bool

stubber.basicgit.**pull**(*repo: pathlib.Path | str*, *branch: str = 'main'*) → bool

pull a repo origin into main repo should be in the form of : path/.git repo = '../micropython/.git' returns True on success

Parameters

- **repo** (*Union[pathlib.Path, str]*) –
- **branch** (*str*) –

Return type
bool

stubber.basicgit.**get_git_describe**(*folder: str | None = None*)

Based on MicroPython makeversionhdr.py returns : current git tag, commits ,commit hash : "v1.19.1-841-g3446"

Parameters
folder (*Optional[str]*) –

stubber.cst_transformer

helper functions for stub transformations

Module Contents

Classes

TypeInfo	contains the functiondefs and classdefs info read from the stubs source
StubTypingCollector	Collect the function/method and class definitions from the stubs source

Functions

update_def_docstr(→ Any)	Update the docstring of a function/method or class
update_module_docstr(→ Any)	Add or update the docstring of a module

Attributes

MODULE_KEY
MODDOC_KEY
_m

class stubber.cst_transformer.**TypeInfo**

contains the functiondefs and classdefs info read from the stubs source

name: str

decorators: Sequence[libcst.Decorator]

params: libcst.Parameters | None

returns: libcst.Annotation | None

docstr_node: libcst.SimpleStatementLine | None

def_node: libcst.FunctionDef | libcst.ClassDef | None

def_type: str = '?'

exception stubber.cst_transformer.**TransformError**

Bases: Exception

Error raised upon encountering a known error while attempting to transform the tree.

stubber.cst_transformer.**MODULE_KEY = ('__module',)**

stubber.cst_transformer.**MODDOC_KEY = ('__module_docstring',)**

stubber.cst_transformer.**_m**

class stubber.cst_transformer.**StubTypingCollector**

Bases: libcst.CSTVisitor

Collect the function/method and class definitions from the stubs source

visit_Module(*node: libcst.Module*) → bool

Store the module docstring

Parameters
node (*libcst.Module*) –

Return type
bool

visit_Comment(*node: libcst.Comment*) → None

connect comments from the source

Parameters
node (*libcst.Comment*) –

Return type
None

visit_ClassDef(*node: libcst.ClassDef*) → bool | None

collect info from a classdef: - name, decorators, docstring

Parameters
node (*libcst.ClassDef*) –

Return type
Optional[bool]

leave_ClassDef(*original_node: libcst.ClassDef*) → None

remove the class name from the stack

Parameters
original_node (*libcst.ClassDef*) –

Return type
None

visit_FunctionDef(*node: libcst.FunctionDef*) → bool | None

collect info from a function/method - name, decorators, params, returns, docstring

Parameters
node (*libcst.FunctionDef*) –

Return type
Optional[bool]

update_append_first_node(*node*)

Store the function/method docstring or function/method sig

leave_FunctionDef(*original_node: libcst.FunctionDef*) → None

remove the function/method name from the stack

Parameters
original_node (*libcst.FunctionDef*) –

Return type
None

stubber.cst_transformer.**update_def_docstr**(*dest_node: libcst.FunctionDef | libcst.ClassDef*, *src_comment: libcst.SimpleStatementLine | None*, *src_node=None*) → Any

Update the docstring of a function/method or class

for functiondefs ending in an ellipsis, the entire body needs to be replaced. in this case the src_body is mandatory.

Parameters

- **dest_node** (*Union[libcst.FunctionDef, libcst.ClassDef]*) –
- **src_comment** (*Optional[libcst.SimpleStatementLine]*) –

Return type
Any

stubber.cst_transformer.**update_module_docstr**(*node: libcst.Module*, *doc_tree: str | libcst.SimpleStatementLine | libcst.BaseCompoundStatement | None*) → Any

Add or update the docstring of a module

Parameters

- **node** (*libcst.Module*) –
- **doc_tree** (*Optional[Union[str, libcst.SimpleStatementLine, libcst.BaseCompoundStatement]]*) –

Return type
Any

stubber.downloader

Download files from a public github repo

Module Contents

Functions

download_file(url, module[, folder])	dowload a file from a public github repo
download_files(repo, frozen_modules, savepath)	dowload multiple files from a public github repo

stubber.downloader.**download_file**(*url: str*, *module: str*, *folder: str = './'*)

dowload a file from a public github repo

Parameters

- **url** (*str*) –
- **module** (*str*) –
- **folder** (*str*) –

stubber.downloader.**download_files**(*repo*, *frozen_modules*, *savepath*)

dowload multiple files from a public github repo

stubber.get_cpython

Download or update the micropyton compatibility modules from pycopy and stores them in the all_stubs folder The all_stubs folder should be mapped/symlinked to the micropython_stubs/stubs repo/folder

Module Contents

Functions

get_core(requirements[, stub_path, family])	Download MicroPython compatibility modules

stubber.get_cpython.**get_core**(*requirements: str*, *stub_path: str | pathlib.Path | None = None*, *family: str = 'core'*)

Download MicroPython compatibility modules

Parameters

- **requirements** (*str*) –
- **stub_path** (*Optional[Union[str, pathlib.Path]]*) –
- **family** (*str*) –

stubber.get_lobo

Collect modules and python stubs from the Loboris MicroPython source project and stores them in the all_stubs folder
The all_stubs folder should be mapped/symlinked to the micropython_stubs/stubs repo/folder

Module Contents

Functions

get_frozen([stub_path, repo, version])	Download Loboris frozen modules direct from github repo

Attributes

FAMILY
PORT

stubber.get_lobo.**FAMILY = 'loboris'**

stubber.get_lobo.**PORT = 'esp32_lobo'**

stubber.get_lobo.**get_frozen**(*stub_path: pathlib.Path | None = None*, ***, *repo: str | None = None*, *version: str = '3.2.24'*)

Download Loboris frozen modules direct from github repo

Parameters

- **stub_path** (*Optional[pathlib.Path]*) –

- **repo** (*Optional[str]*) –
- **version** (*str*) –

stubber.minify

Processing for createstubs.py Minimizes and cross-compiles a MicroPyton file.

Module Contents

Functions

get_whitespace_context(content, index)	Get whitespace count of lines surrounding index
edit_lines(content, edits[, diff])	Edit string by list of edits
minify_script(→ str)	Minifies createstubs.py and variants
reduce_log_print(keep_report, diff, source_content)	
minify(source, target[, keep_report, diff])	Minifies and compiles a script
cross_compile(source, target[, version])	Runs mpy-cross on a (minified) script
pipx_mpy_cross(version, source_file, _target)	Run mpy-cross using pipx
write_to_temp_file(source)	Writes a string to a temp file and returns the Path object
get_temp_file([prefix, suffix])	Get temp file and returns the Path object

Attributes

StubSource
XCompileDest
LineEdits

stubber.minify.**StubSource**

stubber.minify.**XCompileDest**

stubber.minify.**LineEdits**

stubber.minify.**get_whitespace_context**(*content: List[str]*, *index: int*)

Get whitespace count of lines surrounding index

Parameters

- **content** (*List[str]*) –
- **index** (*int*) –

stubber.minify.**edit_lines**(*content: str*, *edits: LineEdits*, *diff: bool = False*)

Edit string by list of edits

Parameters

- **content** (*str*) – content to edit
- **edits** (*[(str, str)]*) – List of edits to make. The first string in the tuple representsthe type of edit to make, can be either: - comment - comment text out (removed on minify) - rprint - replace text with print - rpass - replace text with pass The second string is the matching text to replace
- **diff** (*bool, optional*) – Prints diff of each edit. Defaults to False.

Returns
edited string

Return type
str

stubber.minify.**minify_script**(*source_script:* StubSource, *keep_report: bool = True*, *diff: bool = False*) → str

Minifies createstubs.py and variants

Args: source_script:

- (str): content to edit
- (Path): path to file to edit
- (IOBase): file-like object to edit

keep_report (bool, optional): Keeps single report line in createstubs
Defaults to True.

diff (bool, optional): Print diff from edits. Defaults to False.

Returns
minified source text

Return type
str

Parameters

- **source_script** (StubSource) –
- **keep_report** (*bool*) –
- **diff** (*bool*) –

stubber.minify.**reduce_log_print**(*keep_report*, *diff*, *source_content*)

stubber.minify.**minify**(*source:* StubSource, *target:* StubSource, *keep_report: bool = True*, *diff: bool = False*)

Minifies and compiles a script

Parameters

- **source** (StubSource) –
- **target** (StubSource) –
- **keep_report** (*bool*) –
- **diff** (*bool*) –

stubber.minify.**cross_compile**(*source:* StubSource, *target: XCompileDest*, *version: str = ''*)

Runs mpy-cross on a (minified) script

Parameters

- **source** (StubSource) –
- **target** (*XCompileDest*) –
- **version** (*str*) –

stubber.minify.**pipx_mpy_cross**(*version: str*, *source_file*, *_target*)

Run mpy-cross using pipx

Parameters
version (*str*) –

stubber.minify.**write_to_temp_file**(*source: str*)

Writes a string to a temp file and returns the Path object

Parameters
source (*str*) –

stubber.minify.**get_temp_file**(*prefix: str = 'mpy_cross_'*, *suffix: str = '.py'*)

Get temp file and returns the Path object

Parameters

- **prefix** (*str*) –
- **suffix** (*str*) –

stubber.stubber

Create, Process, and Maintain stubs for MicroPython

stubber.stubs_from_docs

Read the Micropython library documentation files and use them to build stubs that can be used for static typechecking using a custom-built parser to read and process the micropython RST files

Module Contents

Functions

generate_from_rst(→ int)	
clean_destination(dst_path)	Remove all .py/.pyi files in desination folder to avoid left-behinds
get_rst_sources(→ List[pathlib.Path])	Get the list of rst files to process
make_docstubs(dst_path, v_tag, release, suffix, files)	Create the docstubs

stubber.stubs_from_docs.**generate_from_rst**(*rst_path: pathlib.Path*, *dst_path: pathlib.Path*, *v_tag: str*, *release: str | None = None*, *pattern: str = '*.rst'*, *suffix: str = '.py'*, *black: bool = True*) → int

Parameters

- **rst_path** (*pathlib.Path*) –

- **dst_path** (*pathlib.Path*) –
- **v_tag** (*str*) –
- **release** (*Optional[str]*) –
- **pattern** (*str*) –
- **suffix** (*str*) –
- **black** (*bool*) –

Return type
int

stubber.stubs_from_docs.**clean_destination**(*dst_path: pathlib.Path*)

Remove all .py/.pyi files in desination folder to avoid left-behinds

Parameters
dst_path (*pathlib.Path*) –

stubber.stubs_from_docs.**get_rst_sources**(*rst_path: pathlib.Path*, *pattern: str*) → List[pathlib.Path]

Get the list of rst files to process

Parameters

- **rst_path** (*pathlib.Path*) –
- **pattern** (*str*) –

Return type
List[pathlib.Path]

stubber.stubs_from_docs.**make_docstubs**(*dst_path: pathlib.Path*, *v_tag: str*, *release: str*, *suffix: str*, *files: List[pathlib.Path]*)

Create the docstubs

Parameters

- **dst_path** (*pathlib.Path*) –
- **v_tag** (*str*) –
- **release** (*str*) –
- **suffix** (*str*) –
- **files** (*List[pathlib.Path]*) –

stubber.update_fallback

build/update the fallback 'catch-all' stubfolder

Module Contents

Functions

fallback_sources(→ List[Tuple[str, str]])	list of sources to build/update the fallback 'catch-all' stubfolder
update_fallback(stubpath, fallback_path[, version])	update the fallback stubs from the defined sources

Attributes

RELEASED

stubber.update_fallback.**RELEASED = 'v1_18'**

stubber.update_fallback.**fallback_sources**(*version: str*, *fw_version: str | None = None*) → List[Tuple[str, str]]

list of sources to build/update the fallback 'catch-all' stubfolder version : the version to use fw_version : version to source the Firmware stubs from. defaults to the version used , but can be lower

Parameters

- **version** (*str*) –
- **fw_version** (*Optional[str]*) –

Return type
List[Tuple[str, str]]

stubber.update_fallback.**update_fallback**(*stubpath: pathlib.Path*, *fallback_path: pathlib.Path*, *version: str = RELEASED*)

update the fallback stubs from the defined sources

Parameters

- **stubpath** (*pathlib.Path*) –
- **fallback_path** (*pathlib.Path*) –
- **version** (*str*) –

stubber.update_module_list

generate the list of modules that should be attempted to stub for this : - combine the modules from the different texts files - split the lines into individual module names - combine them in one set - remove the ones than cannot be stubbed - remove test modules, ending in *_test* - write updates to:

- board/modulelist.txt
- board/createstubs.py
- TODO: remove the frozen modules from this list
- TODO: bump patch number if there are actual changes

Module Contents

Functions

read_modules(→ Set[str])	read text files with modules per firmware.
update_module_list()	helper script

stubber.update_module_list.**read_modules**(*path: pathlib.Path | None = None*) → Set[str]

read text files with modules per firmware. each contains the output of help("modules") - lines starting with # are comments. - split the other lines at whitespace seperator, - and add each module to a set

Parameters
path (*Optional[pathlib.Path]*) –

Return type
Set[str]

stubber.update_module_list.**update_module_list**()

helper script generate a few lines of code with all modules to be stubbed by createstubs

stubber.variants

Create all variants of createstubs.py - and minify them - and cross compile them

Module Contents

Functions

create_variants(base_path, *, target_path, version, ...)	Create variants of createstubs.py and optionally minify and cross compile them.

Attributes

ALL_VARIANTS
base_path

stubber.variants.**ALL_VARIANTS**

stubber.variants.**create_variants**(*base_path: pathlib.Path, *, target_path: Optional[pathlib.Path] = None, version: str = '', make_variants: List[stubber.codemod.board.CreateStubsVariant] = ALL_VARIANTS[:3], update_modules: bool = True*)

Create variants of createstubs.py and optionally minify and cross compile them.

Parameters

- **base_path** (*Path*) – Path to the base createstubs.py file
- **target_path** (*Path, optional*) – Path to write the variants to, by default None
- **version** (*str, optional*) – Version of mpy-cross to use, by default uses the latest version
- **make_variants** (*List[stubber.codemod.board.CreateStubsVariant]*) –
- **update_modules** (*bool*) –

stubber.variants.**base_path**

19.1.3 Package Contents

stubber.**__version__** = '1.17.2'

19.2 createstubs_mem

Create stubs for (all) modules on a MicroPython board.

This variant of the createstubs.py script is optimised for use on low-memory devices, and reads the list of modules from a text file *modulelist.txt* in the root or *libs* folder that should be uploaded to the device. If that cannot be found then only a single module (micropython) is stubbed. In order to run this on low-memory devices two additional steps are recommended: - minifification, using python-minifier

to reduce overall size, and remove logging overhead.

- cross compilation, using mpy-cross, to avoid the compilation step on the micropython device

This variant was generated from createstubs.py by micropython-stubber v1.17.2

19.2.1 Module Contents

Classes

logging	
Stubber	Generate stubs for modules in firmware

Functions

ensure_folder(path)	Create nested folders if needed
_build(s)	
_info(→ dict[str, str])	
version_str(version)	
get_boardname(→ str)	Read the board name from the boardname.py file that may have been created upfront
get_root(→ str)	Determine the root folder of the device
file_exists(filename)	
show_help()	
read_path(→ str)	get --path from cmdline. [unix/win]
is_micropython(→ bool)	runtime test to determine full or micropython
main()	

Attributes

__version__
ENOENT
_MAX_CLASS_LEVEL
LIBS
log

createstubs_mem.**__version__** = 'v1.17.2'

createstubs_mem.**ENOENT** = 2

createstubs_mem.**_MAX_CLASS_LEVEL** = 2

createstubs_mem.**LIBS** = ['lib', '/lib', '/sd/lib', '/flash/lib', '.']

class createstubs_mem.**logging**

INFO = 20

WARNING = 30

ERROR = 40

level

prnt

static getLogger(*name*)

classmethod basicConfig(*level*)

info(*msg*)

warning(*msg*)

error(*msg*)

createstubs_mem.**log**

class createstubs_mem.**Stubber**(*path: str = None*, *firmware_id: str = None*)

Generate stubs for modules in firmware

Parameters

- **path** (*str*) –
- **firmware_id** (*str*) –

property flat_fwid

Turn _fwid from 'v1.2.3' into '1_2_3' to be used in filename

get_obj_attributes(*item_instance: object*)

extract information of the objects members and attributes

Parameters
item_instance (*object*) –

add_modules(*modules*)

Add additional modules to be exported

create_all_stubs()

Create stubs for all configured modules

create_one_stub(*module_name: str*)

Parameters
module_name (*str*) –

create_module_stub(*module_name: str*, *file_name: str = None*) → bool

Create a Stub of a single python module

Args: - module_name (str): name of the module to document. This module will be imported. - file_name (Optional[str]): the 'path/filename.pyi' to write to. If omitted will be created based on the module name.

Parameters

- **module_name** (*str*) –
- **file_name** (*str*) –

Return type
bool

write_object_stub(*fp*, *object_expr: object*, *obj_name: str*, *indent: str*, *in_class: int = 0*)

Write a module/object stub to an open file. Can be called recursive.

Parameters

- **object_expr** (*object*) –
- **obj_name** (*str*) –
- **indent** (*str*) –
- **in_class** (*int*) –

clean(*path: str = None*)

Remove all files from the stub folder

Parameters
path (*str*) –

report_start(*filename: str = 'modules.json'*)

Start a report of the modules that have been stubbed "create json with list of exported modules

Parameters
filename (*str*) –

report_add(*module_name: str*, *stub_file: str*)

Add a module to the report

Parameters

- **module_name** (*str*) –
- **stub_file** (*str*) –

report_end()

createstubs_mem.**ensure_folder**(*path: str*)

Create nested folders if needed

Parameters
path (*str*) –

createstubs_mem.**_build**(*s*)

createstubs_mem.**_info**() → dict[str, str]

Return type
dict[str, str]

createstubs_mem.**version_str**(*version: tuple*)

Parameters
version (*tuple*) –

createstubs_mem.**get_boardname**() → str

Read the board name from the boardname.py file that may have been created upfront

Return type
str

createstubs_mem.**get_root**() → str

Determine the root folder of the device

Return type
str

createstubs_mem.**file_exists**(*filename: str*)

Parameters
filename (*str*) –

createstubs_mem.**show_help**()

createstubs_mem.**read_path**() → str

get –path from cmdline. [unix/win]

Return type
str

createstubs_mem.**is_micropython**() → bool

runtime test to determine full or micropython

Return type
bool

createstubs_mem.**main**()

19.3 info

19.3.1 Module Contents

Functions

_info(→ dict[str, str])
find_board(info, board_descr, filename)
file_exists(filename)
_build(s)

Attributes

LIBS

info.**LIBS = ['.', '/lib', '/sd/lib', '/flash/lib', 'lib']**

info.**_info**() → dict[str, str]

Return type
dict[str, str]

info.**find_board**(*info: dict*, *board_descr: str*, *filename: str*)

Parameters

- **info** (*dict*) –
- **board_descr** (*str*) –
- **filename** (*str*) –

info.**file_exists**(*filename: str*)

Parameters
filename (*str*) –

info.**_build**(*s*)

19.4 createstubs

Create stubs for (all) modules on a MicroPython board

19.4.1 Module Contents

Classes

logging	
Stubber	Generate stubs for modules in firmware

Functions

ensure_folder(path)	Create nested folders if needed
_build(s)	
_info(→ dict[str, str])	
version_str(version)	
get_boardname(→ str)	Read the board name from the boardname.py file that may have been created upfront
get_root(→ str)	Determine the root folder of the device
file_exists(filename)	
show_help()	
read_path(→ str)	get --path from cmdline. [unix/win]
is_micropython(→ bool)	runtime test to determine full or micropython
main()	

Attributes

__version__
ENOENT
_MAX_CLASS_LEVEL
LIBS
log

createstubs.**__version__** = 'v1.17.2'

createstubs.**ENOENT** = 2

createstubs.**_MAX_CLASS_LEVEL** = 2

createstubs.**LIBS** = ['lib', '/lib', '/sd/lib', '/flash/lib', '.']

class createstubs.**logging**

INFO = 20

WARNING = 30

ERROR = 40

level

prnt

static getLogger(*name*)

classmethod basicConfig(*level*)

info(*msg*)

warning(*msg*)

error(*msg*)

createstubs.**log**

class createstubs.**Stubber**(*path: str = None*, *firmware_id: str = None*)

Generate stubs for modules in firmware

Parameters

- **path** (*str*) –
- **firmware_id** (*str*) –

property flat_fwid

Turn _fwid from 'v1.2.3' into '1_2_3' to be used in filename

get_obj_attributes(*item_instance: object*)

extract information of the objects members and attributes

Parameters
item_instance (*object*) –

add_modules(*modules*)

Add additional modules to be exported

create_all_stubs()

Create stubs for all configured modules

create_one_stub(*module_name: str*)

Parameters
module_name (*str*) –

create_module_stub(*module_name: str*, *file_name: str = None*) → bool

Create a Stub of a single python module

Args: - module_name (str): name of the module to document. This module will be imported. - file_name (Optional[str]): the 'path/filename.pyi' to write to. If omitted will be created based on the module name.

Parameters

- **module_name** (*str*) –
- **file_name** (*str*) –

Return type
bool

write_object_stub(*fp*, *object_expr: object*, *obj_name: str*, *indent: str*, *in_class: int = 0*)

Write a module/object stub to an open file. Can be called recursive.

Parameters

- **object_expr** (*object*) –
- **obj_name** (*str*) –
- **indent** (*str*) –
- **in_class** (*int*) –

clean(*path: str = None*)

Remove all files from the stub folder

Parameters
path (*str*) –

report_start(*filename: str = 'modules.json'*)

Start a report of the modules that have been stubbed "create json with list of exported modules

Parameters
filename (*str*) –

report_add(*module_name: str*, *stub_file: str*)

Add a module to the report

Parameters

- **module_name** (*str*) –

- **stub_file** (*str*) –

report_end()

createstubs.**ensure_folder**(*path: str*)

Create nested folders if needed

Parameters
path (*str*) –

createstubs.**_build**(*s*)

createstubs.**_info**() → dict[str, str]

Return type
dict[str, str]

createstubs.**version_str**(*version: tuple*)

Parameters
version (*tuple*) –

createstubs.**get_boardname**() → str

Read the board name from the boardname.py file that may have been created upfront

Return type
str

createstubs.**get_root**() → str

Determine the root folder of the device

Return type
str

createstubs.**file_exists**(*filename: str*)

Parameters
filename (*str*) –

createstubs.**show_help**()

createstubs.**read_path**() → str

get –path from cmdline. [unix/win]

Return type
str

createstubs.**is_micropython**() → bool

runtime test to determine full or micropython

Return type
bool

createstubs.**main**()

19.5 createstubs_db

Create stubs for (all) modules on a MicroPython board.

This variant of the createstubs.py script is optimized for use on very-low-memory devices. Note: this version has undergone limited testing.

1) reads the list of modules from a text file *modulelist.txt* that should be uploaded to the device.
2) stored the already processed modules in a text file *modulelist.done*
3) **process the modules in the database:**
    - stub the module
    - update the modulelist.done file
    - reboots the device if it runs out of memory
4) creates the modules.json

If that cannot be found then only a single module (micropython) is stubbed. In order to run this on low-memory devices two additional steps are recommended: - minification, using python-minifierto reduce overall size, and remove logging overhead. - cross compilation, using mpy-cross, to avoid the compilation step on the micropython device

This variant was generated from createstubs.py by micropython-stubber v1.17.2

19.5.1 Module Contents

Classes

logging	
Stubber	Generate stubs for modules in firmware

Functions

ensure_folder(path)	Create nested folders if needed
_build(s)	
_info(→ dict[str, str])	
version_str(version)	
get_boardname(→ str)	Read the board name from the boardname.py file that may have been created upfront
get_root(→ str)	Determine the root folder of the device
file_exists(filename)	
show_help()	
read_path(→ str)	get --path from cmdline. [unix/win]
is_micropython(→ bool)	runtime test to determine full or micropython
get_modules([skip])	
write_skip(done)	
read_skip()	
main()	

Attributes

__version__
ENOENT
_MAX_CLASS_LEVEL
LIBS
log
SKIP_FILE

createstubs_db.**__version__** = 'v1.17.2'

createstubs_db.**ENOENT** = 2

createstubs_db.**_MAX_CLASS_LEVEL** = 2

createstubs_db.**LIBS** = ['lib', '/lib', '/sd/lib', '/flash/lib', '.']

class createstubs_db.**logging**

INFO = 20

WARNING = 30

ERROR = 40

level

prnt

static getLogger(*name*)

classmethod basicConfig(*level*)

info(*msg*)

warning(*msg*)

error(*msg*)

createstubs_db.**log**

class createstubs_db.**Stubber**(*path: str = None*, *firmware_id: str = None*)

Generate stubs for modules in firmware

Parameters

- **path** (*str*) –
- **firmware_id** (*str*) –

property flat_fwid

Turn _fwid from 'v1.2.3' into '1_2_3' to be used in filename

get_obj_attributes(*item_instance: object*)

extract information of the objects members and attributes

Parameters
item_instance (*object*) –

add_modules(*modules*)

Add additional modules to be exported

create_all_stubs()

Create stubs for all configured modules

create_one_stub(*module_name: str*)

Parameters
module_name (*str*) –

create_module_stub(*module_name: str*, *file_name: str = None*) → bool

Create a Stub of a single python module

Args: - module_name (str): name of the module to document. This module will be imported. - file_name (Optional[str]): the 'path/filename.pyi' to write to. If omitted will be created based on the module name.

Parameters

- **module_name** (*str*) –

- **file_name** (*str*) –

Return type
bool

write_object_stub(*fp*, *object_expr: object*, *obj_name: str*, *indent: str*, *in_class: int = 0*)

Write a module/object stub to an open file. Can be called recursive.

Parameters

- **object_expr** (*object*) –
- **obj_name** (*str*) –
- **indent** (*str*) –
- **in_class** (*int*) –

clean(*path: str = None*)

Remove all files from the stub folder

Parameters
path (*str*) –

report_start(*filename: str = 'modules.json'*)

Start a report of the modules that have been stubbed "create json with list of exported modules

Parameters
filename (*str*) –

report_add(*module_name: str*, *stub_file: str*)

Add a module to the report

Parameters

- **module_name** (*str*) –
- **stub_file** (*str*) –

report_end()

createstubs_db.**ensure_folder**(*path: str*)

Create nested folders if needed

Parameters
path (*str*) –

createstubs_db.**_build**(*s*)

createstubs_db.**_info**() → dict[str, str]

Return type
dict[str, str]

createstubs_db.**version_str**(*version: tuple*)

Parameters
version (*tuple*) –

createstubs_db.**get_boardname**() → str

Read the board name from the boardname.py file that may have been created upfront

Return type
str

createstubs_db.**get_root**() → str

Determine the root folder of the device

Return type
str

createstubs_db.**file_exists**(*filename: str*)

Parameters
filename (*str*) –

createstubs_db.**show_help**()

createstubs_db.**read_path**() → str

get –path from cmdline. [unix/win]

Return type
str

createstubs_db.**is_micropython**() → bool

runtime test to determine full or micropython

Return type
bool

createstubs_db.**SKIP_FILE = 'modulelist.done'**

createstubs_db.**get_modules**(*skip=0*)

createstubs_db.**write_skip**(*done*)

createstubs_db.**read_skip**()

createstubs_db.**main**()

19.6 fw_info

19.6.1 Module Contents

Functions

_build(s)	
_version_str(version)	
_info(→ dict[str, str])	

fw_info.**_build**(*s*)

fw_info.**_version_str**(*version: tuple*)

Parameters
version (*tuple*) –

fw_info.**_info**() → dict[str, str]

Return type
dict[str, str]

api/index.rst

```
# Indices and tables

- {ref}`genindex`
- {ref}`modindex`
- {ref}`search`
```

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X