Micropython-Stubber

Release 1.10.0

Jos Verlinde

Nov 23, 2022

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

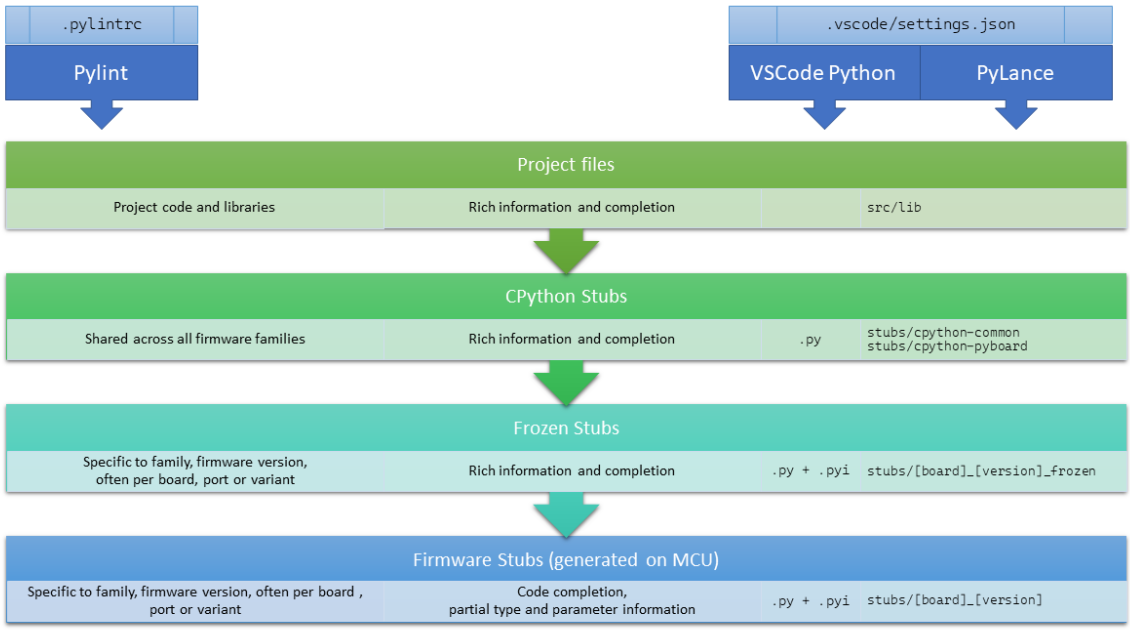
APPROACH TO COLLECTING STUB INFORMATION

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

USING STUBS

3.1 Manual configuration

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

3.2 using micropython-stubber

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

3.3 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

FOUR

VSCODE AND PYLINT CONFIGURATION

The current configuration section describes how to use [Pylance].

> Pylance leverages type stubs (.pyi files) and lazy type inferencing to provide a highly-performant development experience. Pylance supercharges your Python IntelliSense experience with rich type information, helping you write better code, faster.
>
> The Pylance extension is also shipped with a collection of type stubs for popular modules to provide fast and accurate auto-completions and type checking.

Some sections may still refer to the use of [Microsoft Python Language Server][mpls], which has been deprecated.

4.1 Recommended order of the stubs in your config:

1. The src/libs folder(s)
2. The CPython common modules
3. The frozen modules offer more information that can be used in code completion, and therefore should be loaded before the firmware stubs.
4. The firmware stubs generated on or for your board

Announcing Pylance: Fast, feature-rich language support for Python in Visual Studio Code | Python (microsoft.com)

4.2 Relevant VSCode settings

Setting	Default	Description	ref
python.autoComplete.extraPaths	[]	Specifies locations of additional packages for which to load autocomplete data.	Autocomplete Settings
typeshedPaths	[]	Specifies paths to local typeshed repository clone(s) for the Python language server.	Git
python.linting.			Linting Settings
enabled	true	Specifies whether to enable linting in general.	
pylintEnabled	true	Specifies whether to enable Pylint.	

4.2.1 Pylance - pyright

[Pylance](Pylance - Visual Studio Marketplace) is replacing MPLS and provides the same and more functionality.

Setting	Default	Description
python.analysis.stubPath	Typings	Used to allow a user to specify a path to a directory that contains custom type stubs. Each package's type stub file(s) are expected to be in its own subdirectory.
python.analysis.autoSearchPaths	true	Used to automatically add search paths based on some predefined names (like `src`).
python.analysis.extraPaths	[]	Used to specify extra search paths for import resolution. This replaces the old `python.autoComplete.extraPaths` setting.

4.2.2 Sample configuration for Pylance

To update a project configuration from MPLS to Pylance is simple :

Open your VSCode settings file : `.vscode/settings.json`

- change the language server to Pylance `"python.languageServer": "Pylance",`
- remove the section: `python.autoComplete.typeshedPaths`
- remove the section : `python.analysis.typeshedPaths`
- optionally add : `"python.analysis.autoSearchPath": true,`

The result should be something like this :

```
{
     "python.languageServer": "Pylance",
     "python.analysis.autoSearchPath": true,
     "python.autoComplete.extraPaths": [
          "src/lib",
          "all-stubs/cpython_patch",
          "all-stubs/mpy_1_13-nightly_frozen/esp32/GENERIC",
          "all-stubs/esp32_1_13_0-103",
     ]
    "python.linting.enabled": true,
    "python.linting.pylintEnabled": true,
}
```

If you notice problems :

- The paths are case sensitive (which may not be apparent for your platform)
- To allow the config to be used cross platform you can use forward slashes `/`, *note that this is also accepted on Windows*
- If you prefer to use a backslash : in JSON notation the \ (backslash) MUST be escaped as \\ (double backslash)
- Remember to put the 'Frozen' module paths before the generated module paths.

References :

Pylance - Visual Studio Marketplace

microsoft/pyright: Static type checker for Python (github.com)

possible testing / diag :

pyright/command-line.md at microsoft/pyright (github.com)

4.3 pylint

Pylint needs 2 settings :

1. Specify **init-hook** to inform pylint where the stubs are stored. note that the `src` folder is already automagically included, so you do not need to add that.
2. disable some pesky warnings that make no sense for MicroPython, and that are caused by the stubs that have only limited information

File: .pylintrc

```
[MASTER]
# Loaded Stubs:  esp32-micropython-1.11.0
init-hook='import sys;sys.path[1:1] = ["src/lib", "all-stubs/cpython-core", "all-stubs/
↪mpy_1_12/frozen/esp32/GENERIC", "all-stubs/esp32_1_13_0-103",]'

disable = missing-docstring, line-too-long, trailing-newlines, broad-except, logging-
↪format-interpolation, invalid-name,
        no-method-argument, assignment-from-no-return, too-many-function-args,␣
↪unexpected-keyword-arg
        # the 2nd  line deals with the limited information in the generated stubs.
```

4.4 Microsoft Python Language Server settings - Deprecated

MPLS is being replaced by Pylance , and the below configuration is for reference only .

The language server settings apply when python.jediEnabled is false.

Setting	Default	Description	ref
python.jediEnabled	Default: *true*, must be set to FALSE	Indicates whether to use Jedi as the IntelliSense engine (true) or the Microsoft Python Language Server (false). Note that the language server requires a platform that supports .NET Core 2.1 or newer.	
python.analysis.			code analysis settings)
typeshedPaths	[]	Paths to look for typeshed modules on GitHub.	

Our long-term plan is to transition our Microsoft Python Language Server users over to Pylance and eventually deprecate and remove the old language server as a supported option

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

CREATESTUB VARIANTS

There are two variant of the script available, in 3 levels of optimisation:

variant	full documented script	minified script (no logging)	cross-compiled script
full version	board/createstubs.py	minified/createstubs.py	minified/createstubs.mpy
memory optimized	board/createstubs_mem.py	minified/createstubs_mem.py	minified/createstubs_mem.mpy

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.

6.1 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
```

6.2 board/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
```

6.3 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.

6.3.1 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.

6.3.2 Cross compilation

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

CHAPTER

SEVEN

CPYTHON AND FROZEN MODULES

7.1 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. (the .pyi files will be stored alongside the .py files)
4. Include/use them in the configuration

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

7.2 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

7.3 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

EIGHT

DOCUMENTATION STUBS

8.1 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.

8.2 How docstubs are generated

The documentation stubs are generated using `src/stubs_from_docs.py`

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

8.2.1 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

8.2.2 Lookup tables :

- src/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.

8.2.3 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:`

8.2.4 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 coode for this is located in

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

8.2.5 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.

8.2.6 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

NINE

REPO STRUCTURE

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

9.1 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`

9.2 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/*

9.3 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
```

9.4 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.

9.4.1 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
```

9.4.2 Linux/Unix/Mac OS

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

CHAPTER

TEN

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)

10.1 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
```

10.1.1 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

10.1.2 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

10.1.3 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

ELEVEN

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.

11.1 Firmware and libraries

11.1.1 MicroPython firmware and frozen modules *[MIT]*

https://github.com/micropython/micropython

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

11.1.2 Pycopy firmware and frozen modules *[MIT]*

https://github.com/pfalcon/pycopy

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

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

https://github.com/loboris/MicroPython_ESP32_psRAM_LoBo

11.2 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

11.2.1 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

11.2.2 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

TWELVE

REFERENCES

12.1 Inspiration

12.1.1 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

12.1.2 MyPy Stubgen

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

12.1.3 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`

12.2 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

THIRTEEN

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

FOURTEEN

CHANGELOG

CHAPTER

FIFTEEN

V1.6.9

- update board stubber to report pyboards as port=stm32
- update filestructure to reflect this change in nameing

CHAPTER

SIXTEEN

V1.6.8

- fix: `stubber docstubs` failed on methods and function definitions that were split across multiple lines.
- fix: `stubber clone` must fetch and pull to get updates from upstream repos

CHAPTER

SEVENTEEN

V1.6.7

- fix: correct dependencies

CHAPTER

EIGHTEEN

V1.6.6

- fix: allow switch to micropython 1.9.x versions

CHAPTER

NINETEEN

V1.6.4

- unified different scripts into a single CLI tool
- replace submodules with `stubber clone` command

19.1 Tests

- add more clone test variants
- fix incorrect mock
- add minify mock tests
- change coverage reporting to codecov only

19.2 Configuration

- add configuration option via in pyproject.toml
- use config.stub_path for all-stubs
- use typed config
- use configured repo path everywhere

19.3 stubber cli

- usr -t for –tag
- make VERSION_LIST robust
- gracefully handle get tags from non-exitant folder
- use dynamic version list
- add git switch

19.4 common:

- basicgit: accept Paths
- add checkout version, refactor config and version
- refactor postprocessing

19.5 Github Actions

wf: run stubber clone before tests

CHAPTER

TWENTY

V1.6.3 MINOR CLEANUPS

20.1 cli:

- improve help
- add update-fallback to cli
- update toml with config
- refactor functions from cli to utils
- refactor utils
- use repo path
- default clone to repos folder
- add version and logging control
- change cmd `init` to `clone`

20.2 Tests

- testspace: change codecov report type
- refactor and improve tests & mocks

20.3 Developing:

devcontainer: add git graph extension

CHAPTER

TWENTYONE

V1.6.0

CHAPTER

TWENTYTWO

NAMING CONVENTION

- use {family}-v{version}-{port} notation across all scripts and tools
- updated documentation accordingly
- single function to handle version names in various formats (clean_version)
- requirements: pin dependencies to avoid differences when running acriss multiple machines.
- get_mpy_frozen:
  - refactor module
  - checkout the matching commit of micropython-lib withouth this it is not possible to build stubs older versions due to a restructure of micropython-lib
- bulk_stubber:
  - make more robust by including depedencies in the project.
  - detect pyboard based on USB VID & PID
  - prep autostubber for pybv11
- manifest.json generation
  - improve core manifest.json information
  - improve support for grouping and sorting

22.1 stubber cli:

- add init test & refector imports
- fix tests
- merge docstubs into stubber
- reduce tests
- merge get-all-stubs into stubber
- merge get_all_stubs into stubber
- add init
- add stub and some tests
- refactor minify and add -all option

22.2 config

- change to use module tomlli(b)

22.3 common:

- update pyright config to reduce noise

22.4 firmware stubber

fw-stubber: clean up excepts

22.5 Documentation

docs: fix tomllib docs: document the CLI for stubber using sphinx-click readme: add badges for pypi and codecov doc updates fix docbuild for poetry update totdos in source

docstubs: fix json import fix: black formatting across platforms

add settings and script for coverage reporting update snippets (origin/dev/snippets) snippets:add checks

Develop: fix: codespace poetry setup

#test test: fix fewer arguments tests: add firmware stubber version tests test: add missing toml test: add package == version == firmware stubber version tests test: basicgit - add mocktest git test: fix minify test remove unused imports

22.6 Github Actions:

wf: use poetry run and poetry install

pkg: updates to support poetry wf: poetry install verbose pkg: update actions to work with poetry pkg: Move all stubber files into module folder pkg: get version from poetry package pkg: move csv into package config: remove pylance config from vscode settings pkg: add data files

CHAPTER

TWENTYTHREE

USE POETRY

23.1 Dependencies

- bump distro from 1.6.0 to 1.7.0
- bump pytest from 6.2.5 to 7.0.1
- bump myst-parser from 0.16.1 to 0.17.0
- bump coverage from 6.3 to 6.3.1
- bump black from 21.12b0 to 22.1.0
- bump libcst from 0.3.23 to 0.4.1
- bump pytest-mock from 3.6.1 to 3.7.0
- bump mpy-cross from 1.17 to 1.18

23.2 documentation

- Add notes to docstrings/descriptions
- (bluetooth_constants) doc: configure submodules
- docs: add open in VSCode banner
- textual and updates to version notation
- Update 40_firmware_stubs.md

23.3 docstubs:

- fix optional - Optional Parameters are not treated as optional https://github.com/Josverl/micropython-stubber/issues/183
- add docstubs validation
- skip 3 stubs.
- create umodules https://github.com/Josverl/micropython-stubber/issues/176
- allign to micropython PR and update tests
- workaround for re.Match and smarter tests

- fix module & Class constant handling & tests
- Add = `...` to module and class level constants to avoid errors when they are used as a default value in function of nethod

23.4 scripts:

- script update_modulelist, - write updates to: - board/modulelist.txt - board/createsubs.py
- bulk_stubber: Copy generated/firmwarestubs to all-stubs/* use firmware-stubs path for temp storage
- Update BulkStubber & getserial for esp32, esp8266 and stm32
- minify.ps1: posix paths
- bulkstubber: more versions

23.5 Common

- improve black formatting for subfolders
- utils: do not use BaseException
- config: pycache
- data: add firmware modules
- Multi-root with mpy-stdlib based on pico-go
- utils.stubgen: re-apply refactor to avoid use of os.cmd or subprocess.run
- utils.stubgen: refactor to avoid use of os.cmd or subprocess.run

23.6 Validation

- add snippets to test mpy stdlib
- validation: Add more snippets
- update validation snippets
- WS with pyright snippets

23.7 Github Actions:

- wf: do not run minified tests twice, minify all

23.8 Developing:

devcontainer: also init git submodules devcontainer: change python setup

23.9 Tests

test: fix test_socket _class test: fix return type and test deepsleep test: skip test_createstubs on windows + python 3.7 test: native test on windows, include db test on linux test: add createstubs_db on all platforms test: lower theshold to 25 stubs test: add native integration test for micropython_mem test: linux rest on ubuntu and debian, seperate branch / fail logic in docstubs test test: add additional firmwares test: waste less time in by reducing test size test: remove path test debug: fix debug config for tests debug: add missing debug property test: fix pessimistic test and make more robust

CHAPTER

TWENTYFOUR

CREATESUBS V1.5.4

Better exeptions fix missed updates to mem and stub variants

24.1 Change naming convention:

- board: user {family}-v{version}-{port} notation
- change name of master branch to main

24.2 stubber cli

- fix: - rp2.PIO.irq
- fix dequeue
- revert re.match change
- fixup re.match
- add fixes for remaining Documentation errors
- generate Exception Classes
- fix test and default for machine.Pin.**init**
- fix 3 Non-default argument follows default argument errors
- run pyflake to remove unused imports
- fix black parameters
- no need to redefine Exception
- use 'Latest' as a version tag for the most recent master
- use v for version

24.3 Common

- pyright: restore exec environments test & board
- get_frozen: improve force **init**.py
- get_frozen: run black on new/updated stubs
- createstubs: support for RP2 , and accept kwargs for methods and functions
- lobo: skip some modules
- get_mpy: match_lib - add v1.18 rename to .csv
- version: latest
- basicgit: remove debugging code
- basicgit: fix gettag 'latest'
- bulk_stubber: make more robust by adding the dependencies to the repo
- get_frozen: clean collected modules - freeze to empty directory - block CI/CD manifest
- get_mpy: for frozen modules: checkpout the matching commit of micropython-lib
- mpy_frozen: refactor module
- process: –source allows scriptname to be specified & add mpy-cross step
- update_pyi : rename and more efficient updating find .pyi`s created in the wrong location and move them
- Manifiest: release is optional
- use **version** and bump version
- get_mpy: improve porcessing of frozen modules V1.16 and newer
- schema: change stubtype property
- Module manifest: Sorting and add port name
- update get-frozen
- improve frozen manifest processing (#88)
- utils: update clean version
- createstubs* : Add stubtype to manifest - remove redundant try/catch
- process.py: cleanup unused variables
- process.py: use click

24.4 scripts

- add scripts to simplify copy and updates
- Merge generate stubs from documentation

24.5 Firmware stubber

- createstubs: sample bootfile works on pyb & esp
- createstubs: get_root can detect /sd cards

24.6 Dependencies

- python requirements: pin versions
- bump coverage from 6.2 to 6.3
- bump sphinx from 4.3.2 to 4.4.0
- add autoflake
- bump mpy-cross from 1.16 to 1.17
- bump myst-parser from 0.15.2 to 0.16.1
- bump mypy from 0.930 to 0.931
- bump sphinx from 4.3.1 to 4.3.2
- bump rshell from 0.0.30 to 0.0.31

24.7 Documentation

- docs: update naming convention
- fix stuborder image
- Update 10_approach.md
- Naming convention
- one less
- add branch rename instructions
- documented Docs to Stubs process

24.8 validation

- basic setup for stub validation
- add code snippets for validation

24.9 BugFixes:

- fix: ensure that `from __future__ ...` is at start
- add .pyi stubs for umqtt.robust Workaround for missing **init**.py in source
- fix: allow stubgen of async yield in stub (python 3.6) closes #137
- fix Exception lineskip
- fix: get_mpy - save&resore cwd
- fix lobotest to match reduced modules
- Fix core module sort order closes #68

24.10 Github Actions:

- wf test: ensure artefact upload on error
- wf: report coverage
- testspace: use folders
- pytest: upload to testspace
- wf: allign naming

24.11 Scripts:

- prep autostubber for pybv11
- detect pyboard

24.12 code quality:

- fix warnings
- improve core manifest.json

24.13 Tests

- test: fix testregression for latest version
- tests: docstubs improve test of class documented as function
- testspace script
- test: add mpy-cross test and split to seperate folder
- improve grouping
- pytest - use matrix.os in test results
- test: do not run pytest-cov

- tests: drop utf-8 encoding
- test : init logging
- test : fix path
- test: stabelize pyright & black detection
- tests: skip basicgit tests
- add integration tests for cmdline

CHAPTER

TWENTYFIVE

IMPROVE DOCSTUBS

- rst: cleaner import though use of **all**
- document: debug subProcess
- docstubs: fix random.choice(): Any
- docstubs: workaround black on Py3.7

25.1 common:

- add header to modulelist
- devcontainer: simpler requirements
- utils: fix generating .pyi files from .py files with errors
- fix: remove duplication fix minor issues
- github: group logging

CHAPTER

TWENTYSIX

TESTS

- pin python version
- pytest: add test markers
- change ubuntu version detection to codename
- add mark.minfied to linux tests add new platform markers
- implement workaround for failing minified tests
- tests: restructure board test to share testdata
- add debug config
- tests: fix minified tests
- tests: improve & merge testing
- test: linux detection++
- tests: use pathlib and fix paths
- clarify intentional error
- remove duplicate tests
- tests: Restucture Micropython on Cpython tests
- tests: resolve issues due to sloppy imports in tests
- stubgen test : add logging for python 3.7

26.1 minify:

- minify all variants
- also remove _log * lines
- also: minimize createstubs_mem.py
- Fix: remove duplicate builtins from minified
- minify : fix mpy-cross on Python 3.7

26.2 createstubs:

- add normal and mem_constrained versions
- add database variant
- Save state to database and reboot on memory error
- reduce complexity from createstubs to save ram possibly loosing some edge cases ++ docs
- update to handle multi-file uploads to overcome esp8266 memory constraints
- both normal and db work on esp8266 (report not done)
- keep get_root and _info to allow for testing
- update tests to clean-up sys.path after the tests
- keep **version** in minified
- use **version** move unneeded import of machine
- gracefully handle missing `modules.txt`
- stubber: fix firmwareid for mpy esp8266
- stub_lvgl: fix **version**

26.3 Github Actions:

- use posix path
- workflow: test : on checkout fetch all branches and tags

26.4 Docs:

- do not keep generated API Docs
- refactor documenation to more docs
- add explanation om memory optimization.
- add cloning of submodules
- safer path insertion

26.5 Other :

- add remote_stubber script
- bulk_stubber: run black formatter after all downloads
- remote_stubber: improve reporting
- bulk stubber: make esp8266 work more reliable ESP32 WIP - scrips fail to upload

CHAPTER

TWENTYSEVEN

CREATESTUBS: V1.5.1

```
- get_root can detect /sd cards
- sample bootfile works on pyb1.1, esp323, esp8266
```

27.1 Documentation Stubs

- avoid the use of BaseException

27.2 createstubs.py - v1.4.3

- significant memory optimisation for use on low-memory devices such as the esp8266 family
  - load the list of modules to be stubbed from a text file rather than as part of the source
  - use both minification and the `mpy-cross` compiler to reduce the claim on memory (RAM)

Warning: This is a potential breaking change for external tools that expect to either directly execute the script or upload only a single file to an MCU in order to stub.

- the current process is automated in `remote_stubber.ps1'

27.3 createstubs.py - v1.4.2

- Fixes a regression introduced in 1.4-beta where function definitions would include a self parameter.

27.4 minified createstubs.py - v1.4.1

- Switched to use python-minifier for the minification due to the end-of-life of the previous minification tool The new minification tool produces more compact code, although that is still not sufficient for some memory constrained devices.
  - there are no functional changes,
  - the detection of Micropython was adjusted to avoid the use of eval which blocked a minification rule
  - several tests were adjusted

27.5 documentation

- Add Sphinxs documentation
  - changelog
  - automatic API documentation for
    - createstubs.py (board)
    - scripts to run on PC / Github actions
- Publish documentation to readthedocs

27.6 createstubs - v1.4-beta

- createstubs.py
  - improvements to handle nested classes to be able to create stubs for lvgl. this should also benefit other more complex modules.
- added `stub_lvgl.py` helper script

27.7 createstubs.py - v1.3.16

- createstubs.py
  - fix for micropython v1.16
  - skip _test modules in module list
  - black formatting
  - addition of **init** methods (based on runtime / static)
  - class method decorator
  - additional type information for constants using comment style typing
  - detect if running on MicroPython or CPython
  - improve report formatting to list each module on a separate line to allow for better comparison
- workflows
  - move to ubuntu 20.04
    - move to test/tools/ubuntu_20_04/micropython_v1.xx
  - run more tests in GHA
- postprocessing
  - minification adjusted to work with `black`
  - use `mypy.stubgen`
  - run per folder
    - verify 1:1 relation .py-.pyi
    - run `mypy.stubgen` to generate missing .pyi files

  - publish test results to GH

- develop / repo setup
  - updated dev requirements (requirements-dev.txt)
  - enable developing on GitHub codespaces
  - switched to using submodules to remove external dependencies how to clone : `git submodule init` `git submodule update`
  - added black configuration file to avoid running black on minified version
  - switched to using .venv on all platforms
  - added and improved tests
    - test coverage increased to 82%
  - move to test/tools/ubuntu_20_04/micropython_v1.xx
    - for test (git workflows)
    - for tasks
  - make use of CPYTHON stubs to alle makestubs to run well on CPYTHON
    - allows pytest, and debugging of tests
  - add tasks to :
    - run createstubs in linux version

CHAPTER

TWENTYEIGHT

TO-DO (PROVISIONAL)

CHAPTER

TWENTYNINE

UPSTREAM DOCUMENTATION

in docstubs:

29.1 ifconfig

in order to accept `ifconfig()` withouth any parameters from : configtuple: Optional[Any] to : configtuple: Optional[Tuple] = None

29.2 ap.config

from: def config(self, param) -> Any: to: def config(self, param:str="", **kwargs) -> Any:

29.3 write_pulses

Argument of type "Literal[0]" cannot be assigned to parameter "data" of type "bool" in function "write_pulses" "Literal[0]" is incompatible with "bool"

from: def write_pulses(self, duration, data=True) -> Any: to: def write_pulses(self, duration:Union[List, Tuple], data:int=1) -> None:

or even better an overload

```
@overload
def write_pulses(self, duration:Union[List, Tuple], data:int=1) -> None:
    ...
@overload
def write_pulses(self, duration:int, data: Union[List, Tuple]) -> None:
    ...
def write_pulses(self, duration:Union[List, Tuple], data:Union[List, Tuple]) -> None:
```

29.4 working on it

29.4.1 documentation

- how to run post-processing
- how the debug setup works

29.4.2 stubber :

- document - that gc and sys modules are somehow ignored by pylint and will keep throwing errors
- add mpy information to manifest
- use 'nightly' naming convention in createstubs.py
- change firmware naming

29.4.3 frozen stubs

- add simple readme.md ?

29.4.4 Stub augmentation/ merging typeinformation from copied / generated type-rich info

https://libcst.readthedocs.io/en/latest/tutorial.html

- test to auto-merge common prototypes by stubber ie. add common return types to make_stub_files.cfg

29.4.5 Webrepl

Unable to import 'webrepl' can include in common modules C:\develop\MyPython\micropython\extmod\webrepl\webrepl.py

CHAPTER

THIRTY

DEVELOPING

30.1 Cloning the repo

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

poetry install
stubber clone
```

30.2 Windows 10

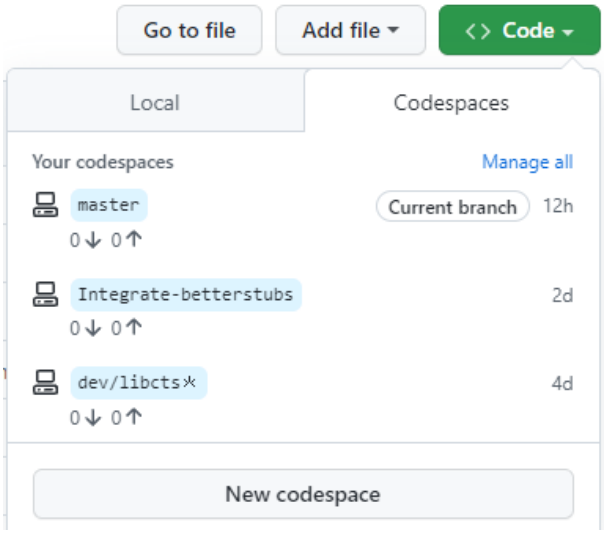
I use Windows 10/11 and use WSL2 to run the linux based parts. if you develop on other platform, it is quite likely that you may need to change some details. if that is needed , please update/add to the documentation and send a documentation PR.

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

30.3 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.



30.4 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.

30.5 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.

30.6 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.

30.7 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"
            }
        },
```

30.8 github actions

30.8.1 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

30.8.2 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

THIRTYONE

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

31.1 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)

31.2 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

31.3 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

THIRTYTWO

API REFERENCE

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

32.1 createstubs

Create stubs for (all) modules on a MicroPython board

32.1.1 Module Contents

Classes

Stubber	Generate stubs for modules in firmware

Functions

ensure_folder(path)	Create nested folders if needed
_info()	collect base information on this runtime
get_root(→ str)	Determine the root folder of the device
file_exists(filename)	
show_help()	
read_path(→ str)	get --path from cmdline. [unix/win]
isMicroPython(→ bool)	runtime test to determine full or micropython
main()	

[1] Created with sphinx-autoapi

Attributes

__version__
ENOENT
_MAX_CLASS_LEVEL
_log

createstubs.**__version__ = 1.9.11**

createstubs.**ENOENT = 2**

createstubs.**_MAX_CLASS_LEVEL = 2**

class createstubs.**Stubber**(*path: str = None*, *firmware_id: str = None*)
Generate stubs for modules in firmware

Parameters

- **path** (*str*) –
- **firmware_id** (*str*) –

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.py' 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*) –

property flat_fwid
Turn _fwid from 'v1.2.3' into '1_2_3' to be used in filename

clean(*path: str = None*)
Remove all files from the stub folder

Parameters **path** (*str*) –

report(*filename: str = 'modules.json'*)
create json with list of exported modules

Parameters **filename** (*str*) –

createstubs.**ensure_folder**(*path: str*)
Create nested folders if needed

Parameters **path** (*str*) –

createstubs.**_info**()
collect base information on this runtime

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.**isMicroPython**() → bool
runtime test to determine full or micropython

Return type bool

createstubs.**main**()

createstubs.**_log**

32.2 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

You should find a cross-compiled version located here: `./minified/createstubs_db.mpy

32.2.1 Module Contents

Classes

Stubber	Generate stubs for modules in firmware

Functions

ensure_folder(path)	Create nested folders if needed
_info()	collect base information on this runtime
get_root(→ str)	Determine the root folder of the device
file_exists(filename)	
show_help()	
read_path(→ str)	get --path from cmdline. [unix/win]
isMicroPython(→ bool)	runtime test to determine full or micropython
main_esp8266()	

Attributes

__version__
ENOENT
_MAX_CLASS_LEVEL
_log

createstubs_db.**__version__** = 1.9.11

createstubs_db.**ENOENT** = 2

createstubs_db.**_MAX_CLASS_LEVEL** = 2

class createstubs_db.**Stubber**(*path: str = None*, *firmware_id: str = None*)

Generate stubs for modules in firmware

Parameters

- **path** (*str*) –
- **firmware_id** (*str*) –

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.py' 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*) –

property flat_fwid
Turn _fwid from 'v1.2.3' into '1_2_3' to be used in filename

clean(*path: str = None*)
Remove all files from the stub folder

Parameters **path** (*str*) –

report(*filename: str = 'modules.json'*)
create json with list of exported modules

Parameters **filename** (*str*) –

createstubs_db.**ensure_folder**(*path: str*)
Create nested folders if needed

Parameters **path** (*str*) –

createstubs_db._info()
collect base information on this runtime

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.isMicroPython() → bool
runtime test to determine full or micropython

Return type bool

createstubs_db.main_esp8266()

createstubs_db._log

32.3 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* 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 additioanl 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

you can find a cross-compiled version located here: ./minified/createstubs_mem.mpy

32.3.1 Module Contents

Classes

Stubber	Generate stubs for modules in firmware

Functions

ensure_folder(path)	Create nested folders if needed
_info()	collect base information on this runtime
get_root(→ str)	Determine the root folder of the device
file_exists(filename)	
show_help()	
read_path(→ str)	get --path from cmdline. [unix/win]
isMicroPython(→ bool)	runtime test to determine full or micropython
main()	

Attributes

__version__
ENOENT
_MAX_CLASS_LEVEL
_log

createstubs_mem.**__version__ = 1.9.11**

createstubs_mem.**ENOENT = 2**

createstubs_mem.**_MAX_CLASS_LEVEL = 2**

class createstubs_mem.**Stubber**(*path: str = None*, *firmware_id: str = None*)

Generate stubs for modules in firmware

Parameters

- **path** (*str*) –
- **firmware_id** (*str*) –

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.py' 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*) –

property flat_fwid
Turn _fwid from 'v1.2.3' into '1_2_3' to be used in filename

clean(*path: str = None*)
Remove all files from the stub folder

Parameters **path** (*str*) –

report(*filename: str = 'modules.json'*)
create json with list of exported modules

Parameters **filename** (*str*) –

createstubs_mem.**ensure_folder**(*path: str*)
Create nested folders if needed

Parameters **path** (*str*) –

createstubs_mem.**_info**()
collect base information on this runtime

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.**isMicroPython**() → bool
runtime test to determine full or micropython

Return type bool

createstubs_mem.**main**()

createstubs_mem.**_log**

32.4 stub_lvgl

Helper module to create stubs for the lvgl modules. 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.

32.4.1 Module Contents

Functions

main()	Create stubs for the lvgl modules using the lvlg version number.

stub_lvgl.**main**()
Create stubs for the lvgl modules using the lvlg version number.

32.5 stubber

read the version from pyproject or the wheels

32.5.1 Subpackages

stubber.codemod

Submodules

stubber.codemod.add_comment

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*) –
- **comments** (*List[str]*) –

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

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

Override this to add arguments to the CLI argument parser. These args will show up when the user invokes `libcst.tool codemod` with `--help`. They will also be presented to your class's `__init__` method. So, if you define a command with an argument 'foo', you should also have a corresponding 'foo' positional or keyword argument in your class's `__init__` method.

Parameters **arg_parser** (*argparse.ArgumentParser*) –

Return type None

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

Parameters **node** (*libcst.Comment*) –

Return type None

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

Parameters

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

Return type libcst.Module

stubber.codemod.enrich

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=False*, *write_back=False*) → Optional[str]

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 containg the doc-stubs
- **diff** – if True, return the diff between the original and the enriched source file
- **write_back** – 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_folder: pathlib.Path*, *docstub_path: pathlib.Path*, *show_diff=False*, *write_back=False*, *require_docstub=False*) → int

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

Returns the number of files enriched.

Parameters

- **source_folder** (*pathlib.Path*) –
- **docstub_path** (*pathlib.Path*) –

Return type int

stubber.codemod.merge_docstub

Module Contents

Classes

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

class stubber.codemod.merge_docstub.**MergeCommand**(*context: libcst.codemod.CodemodContext*, *stub_file: Union[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*) –
- **stub_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

Update the Module docstring

Parameters

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

Return type libcst.Module

visit_ClassDef(*node: libcst.ClassDef*) → Optional[bool]

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*) → Optional[bool]

Parameters **node** (*libcst.FunctionDef*) –

Return type Optional[bool]

leave_FunctionDef(*original_node: libcst.FunctionDef*, *updated_node: libcst.FunctionDef*) → Union[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.commands

Submodules

stubber.commands.cli

command line interface - main group

Module Contents

Functions

stubber_cli(ctx[, verbose])	

stubber.commands.cli.**stubber_cli**(*ctx*, *verbose: int = 0*)

Parameters **verbose** (*int*) –

stubber.commands.clone_cmd

Module Contents

Functions

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

stubber.commands.clone_cmd.**cli_clone**(*path: Union[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

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: Union[str, pathlib.Path]*, *docstubs_folder: Union[str, pathlib.Path]*, *diff=False*, *dry_run=False*)

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]*) –

stubber.commands.get_core_cmd

Module Contents

Functions

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

stubber.commands.get_core_cmd.**cli_get_core**(*stub_folder: str = CONFIG.stub_path.as_posix()*, *pyi: 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*) –
- **pyi** (*bool*) –
- **black** (*bool*) –

stubber.commands.get_docstubs_cmd

get-docstubs

Module Contents

Functions

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

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

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

- **path** (*str*) –
- **target** (*str*) –
- **black** (*bool*) –

stubber.commands.get_frozen_cmd

Module Contents

Functions

cli_get_frozen([stub_folder, version, pyi, 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 = ''*, *pyi: bool = True*, *black: 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*) –
- **pyi** (*bool*) –
- **black** (*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, family)	Enrich the stubs in stub_folder with the docstubs in docstubs_folder.

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

Enrich the stubs in stub_folder with the docstubs in docstubs_folder.

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

stubber.commands.minify_cmd

Module Contents

Functions

cli_minify(→ int)	Minify createstubs*.py.

stubber.commands.minify_cmd.**cli_minify**(*ctx*, *source: Union[str, pathlib.Path]*, *target: Union[str, pathlib.Path]*, *keep_report: bool*, *diff: bool*, *cross_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

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

Return type int

stubber.commands.publish_cmd

Module Contents

Functions

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

Attributes

LAST_VERSION
ALL_VERSIONS
ALL_PORTS
ALL_BOARDS

stubber.commands.publish_cmd.**LAST_VERSION = 1.19.1**

stubber.commands.publish_cmd.**ALL_VERSIONS = ['1.17', '1.18', '1.19', '1.19.1']**

stubber.commands.publish_cmd.**ALL_PORTS = ['stm32', 'esp32', 'esp8266', 'rp2']**

stubber.commands.publish_cmd.**ALL_BOARDS = ['GENERIC']**

stubber.commands.publish_cmd.**cli_publish**(*family: str*, *versions: Union[str, List[str]]*, *ports: Union[str, List[str]]*, *boards: Union[str, List[str]]*, *production: bool*, *dryrun: bool*, *force: bool*, *clean: bool*)

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*) –
- **dryrun** (*bool*) –
- **force** (*bool*) –
- **clean** (*bool*) –

stubber.commands.stub_cmd

Module Contents

Functions

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

stubber.commands.stub_cmd.**cli_stub**(*source: Union[str, pathlib.Path]*)
Create or update .pyi type hint files.

Parameters **source** (*Union[str, pathlib.Path]*) –

stubber.commands.switch_cmd

switch

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: Union[str, pathlib.Path]*, *tag: Optional[str] = None*)
Switch to a specific version of the micropython repos.

Specify the version of the MicroPython repo. use of the –tag or –version is deprocated

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.freeze

Submodules

stubber.freeze.common

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

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_1

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

freeze_one_manifest_1(manifest, stub_path, mpy_path, ...)	get and parse the to-be-frozen .py modules for micropython to extract the static type information

Attributes

FAMILY
regex_port
regex_port_board

stubber.freeze.freeze_manifest_1.**FAMILY = micropython**

stubber.freeze.freeze_manifest_1.**regex_port =**
(?P<port>.*[\\/]+ports[\\/]+\w+)[\\/]+boards[\\/]+\w+

stubber.freeze.freeze_manifest_1.**regex_port_board =**
(?P<board>(?P<port>.*[\\/]+ports[\\/]+\w+)[\\/]+boards[\\/]+\w+)

stubber.freeze.freeze_manifest_1.**freeze_one_manifest_1**(*manifest: pathlib.Path*, *stub_path: pathlib.Path*, *mpy_path: pathlib.Path*, *mpy_lib_path: pathlib.Path*, *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 through the manifest.py introduced in MicroPython 1.12 Suitable for version v1.12 - v1.19 - manifest.py is used for board specific and daily builds - manifest_release.py is used for the release builds

Parameters

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

stubber.freeze.freeze_manifest_2

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: Optional[str] = None*, *board: Optional[str] = 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])	return a list of all manifests.py files found in the ports folder
freeze_any(stub_folder, version[, mpy_path, mpy_lib_path])	get and parse the to-be-frozen .py modules for micropython to extract the static type information

Attributes

FAMILY

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

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

return a list of all manifests.py files found in the ports folder

Parameters **mpy_path** (*pathlib.Path*) –

Return type List[pathlib.Path]

stubber.freeze.get_frozen.**freeze_any**(*stub_folder: pathlib.Path*, *version: str*, *mpy_path: Optional[pathlib.Path] = None*, *mpy_lib_path: Optional[pathlib.Path] = None*)

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 repos should be checked out to the version

Parameters

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

stubber.freeze.makemanifest_1

Classes and functions copied & adapted from micropypythons makemanifest.py to ensure that the manifest.py files can be processed

Module Contents

Classes

IncludeOptions	

Functions

freeze_as_mpy(path[, script, opt])	
freeze_as_str(path)	
freeze_mpy(path[, script, opt])	Freeze the input (see above), which must be .mpy files that are
freeze(path[, script, opt])	Freeze the input, automatically determining its type. A .py script
include(manifest, **kwargs)	Include another manifest.
convert_path(→ str)	Perform variable substitution in path
freeze_internal(path, script[, opt])	Copy the to-be-frozen module to the destination folder to be stubbed.

Attributes

path_vars	
stub_dir	

stubber.freeze.makemanifest_1.**path_vars**

exception stubber.freeze.makemanifest_1.**FreezeError**
Bases: Exception

Common base class for all non-exit exceptions.

class stubber.freeze.makemanifest_1.**IncludeOptions**(***kwargs*)
defaults(***kwargs*)

__getattr__(*name*)

stubber.freeze.makemanifest_1.**freeze_as_mpy**(*path*, *script=None*, *opt=0*)

stubber.freeze.makemanifest_1.**freeze_as_str**(*path*)

stubber.freeze.makemanifest_1.**freeze_mpy**(*path*, *script=None*, *opt=0*)
Freeze the input (see above), which must be .mpy files that are frozen directly.

stubber.freeze.makemanifest_1.**freeze**(*path*, *script=None*, *opt=0*)
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 filename to freeze, and can include extra directories before the file. The file will be searched for in *path*.

opt is the optimisation level to pass to mpy-cross when compiling .py to .mpy. (ignored in this implementation)

stubber.freeze.makemanifest_1.**include**(*manifest: Union[str, List[str]]*, ***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.

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.

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

stubber.freeze.makemanifest_1.**convert_path**(*path: str*) → str
Perform variable substitution in path

Parameters **path** (*str*) –

Return type str

stubber.freeze.makemanifest_1.**stub_dir :str =**

stubber.freeze.makemanifest_1.**freeze_internal**(*path: str*, *script: str*, *opt=None*)
Copy the to-be-frozen module to the destination folder to be stubbed.

Parameters: path (str) : the source path script (str): the source script to be frozen opt (Any): freeze option (ignored)

Parameters

- **path** (*str*) –
- **script** (*str*) –

stubber.publish

Submodules

stubber.publish.bump

Module Contents

Functions

bump_postrelease(→ packaging.version.Version)	Increases the post release version number

stubber.publish.bump.**bump_postrelease**(*current: packaging.version.Version*, *rc: int = 0*) → packaging.version.Version

Increases the post release 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*) –
- **rc** (*int*) –

Return type packaging.version.Version

stubber.publish.candidates

Module Contents

Functions

subfolder_names(path)	returns a list of names of the subfolders of the given path
version_cadidates(→ 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, version, mpy_path])	get list of micropython ports for a given family and version
frozen_candidates(→ Generator[Dict[str, Any], None, None])	generate a list of possible firmware stubs for the given family (, version port and board) ?
docstub_candidates([family, versions, path])	generate a list of possible documentation stub candidates for the given family and version.
firmware_candidates([family, versions, mpy_path])	generate a list of possible firmware stub candidates for the given family and version.

Attributes

OLDEST_VERSION	This is the oldest MicroPython version to build the stubs on
V_LATEST	

stubber.publish.candidates.**OLDEST_VERSION = 1.16**
This is the oldest MicroPython version to build the stubs on

stubber.publish.candidates.**V_LATEST = latest**

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_cadidates**(*suffix: str*, *prefix='.*'*, ***, *path=CONFIG.stub_path*, *oldest=OLDEST_VERSION*) → Generator[str, None, None]
get a list of versions for the given family and suffix

Parameters **suffix** (*str*) –

Return type Generator[str, None, None]

stubber.publish.candidates.**list_frozen_ports**(*family: str = 'micropython'*, *version: str = V_LATEST*, *path=CONFIG.stub_path*)
get list of ports with frozen stubs for a given family and version

Parameters

- **family** (*str*) –
- **version** (*str*) –

stubber.publish.candidates.**list_micropython_ports**(*family: str = 'micropython'*, *version: str = V_LATEST[0]*, *mpy_path=CONFIG.mpy_path*)
get list of micropython ports for a given family and version

Parameters

- **family** (*str*) –
- **version** (*str*) –

stubber.publish.candidates.**frozen_candidates**(*family: str = 'micropython'*, *versions: Union[str, List[str]] = V_LATEST*, *ports: Union[str, List[str]] = 'auto'*, *boards: Union[str, List[str]] = 'auto'*, ***, *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' shoould be specifid in CAPS

Parameters

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

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

stubber.publish.candidates.**docstub_candidates**(*family: str = 'micropython'*, *versions: Union[str, List[str]] = V_LATEST*, *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 exeption of auto which will scan the stubs folder for versions of docstubs

Parameters

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

stubber.publish.candidates.**firmware_candidates**(*family: str = 'micropython'*, *versions: Union[str, List[str]] = V_LATEST*, ***, *mpy_path=CONFIG.mpy_path*)

generate a list of possible firmware stub candidates for the given family and version.

Parameters

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

stubber.publish.database

Module Contents

Functions

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

stubber.publish.database.**get_database**(*publish_path: Union[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.enums

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

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.merge_docstubs

Merge firmware stubs and docstubs into a single folder

Module Contents

Functions

merge_all_docstubs(versions[, family, mpy_path])	merge docstubs into firmware stubs
copy_docstubs(fw_path, dest_path, docstub_path)	**param fw_path** Path to firmware stubs (absolute path)

stubber.publish.merge_docstubs.**merge_all_docstubs**(*versions*, *family: str = 'micropython'*, ***, *mpy_path=CONFIG.mpy_path*)

merge docstubs into firmware stubs

Parameters **family** (*str*) –

stubber.publish.merge_docstubs.**copy_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.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_info(→ Union[Dict, None])	get a package's record from the json db if it can be found
create_package(→ stubber.publish.stubpacker.StubPackage)	create and initialize a package with the correct sources

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

generate a package name for the given package type

Parameters

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

Return type str

stubber.publish.package.**get_package_info**(*db: pysondb.PysonDB*, *pub_path: pathlib.Path*, ***, *pkg_name: str*, *mpy_version: str*) → Union[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=COMBO_STUBS*) → *stubber.publish.stubpacker.StubPackage*

create and initialize a package with the correct sources

Parameters

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

Return type *stubber.publish.stubpacker.StubPackage*

stubber.publish.publish

prepare a set of stub files for publishing to PyPi

required folder structure: NOTE: stubs and publish paths can be located in different locations and repositories

+–stubs - [config.stubs_path] | +–<any stub folders in repo> | +–micropython-v1_18-esp32

+–publish - [config.publish_path] | +–package_data.jsondb | +–package_data_test.jsondb | +–template - [config.template_path] | +–pyproject.toml | +–README.md | +–LICENSE.md | +–<folder for each package> | +–<package name> double nested to match the folder structure | +–<family>-<port>-<board>-<type>-stubs-<version> | +–micropython-esp32-stubs-v1_18 | +–micropython-stm32-stubs-v1_18 | +–micropython-stm32-stubs-v1_19_1 | +–... |

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

Module Contents

Functions

publish(→ Dict[str, Any])	Publish a package to PyPi
publish_one([family, versions, ports, boards, frozen, ...])	Publish a bunch of stub packages
publish_multiple([family, versions, ports, boards, ...])	Publish a bunch of stub packages

stubber.publish.publish.**publish**(*, *db: pysondb.PysonDB*, *pkg_type*, *version: str*, *family='micropython'*, *production*, *dryrun=False*, *force=False*, *clean: bool = False*, *port: str = ''*, *board: str = ''*) → Dict[str, Any]

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

Parameters

- **db** (*pysondb.PysonDB*) –
- **version** (*str*) –
- **clean** (*bool*) –
- **port** (*str*) –
- **board** (*str*) –

Return type Dict[str, Any]

stubber.publish.publish.**publish_one**(*family='micropython'*, *versions: Union[str, List[str]] = 'v1.18'*, *ports: Union[str, List[str]] = 'auto'*, *boards: Union[str, List[str]] = 'GENERIC'*, *frozen: bool = False*, *production=False*, *dryrun: bool = False*, *clean: bool = False*, *force: bool = False*)

Publish a bunch of stub packages

Parameters

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

stubber.publish.publish.**publish_multiple**(*family='micropython'*, *versions: Union[str, List[str]] = ['v1.18', 'v1.19']*, *ports: Union[str, List[str]] = 'auto'*, *boards: Union[str, List[str]] = 'GENERIC'*, *frozen: bool = False*, *production=False*, *dryrun: bool = False*, *clean: bool = False*, *force: bool = False*)

Publish a bunch of stub packages

Parameters

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

stubber.publish.pypi

Module Contents

Functions

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

stubber.publish.pypi.**get_pypy_versions**(*package_name: str*, *base: Optional[packaging.version.Version] = 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.stubpacker

Module Contents

Classes

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

class stubber.publish.stubpacker.**StubPackage**(*package_name: str*, *version: str = '0.0.1'*, *description: str = 'MicroPython stubs'*, *stubs: Optional[List[Tuple[str, pathlib.Path]]] = None*, *json_data: Optional[Dict[str, Any]] = None*)

Create a stub-only package for a specific version 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*) –
- **version** (*str*) –
- **description** (*str*) –
- **stubs** (*Optional[List[Tuple[str, pathlib.Path]]]*) –
- **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

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 pkg_version → str
return the version of the package

Return type str

property pyproject → Union[Dict[str, Any], None]
parsed pyproject.toml or None

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

to_dict()
return the package as a dict to store in the jsondb

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

from_dict(*json_data*)
load the package from a dict (from the jsondb)

update_package_files()

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

`copy_stubs()`
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

`create_readme()`

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)

`create_license()`

Create a license file for the package

- copied from the template license file

`create_update_pyproject_toml()`
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

`update_included_stubs()`
Add the stub files to the pyproject.toml file

`clean()`
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.

`create_hash(include_md=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 has is created across all files, the files are sorted prior to hashing to ensure that the hash is stable.

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

Return type str

`update_hashes()`
Update the pachage hashes

`is_changed()` → bool
Check if the package has changed, based on the current and the stored hash

Return type bool

bump() → str
bump the postrelease version of the package, and write the change to disk

Return type str

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

write_package_json()

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

build() → bool
build the package by running *poetry build*

Return type bool

publish(*production=False*) → bool

Return type bool

stubber.rst

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

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)

Module Contents

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

stubber.rst.lookup.**RST_DOC_FIXES = [['.. method:: match.', '.. method:: Match.'], ['match.end', ' ...**

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

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 = [['*', '*'], ['**', '*'], ['/*', '*'], ["'param'", 'param'], ['lambda', 'lambda_fn'],...**

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'*)

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

Return type str

__add__(*dict:* SourceDict)

Parameters **dict** (SourceDict) –

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

Parameters

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

- **extra** (*int*) –

add_comment(*line: Union[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 = 'Any'*, *docstr: List[str] = []*, *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** (*List[str]*) –
- **autoindent** (*bool*) –

abstract find(*name: str*) → Union[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__()
convert the OD into a string

find(*name: str*) → Union[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: Union[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] = ['""" """']*, *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** (*List[str]*) –
- **init** (*str*) –
- **indent** (*int*) –

class stubber.rst.output_dict.**FunctionSourceDict**(*name: str*, ***, *definition: List[str] = []*, *docstr: List[str] = ['""" """']*, *indent: int = 0*, *decorators: List[str] = []*, *lf='\n'*)
Bases: *SourceDict*

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

Parameters

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

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 typing import IO, Any, Callable, Coroutine, Dict, Generator, Iterator, List, NoReturn,...**

stubber.rst.rst_utils.**simple_candidates**(*type: str*, *match_string: str*, *keywords: List[str]*, *rate: float = 0.5*, *exclude: List[str] = []*)

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** (*List[str]*) –

stubber.rst.rst_utils.**compound_candidates**(*type: str*, *match_string: str*, *keywords: List[str]*, *rate: float = 0.85*, *exclude: List[str] = []*)

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** (*List[str]*) –

stubber.rst.rst_utils.**object_candidates**(*match_string: str*, *rate: float = 0.81*, *exclude: List[str] = ['IRQ']*)

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

Parameters

- **match_string** (*str*) –
- **rate** (*float*) –
- **exclude** (*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: Union[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: Union[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

LOOKUP_LIST
NONE_VERBS
CHILD_PARENT_CLASS
PARAM_FIXES
MODULE_GLUE

continues on next page

Table 60 – continued from previous page

RST_DOC_FIXES
DOCSTUB_SKIP
U_MODULES
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.**LOOKUP_LIST**

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

stubber.rst.**CHILD_PARENT_CLASS**

stubber.rst.**PARAM_FIXES = [['*', '*'], ['**', '*'], ['/*', '*'], ["'param'", 'param'], ['lambda', 'lambda_fn'],...**

stubber.rst.**MODULE_GLUE**

stubber.rst.**RST_DOC_FIXES = [['.. method:: match.', '.. method:: Match.'], ['match.end', ' ...**

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

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

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

Return type str

__add__(*dict:* SourceDict)

Parameters **dict** (SourceDict) –

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

Parameters

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

add_comment(*line: Union[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 = 'Any'*, *docstr: List[str] = []*, *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** (*List[str]*) –
- **autoindent** (*bool*) –

abstract find(*name: str*) → Union[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__()
convert the OD into a string

find(*name: str*) → Union[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: Union[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] = ['""" """']*, *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** (*List[str]*) –
- **init** (*str*) –
- **indent** (*int*) –

class stubber.rst.**FunctionSourceDict**(*name: str*, *, *definition: List[str] = []*, *docstr: List[str] = ['""" """']*, *indent: int = 0*, *decorators: List[str] = []*, *lf='\n'*)

Bases: *SourceDict*

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

Parameters

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

stubber.rst.**simple_candidates**(*type: str*, *match_string: str*, *keywords: List[str]*, *rate: float = 0.5*, *exclude: List[str] = []*)
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** (*List[str]*) –

stubber.rst.**compound_candidates**(*type: str*, *match_string: str*, *keywords: List[str]*, *rate: float = 0.85*, *exclude: List[str] = []*)

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** (*List[str]*) –

stubber.rst.**object_candidates**(*match_string: str*, *rate: float = 0.81*, *exclude: List[str] = ['IRQ']*)

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

Parameters

- **match_string** (*str*) –
- **rate** (*float*) –
- **exclude** (*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: Union[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: Union[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 typing import IO, Any, Callable, Coroutine, Dict, Generator, Iterator, List, NoReturn,...**

stubber.rst.**__all__**

stubber.tools

Submodules

stubber.tools.manifestfile

Module Contents

Classes

ManifestFile	Process a firmware manifest files

exception stubber.tools.manifestfile.**ManifestFileError**
Bases: Exception

Common base class for all non-exit exceptions.

class stubber.tools.manifestfile.**ManifestFile**(*mode: int*, *path_vars: Optional[Dict[str, str]] = None*)
Process a firmware manifest files

params: mode: MODE_*. The mode to process the manifest in, MODE_FREEZE or MODE_COMPILE. path_vars: dict of path variables to substitute in the manifest.

{ "MPY_DIR": (str) path to the micropython repo, "MPY_LIB_DIR": (str) path to the micropython-lib (sub)repo, "PORT_DIR": (str) full path to the port directory, "BOARD_DIR": (str) full path to the board directory,

or the same as PORT_DIR if no port, or the location of a variant's manifest file

}

Parameters

- **mode** (*int*) –
- **path_vars** (*Optional[Dict[str, str]]*) –

_resolve_path(*path*)

_manifest_globals(*kwargs*)

files()

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(*description=None*, *version=None*, *license=None*)
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.

include(*manifest_path*, *top_level=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*, ***kwargs*)
Require a module by name from micropython-lib.

Optionally specify unix_ffi=True to use a module from the unix-ffi directory.

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.tools.pyboard

pyboard interface

This module provides the Pyboard class, used to communicate with and control a MicroPython device over a communication channel. Both real boards and emulated devices (e.g. running in QEMU) are supported. Various communication channels are supported, including a serial connection, telnet-style network connection, external process connection.

Example usage:

import pyboard pyb = pyboard.Pyboard('/dev/ttyACM0')

Or:

pyb = pyboard.Pyboard('192.168.1.1')

Then:

pyb.enter_raw_repl() pyb.exec('import pyb') pyb.exec('pyb.LED(1).on()') pyb.exit_raw_repl()

Note: if using Python2 then pyb.exec must be written as **pyb.exec_**. To run a script from the local machine on the board and print out the results:

import pyboard pyboard.execfile('test.py', device='/dev/ttyACM0')

This script can also be run directly. To execute a local script, use:

./pyboard.py test.py

Or:

python pyboard.py test.py

Module Contents

Classes

TelnetToSerial	
ProcessToSerial	Execute a process and emulate serial connection using its stdin/stdout.
ProcessPtyToTerminal	Execute a process which creates a PTY and prints slave PTY as
Pyboard	

Functions

stdout_write_bytes(b)	
execfile(filename[, device, baudrate, user, password])	
filesystem_command(pyb, args)	
main()	

Attributes

stdout	
_injected_import_hook_code	

stubber.tools.pyboard.**stdout**

stubber.tools.pyboard.**stdout_write_bytes**(*b*)

exception stubber.tools.pyboard.**PyboardError**

Bases: Exception

Common base class for all non-exit exceptions.

class stubber.tools.pyboard.**TelnetToSerial**(*ip*, *user*, *password*, *read_timeout=None*)

__del__()

close()

read(*size=1*)

write(*data*)

inWaiting()

class stubber.tools.pyboard.**ProcessToSerial**(*cmd*)
Execute a process and emulate serial connection using its stdin/stdout.

close()

read(*size=1*)

write(*data*)

inWaiting()

class stubber.tools.pyboard.**ProcessPtyToTerminal**(*cmd*)
Execute a process which creates a PTY and prints slave PTY as first line of its output, and emulate serial connection using this PTY.

close()

read(*size=1*)

write(*data*)

inWaiting()

class stubber.tools.pyboard.**Pyboard**(*device*, *baudrate=115200*, *user='micro'*, *password='python'*, *wait=0*, *exclusive=True*)

close()

read_until(*min_num_bytes*, *ending*, *timeout=10*, *data_consumer=None*)

enter_raw_repl(*soft_reset=True*)

exit_raw_repl()

follow(*timeout*, *data_consumer=None*)

raw_paste_write(*command_bytes*)

exec_raw_no_follow(*command*)

exec_raw(*command*, *timeout=10*, *data_consumer=None*)

eval(*expression*)

exec_(*command*, *data_consumer=None*)

execfile(*filename*)

get_time()

fs_ls(*src*)

fs_cat(*src*, *chunk_size=256*)

fs_get(*src*, *dest*, *chunk_size=256*)

fs_put(*src*, *dest*, *chunk_size=256*)

fs_mkdir(*dir*)

fs_rmdir(*dir*)

fs_rm(*src*)

stubber.tools.pyboard.**execfile**(*filename*, *device='/dev/ttyACM0'*, *baudrate=115200*, *user='micro'*, *password='python'*)

stubber.tools.pyboard.**filesystem_command**(*pyb*, *args*)

stubber.tools.pyboard.**_injected_import_hook_code = Multiline-String**

```
import uos, uio
class _FS:
  class File(uio.IOBase):
    def __init__(self):
      self.off = 0
    def ioctl(self, request, arg):
      return 0
    def readinto(self, buf):
      buf[:] = memoryview(_injected_buf)[self.off:self.off + len(buf)]
      self.off += len(buf)
      return len(buf)
  mount = umount = chdir = lambda *args: None
  def stat(self, path):
    if path == '_injected.mpy':
      return tuple(0 for _ in range(10))
    else:
      raise OSError(-2) # ENOENT
  def open(self, path, mode):
    return self.File()
uos.mount(_FS(), '/_')
uos.chdir('/_')
from _injected import *
uos.umount('/_')
del _injected_buf, _FS
```

stubber.tools.pyboard.**main**()

stubber.utils

Submodules

stubber.utils.config

Module Contents

Classes

StubberConfig	stubber confguration 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: Optional[str] = None*, *sources: Optional[List[typedconfig.source.ConfigSource]] = None*, *provider: Optional[typedconfig.provider.ConfigProvider] = None*)

Bases: typedconfig.config.Config

stubber confguration 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

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([path])	return the version info from the git repository specified.
get_version_build_from_git([path])	

stubber.utils.makeversionhdr.**get_version_info_from_git**(*path: pathlib.Path = Path.cwd()*)
return the version info from the git repository specified. returns: a 2-tuple containing git_tag, short_hash

Parameters **path** (*pathlib.Path*) –

stubber.utils.makeversionhdr.**get_version_build_from_git**(*path: pathlib.Path = Path.cwd()*)

Parameters **path** (*pathlib.Path*) –

stubber.utils.manifest

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: Optional[str] = None, port: Optional[str] = None, platform: Optional[str] = None, sysname: Optional[str] = None, nodename: Optional[str] = None, version: Optional[str] = None, release: Optional[str] = None, firmware: Optional[str] = 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.my_version

find the version of this package

Module Contents

stubber.utils.my_version.**__version__ = 0.0.0**

stubber.utils.my_version.**__version__**

stubber.utils.post

Pre/Post Processing for createstubs.py

Module Contents

Functions

do_post_processing(stub_paths, pyi, black)	Common post processing
run_black(path[, capture_output])	run black to format the code / stubs
run_autoflake(path[, capture_output, progress_pyi])	run autoflake to remove unused imports

stubber.utils.post.**do_post_processing**(*stub_paths: List[pathlib.Path]*, *pyi: bool*, *black: bool*)
Common post processing

Parameters

- **stub_paths** (*List[pathlib.Path]*) –
- **pyi** (*bool*) –
- **black** (*bool*) –

stubber.utils.post.**run_black**(*path: pathlib.Path*, *capture_output=False*)
run black to format the code / stubs

Parameters **path** (*pathlib.Path*) –

stubber.utils.post.**run_autoflake**(*path: pathlib.Path*, *capture_output=False*, *progress_pyi=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*) –

stubber.utils.repos

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 matchin micropython-lib commit-hashes
match_lib_with_mpy(version_tag, lib_path)	

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='data/micropython_tags.csv'*)

Read a csv with the micropython version and matchin 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

TODO: it would be nice if micropython-lib had matching commit-tags

git for-each-ref –sort=creatordate –format '%(refname) %(creatordate)' refs/tags

stubber.utils.repos.**match_lib_with_mpy**(*version_tag: str*, *lib_path: pathlib.Path*)

Parameters

- **version_tag** (*str*) –
- **lib_path** (*pathlib.Path*) –

stubber.utils.stubmaker

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: Optional[str] = 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*) → Optional[str]

Parameters

- **section_name** (*str*) –
- **key_name** (*str*) –

Return type Optional[str]

stubber.utils.versions

Module Contents

Functions

clean_version(version, *[, build, patch, commit, ...])	Clean up and transform the many flavours of versions
micropython_versions([start])	

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.**micropython_versions**(*start='v1.9.2'*)

Package Contents

Functions

readconfig([filename, prefix, must_exist])	read the configuration from the pyproject.toml file
make_manifest(→ bool)	Create a *module.json* manifest listing all files/stubs in this folder and subfolders.
manifest(→ dict)	create a new empty manifest dict
do_post_processing(stub_paths, pyi, black)	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
clean_version(version, *[, build, patch, commit, ...])	Clean up and transform the many flavours of versions

stubber.utils.**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.**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: Optional[str] = None*, *port: Optional[str] = None*, *platform: Optional[str] = None*, *sysname: Optional[str] = None*, *nodename: Optional[str] = None*, *version: Optional[str] = None*, *release: Optional[str] = None*, *firmware: Optional[str] = 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]*, *pyi: bool*, *black: bool*)

Common post processing

Parameters

- **stub_paths** (*List[pathlib.Path]*) –
- **pyi** (*bool*) –
- **black** (*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.**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*) –

32.5.2 Submodules

stubber.basicgit

simple Git module, where needed via powershell

Module Contents

Functions

_run_git(cmd[, repo, expect_stderr, capture_output, ...])	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_tag(→ Union[str, None])	get the most recent git version tag of a local repo
get_tags(→ List[str])	get list of tag of a local repo
checkout_tag(→ bool)	checkout a specific git tag
synch_submodules(→ bool)	make sure any submodules are in syncj
checkout_commit(→ bool)	Checkout a specific commit
switch_tag(→ bool)	get the most recent git version tag of a local repo"
switch_branch(→ bool)	get the most recent git version tag of a local repo"
fetch(→ bool)	fetches a repo
pull(→ bool)	pull a repo origin into main

stubber.basicgit.**_run_git**(*cmd: List[str]*, *repo: Optional[Union[pathlib.Path, str]] = 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=False*, *tag: Optional[str] = None*) → bool

git clone [–depth 1] [–branch <tag_name>] <remote> <directory>

Parameters

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

Return type bool

stubber.basicgit.**get_tag**(*repo: Optional[Union[str, pathlib.Path]] = None*, *abbreviate: bool = True*) → Union[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_tags**(*repo: Optional[pathlib.Path] = None*, *minver: Optional[str] = None*) → List[str]

get list of tag of a local repo

Parameters

- **repo** (*Optional[pathlib.Path]*) –
- **minver** (*Optional[str]*) –

Return type List[str]

stubber.basicgit.**checkout_tag**(*tag: str*, *repo: Optional[Union[str, pathlib.Path]] = None*) → bool

checkout a specific git tag

Parameters

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

Return type bool

stubber.basicgit.**synch_submodules**(*repo: Optional[Union[pathlib.Path, str]] = None*) → bool

make sure any submodules are in syncj

Parameters **repo** (*Optional[Union[pathlib.Path, str]]*) –

Return type bool

stubber.basicgit.**checkout_commit**(*commit_hash: str*, *repo: Optional[Union[pathlib.Path, str]] = 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*, *repo: Optional[Union[pathlib.Path, str]] = None*) → bool

get the most recent git version tag of a local repo" repo should be in the form of : path/.git repo = '../micropython/.git' returns the tag or None

Parameters

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

Return type bool

stubber.basicgit.**switch_branch**(*branch: str*, *repo: Optional[Union[pathlib.Path, str]] = None*) → bool
get the most recent git version tag of a local repo" repo should be in the form of : path/.git repo = '../micropython/.git' returns the tag or None

Parameters

- **branch** (*str*) –
- **repo** (*Optional[Union[pathlib.Path, str]]*) –

Return type bool

stubber.basicgit.**fetch**(*repo: Union[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: Union[pathlib.Path, str]*, *branch='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]*) –

Return type bool

stubber.cst_transformer

Module Contents

Classes

TypeInfo	contains the functiondefs and classdefs info read from the stubs source
StubTypingCollector	The low-level base visitor class for traversing a CST. This should be used in

Functions

update_def_docstr(→ Any)	Update the docstring of a function/method or class
update_module_docstr(→ Any)	Update the docstring of a module

Attributes

MODULE_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 :Optional[libcst.Parameters]

returns :Optional[libcst.Annotation]

docstr_node :Optional[libcst.SimpleStatementLine]

def_node :Optional[Union[libcst.FunctionDef, libcst.ClassDef]]

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**

stubber.cst_transformer.**_m**

class stubber.cst_transformer.**StubTypingCollector**
Bases: libcst.CSTVisitor

The low-level base visitor class for traversing a CST. This should be used in conjunction with the `visit()` method on a `CSTNode` to visit each element in a tree starting with that node. Unlike `CSTTransformer`, instances of this class cannot modify the tree.

When visiting nodes using a `CSTVisitor`, the return value of `visit()` will equal the passed in tree.

visit_Module(*node: libcst.Module*) → bool
Store the module docstring

Parameters **node** (*libcst.Module*) –

Return type bool

visit_ClassDef(*node: libcst.ClassDef*) → Optional[bool]
Store the class docstring

Parameters **node** (*libcst.ClassDef*) –

Return type Optional[bool]

leave_ClassDef(*original_node: libcst.ClassDef*) → None

Parameters **original_node** (*libcst.ClassDef*) –

Return type None

visit_FunctionDef(*node: libcst.FunctionDef*) → Optional[bool]
store each function/method signature

Parameters **node** (*libcst.FunctionDef*) –

Return type Optional[bool]

leave_FunctionDef(*original_node: libcst.FunctionDef*) → None

Parameters **original_node** (*libcst.FunctionDef*) –

Return type None

stubber.cst_transformer.**update_def_docstr**(*dest_node: Union[libcst.FunctionDef, libcst.ClassDef]*, *src_comment: Optional[libcst.SimpleStatementLine]*, *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: Optional[libcst.SimpleStatementLine]*) → Any

Update the docstring of a module

Parameters

- **node** (*libcst.Module*) –
- **doc_tree** (*Optional[libcst.SimpleStatementLine]*) –

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*, *stub_path=None*, *family: str = 'core'*)
Download MicroPython compatibility modules

Parameters **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: Optional[pathlib.Path] = None*, ***, *repo: Optional[str] = None*, *version='3.2.24'*)
Download Loboris frozen modules direct from github repo

Parameters

- **stub_path** (*Optional[pathlib.Path]*) –
- **repo** (*Optional[str]*) –

stubber.minify

Processing for createstubs.py minimizes and cross-compiles a micropyton file.

Module Contents

Functions

edit_lines(content, edits[, diff])	Edit string by list of edits
minify_script(→ str)	minifies createstubs.py
minify(source, target[, keep_report, diff, cross_compile])	

Attributes

python_minifier

stubber.minify.**python_minifier**

stubber.minify.**edit_lines**(*content*, *edits*, *diff=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: pathlib.Path*, *keep_report=True*, *diff=False*) → str
minifies createstubs.py

Parameters

- **keep_report** (*bool, optional*) – Keeps single report line in createstubs Defaults to True.
- **diff** (*bool, optional*) – Print diff from edits. Defaults to False.
- **source_script** (*pathlib.Path*) –

Returns minified source text

Return type str

stubber.minify.**minify**(*source: Union[str, pathlib.Path]*, *target: Union[str, pathlib.Path]*, *keep_report: bool = True*, *diff: bool = False*, *cross_compile: bool = False*)

Parameters

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

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 - 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* 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

RSTReader

Functions

is_balanced(→ bool)	Check if a string has balanced parentheses
generate_from_rst(→ int)	

Attributes

NEW_OUTPUT

SEPERATOR

stubber.stubs_from_docs.**NEW_OUTPUT = True**

stubber.stubs_from_docs.**SEPERATOR = ::**

stubber.stubs_from_docs.**is_balanced**(*s: str*) → bool
Check if a string has balanced parentheses

Parameters **s** (*str*) –

Return type bool

class stubber.stubs_from_docs.**RSTReader**(*v_tag='v1.xx'*)

verbose = False

gather_docs = False

target = .py

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

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

property module_names → List[str]
list of possible module names [uname , name] (longest first)

Return type List[str]

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*) –

leave_class()

read_file(*filename: pathlib.Path*)

Parameters **filename** (*pathlib.Path*) –

prepare_output()
clean up some trailing spaces and commas

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

Parameters **filename** (*pathlib.Path*) –

Return type bool

at_anchor() → bool
Stop at anchor (however .. note: should be added)

Return type bool

at_heading() → bool
stop at heading

Return type bool

parse_docstring() → 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.

Return type List[str]

fix_parameters(*params: str*, *name: str = ''*) → str
Patch / correct the documentation parameter notation to a supported format that works for linting. - name is the name of the function or method or Class

Parameters

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

Return type 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()

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

parse_method()

parse_exception()

parse_name(*line: Optional[str] = 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()

parse()

stubber.stubs_from_docs.**generate_from_rst**(*rst_path: pathlib.Path*, *dst_path: pathlib.Path*, *v_tag: str*, *release: Optional[str] = None*, *pattern: str = '*.rst'*, *verbose: bool = False*, *suffix='.py'*) → int

Parameters

- **rst_path** (*pathlib.Path*) –
- **dst_path** (*pathlib.Path*) –
- **v_tag** (*str*) –
- **release** (*Optional[str]*) –
- **pattern** (*str*) –
- **verbose** (*bool*) –

Return type int

stubber.update_fallback

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: Optional[str] = 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/createsubs.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.
main()	helper script

stubber.update_module_list.**read_modules**(*path: Optional[pathlib.Path] = 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.**main**()

helper script generate a few lines of code with all modules to be stubbed by createstubs

32.5.3 Package Contents

stubber.__version__

CHAPTER

THIRTYTHREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

S

INDEX

Symbols

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W