
Micropython-Stubber

Release 1.6.0

Jos Verlinde

Mar 14, 2022

CONTENTS:

1	Boost MicroPython productivity in VSCode	1
1.1	Licensing	2
2	Approach to collecting stub information	3
2.1	Stub collection process	3
2.2	Firmware Stubs format and limitations	4
2.3	Stub naming convention	4
3	Using stubs	5
3.1	Manual configuration	5
3.2	using micropython-stubber	5
3.3	Using micropy-cli	5
4	VSCode and Pylint configuration	7
4.1	Recommended order of the stubs in your config:	7
4.2	Relevant VSCode settings	7
4.3	pylint	9
4.4	Microsoft Python Language Server settings - Deprecated	9
5	Create Firmware Stubs	11
5.1	Running the script	11
5.2	Generating Stubs for a specific Firmware	12
5.3	Downloading the files	12
5.4	Custom firmware	12
5.5	The Unstubbables	13
6	createstub variants	15
6.1	board/createstubs.py	15
6.2	board/createstubs_mem.py	15
6.3	Optimisations	16
7	CPython and Frozen modules	17
7.1	Frozen Modules	17
7.2	Collect Frozen Stubs (micropython)	17
7.3	Postprocessing	18
8	Documentation Stubs	19
8.1	What / Why	19
8.2	How docstubs are generated	19
9	Repo structure	23

9.1	This and sister repos	23
9.2	Structure of this repo	24
9.3	Naming Convention and Stub folder structure	24
9.4	Create a symbolic link	25
10	PowerShell Scripts	27
10.1	bulk_stubber.ps1	27
11	Overview of Stubs	29
11.1	Firmware and libraries	29
11.2	Included custom stubs	29
12	References	31
12.1	Inspiration	31
12.2	Documentation on Type hints	32
13	Documentation	33
14	Changelog	35
14.1	Documentation Stubs	35
14.2	createstubs.py - v1.4.3	35
14.3	createstubs.py - v1.4.2	35
14.4	minified createstubs.py - v1.4.1	35
14.5	documentation	36
14.6	createstubs - v1.4-beta	36
14.7	createstubs.py - v1.3.16	36
15	TO-DO (provisional)	39
16	Upstream Documentation	41
16.1	ifconfig	41
16.2	ap.config	41
16.3	write_pulses	41
16.4	working on it	42
17	Developing	43
17.1	Cloning the repo	43
17.2	Windows 10	43
17.3	Github codespaces	44
17.4	Wrestling with two pythons	44
17.5	Minification	44
17.6	Testing	45
17.7	Debugging Cpython code that run Micropython	45
17.8	github actions	46
18	Testing	47
18.1	testing & debugging createstubs.py	47
18.2	platform detection	47
18.3	Code Coverage	48
19	API Reference	49
19.1	createstubs_mem	49
19.2	stub_lvgl	52
19.3	createstubs_db	52
19.4	createstubs	55
19.5	stubber	57

19.6	pyboard	89
19.7	visitors	92
19.8	commands	98
19.9	basics	102
19.10	ata_script	104
19.11	samples	105
19.12	simple	108
19.13	docstrings	109
20	Indices and tables	111
	Python Module Index	113
	Index	115

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](#).

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:

![stub processing order][./img/stuborder_pylance.png]

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 [pypcopy-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

USING STUBS

3.1 Manual configuration

The manual configuration, including sample configuration files is described in detail in the sister-repo [micropython-stubs][1] section [using-the-stubs][2]

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.

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	De-fault	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	De- fault	Description
python.analysis.stubPath	Typ- ings	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.autoSearchPath	True	Used to automatically add search paths based on some predefined names (like <code>src</code>).
python.analysis.extraPaths	False	Used to specify extra search paths for import resolution. This replaces the old <code>python.autoComplete.extraPaths</code> 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.

Set-ting	Default	Description	ref
<code>python.jediEnabled</code>	<code>True</code> , must be set to <code>FALSE</code>	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.	
<code>python.analysis.</code>			code analysis settings)
<code>type-shed-Paths</code>	<code>[]</code>	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

CREATE FIRMWARE STUBS

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

the script goes through 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-pyboard
 - 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-pyboard
 - /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 may 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 than it 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 frozen modules over the firmware stubs.

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	mini-fied/createstubs_mem.py	mini-fied/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:

- Minification, 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

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`

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 interpretation 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 **Exceptions** 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 they 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 wilddcard are added as comments only
```

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

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	to collect frozen modules	submodule of micropython-stubber	develop/micropython-stubber/micropython
micropython-lib	to collect frozen modules	submodule of micropython-stubber	develop/micropython-stubber/micropython-lib
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`
-

Note:

- For Git submodules please refer to <https://git-scm.com/book/en/v2/Git-Tools-Submodules>
-

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](#).

Note: I in the repo tests I have used the folders TESTREPO-micropython and TESTREPO-micropython-lib to avoid conflicts with any development that you might be doing on similar micropython repos at the potential cost of a little disk space.

```
cd /develop

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

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
```


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 Minification 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.

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>

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](#)

DOCUMENTATION

This documentation is built using [Sphinx](#) 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 Sphinx [AutoApi plugin](#)

CHANGELOG

14.1 Documentation Stubs

- avoid the use of `BaseException`

14.2 `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``

14.3 `createstubs.py` - v1.4.2

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

14.4 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

14.5 documentation

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

14.6 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

14.7 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

TO-DO (PROVISIONAL)

UPSTREAM DOCUMENTATION

in docstubs:

16.1 ifconfig

in order to accept `ifconfig()` without any parameters from `: configtuple: Optional[Any]` to `: configtuple: Optional[Tuple] = None`

16.2 ap.config

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

16.3 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:
```

16.4 working on it

16.4.1 documentation

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

16.4.2 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

16.4.3 frozen stubs

- add simple readme.md ?

16.4.4 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

16.4.5 Webrepl

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

17.1 Cloning the repo

The repo uses two submodules in order to generate the MicroPython frozen stubs. In order to fully check-out the repo you need to run additional commands: how to clone including submodules:

```
git clone https://github.com/Josverl/micropython-stubber.git
cd micropython-stubber
git submodule init
git submodule update --recursive
# install pyright
npm install
```

17.1.1 adding additional submodules

how to add :

```
git submodule add --force -b main https://github.com/micropython/micropython.git
git submodule add --force -b main https://github.com/micropython/micropython-lib.git
```

17.2 Windows 10

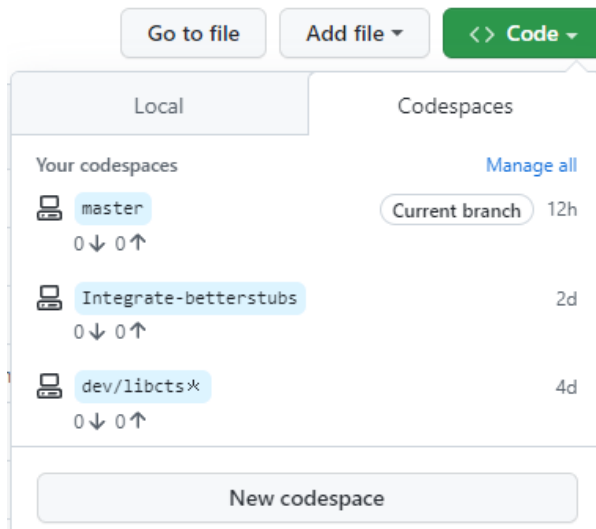
I use Windows 10/11 and use WSL2 to run the linux based parts. if you develop on other platform, it is quite likely that you may need to change some details. if that is needed , please update/add to the documentation and send a documentation PR.

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

17.3 Github codespaces

It 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.



17.4 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.

17.5 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.

17.6 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 module retrieval of frozen modules	does not use mocking but actually retrieves different firmware versions locally using git or downloads modules for online	local windows
common	all other tests	common	local + github action

Note: Also see [test documentation](#)

Platform detection to support pytest In order to allow both simple usability on 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.

17.7 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": {
```

(continues on next page)

(continued from previous page)

```
        "PYTEST_ADDOPTS": "--no-cov"  
    },  
    }
```

17.8 github actions

17.8.1 pytest.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

17.8.2 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

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

18.1 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 its 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)

18.2 platform detection

In order to allow both simple usability on 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 tests are platform dependent and have been marked to only run on linux or windows

18.3 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

API REFERENCE

This page contains auto-generated API reference documentation¹.

19.1 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 additional steps are recommended: - minification, using python-minifier

to reduce overall size, and remove logging overhead.

- cross compilation, using mpy-cross, to avoid the compilation step on the micropython device

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

19.1.1 Module Contents

Classes

<i>Stubber</i>	Generate stubs for modules in firmware
----------------	--

Functions

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

continues on next page

¹ Created with sphinx-autoapi

Table 2 – continued from previous page

`main()`

Attributes

`__version__`

`ENOENT`

`_MAX_CLASS_LEVEL`

`_log`

`createstubs_mem.__version__ = 1.5.5`

`createstubs_mem.ENOENT = 2`

`createstubs_mem._MAX_CLASS_LEVEL = 2`

`createstubs_mem.resetWDT()`

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(*self*, *item_instance*: *object*)

extract information of the objects members and attributes

Parameters `item_instance` (*object*) –

add_modules(*self*, *modules*: *list*)

Add additional modules to be exported

Parameters `modules` (*list*) –

create_all_stubs(*self*)

Create stubs for all configured modules

create_one_stub(*self*, *module_name*)

create_module_stub(*self*, *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(self, 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(self)

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

clean(self, path: *str* = None)

Remove all files from the stub folder

Parameters **path** (*str*) –

report(self, 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.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

19.2 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.

19.2.1 Module Contents

Functions

<code>main()</code>	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.

19.3 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) **creates a btree database of the files that should be stubbed**
 - todo:
 - add a main.py that starts stubbing
- 3) **process the modules in the database:**
 - stub the module
 - 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` to 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`

19.3.1 Module Contents

Classes

<i>Stubber</i>	Generate stubs for modules in firmware
----------------	--

Functions

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

Attributes

<i>__version__</i>
<i>ENOENT</i>
<i>_MAX_CLASS_LEVEL</i>
<i>_log</i>

```
createstubs_db.__version__ = 1.5.5
```

```
createstubs_db.ENOENT = 2
```

```
createstubs_db._MAX_CLASS_LEVEL = 2
```

```
createstubs_db.resetWDT()
```

```
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(self, item_instance: object)
```

extract information of the objects members and attributes

Parameters **item_instance** (object) –

add_modules(*self*, *modules*: *list*)

Add additional modules to be exported

Parameters *modules* (*list*) –

create_all_stubs(*self*)

Create stubs for all configured modules

create_one_stub(*self*, *module_name*)

create_module_stub(*self*, *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(*self*, *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(*self*)

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

clean(*self*, *path*: *str* = *None*)

Remove all files from the stub folder

Parameters *path* (*str*) –

report(*self*, *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.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`

19.4 createstubs

Create stubs for (all) modules on a MicroPython board

19.4.1 Module Contents

Classes

<code>Stubber</code>	Generate stubs for modules in firmware
----------------------	--

Functions

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

Attributes

<code>__version__</code>	
<code>ENOENT</code>	
<code>_MAX_CLASS_LEVEL</code>	
<code>_log</code>	

`createstubs.__version__ = 1.5.5`
`createstubs.ENOENT = 2`

`createstubs._MAX_CLASS_LEVEL = 2`

`createstubs.resetWDT()`

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(*self*, *item_instance*: *object*)

extract information of the objects members and attributes

Parameters *item_instance* (*object*) –

add_modules(*self*, *modules*: *list*)

Add additional modules to be exported

Parameters *modules* (*list*) –

create_all_stubs(*self*)

Create stubs for all configured modules

create_one_stub(*self*, *module_name*)

create_module_stub(*self*, *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(*self*, *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(*self*)

Turn *fwid* from 'v1.2.3' into '1_2_3' to be used in filename

clean(*self*, *path*: *str* = None)

Remove all files from the stub folder

Parameters *path* (*str*) –

report(*self*, *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.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`

19.5 stubber

19.5.1 Subpackages

`stubber.rst`

Submodules

`stubber.rst.classsort`

Sort list of classes 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

<code>sort_classes(classes: List[str])</code>	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.RST_DOC_FIXES = [['.. method:: match.', '.. method:: Match.'], ['match.end', ' ...']
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'], ['(adcx, adcy, ...)', '(bufx,...']
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

<i>SourceDict</i>	(abstract) dict to store source components respecting parent child dependencies and proper definition order
<i>ModuleSourceDict</i>	(abstract) dict to store source components respecting parent child dependencies and proper definition order
<i>ClassSourceDict</i>	(abstract) dict to store source components respecting parent child dependencies and proper definition order
<i>FunctionSourceDict</i>	(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__(self) → str
convert the OD into a string

Return type str

__add__(self, dict: SourceDict)

Parameters dict (SourceDict) –

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

Parameters

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

add_comment(self, line: Union[str, List[str]])
Add a comment, or list of comments, to this block.

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

add_constant(*self*, *line*: str, *autoindent*: bool = True)
 add constant to the constant scope of this block

Parameters

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

add_constant_smart(*self*, *name*: str, *type*: str = 'Any', *docstr*: List[str] = [], *autoindent*: bool = True)
 add 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(*self*, *name*: str) → Union[str, None]

Parameters **name** (str) –

Return type Union[str, None]

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

Parameters

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

index(*self*, *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(*self*)
 make sure all classdefs are in order

__str__(*self*)
 convert the OD into a string

find(*self*, *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(*self*)
get a list of the class names in parent-child order

add_import(*self*, 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: pathlib.Path, pattern: str)

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

<i>simple_candidates</i> (type: str, match_string: str, key-words: List[str], rate: float = 0.5, exclude: List[str] = [])	find and rate possible types and confidence weighting for simple types.
---	---

continues on next page

Table 14 – continued from previous page

<code>compound_candidates</code> (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.
<code>object_candidates</code> (match_string: str, rate: float = 0.81, exclude: List[str] = ['IRQ'])	find and rate possible types and confidence weighting for Object types.
<code>distill_return</code> (return_text: str) → List[Dict]	Find return type and confidence.
<code>return_type_from_context</code> (docstring: Union[str, List[str]], signature: str, module: str, literal: bool = False)	
<code>_type_from_context</code> (docstring: Union[str, List[str]], signature: str, module: str, literal: bool = False)	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

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

Functions

<i>sort_classes</i> (classes: List[str])	sort a list of classes to respect the parent-child order
<i>simple_candidates</i> (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.
<i>compound_candidates</i> (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.
<i>object_candidates</i> (match_string: str, rate: float = 0.81, exclude: List[str] = ['IRQ'])	find and rate possible types and confidence weighting for Object types.
<i>distill_return</i> (return_text: str) → List[Dict]	Find return type and confidence.
<i>return_type_from_context</i> (docstring: Union[str, List[str]], signature: str, module: str, literal: bool = False)	
<i>_type_from_context</i> (docstring: Union[str, List[str]], signature: str, module: str, literal: bool = False)	Determine the return type of a function or method based on:

Attributes

<i>LOOKUP_LIST</i>
<i>NONE_VERBS</i>
<i>CHILD_PARENT_CLASS</i>
<i>PARAM_FIXES</i>
<i>MODULE_GLUE</i>

continues on next page

Table 18 – continued from previous page

<i>RST_DOC_FIXES</i>
<i>TYPING_IMPORT</i>
<i>__all__</i>
<p><code>stubber.rst.sort_classes(classes: List[str])</code> sort a list of classes to respect the parent-child order</p> <p>Parameters <code>classes</code> (List[str]) –</p> <p><code>stubber.rst.LOOKUP_LIST</code></p> <p><code>stubber.rst.NONE_VERBS = ['Activate ', 'Build a ', 'Cancel ', 'Clear ', 'Close ', 'cancel ', 'Configure ', 'Connect ', ...]</code></p> <p><code>stubber.rst.CHILD_PARENT_CLASS</code></p> <p><code>stubber.rst.PARAM_FIXES = [['*', '*'], ['**', '**'], ['/*', '*'], ['"param"', 'param'], ['(adcx, adcy, ...), (bufx, ...]</code></p> <p><code>stubber.rst.MODULE_GLUE</code></p> <p><code>stubber.rst.RST_DOC_FIXES = ['.. method:: match.', '.. method:: Match.'], ['match.end', ' ...]</code></p> <p>class <code>stubber.rst.SourceDict</code>(base: List, indent: int = 0, body: int = 0, lf: str = '\n') Bases: OrderedDict</p> <p>(abstract) dict to store source components respecting parent child dependencies and proper definition order</p> <p>Parameters</p> <ul style="list-style-type: none"> • base (List) – • indent (int) – • body (int) – • lf (str) – <p><code>__str__(self) → str</code> convert the OD into a string</p> <p>Return type str</p> <p><code>__add__(self, dict: SourceDict)</code></p> <p>Parameters <code>dict</code> (SourceDict) –</p> <p><code>add_docstr(self, docstr: Union[str, List[str]], extra: int = 0)</code></p> <p>Parameters</p> <ul style="list-style-type: none"> • docstr (Union[str, List[str]]) – • extra (int) – <p><code>add_comment(self, line: Union[str, List[str]])</code> Add a comment, or list of comments, to this block.</p> <p>Parameters <code>line</code> (Union[str, List[str]]) –</p>

add_constant(self, line: *str*, autoindent: *bool* = True)
add constant to the constant scope of this block

Parameters

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

add_constant_smart(self, name: *str*, type: *str* = 'Any', docstr: List[*str*] = [], autoindent: *bool* = True)
add 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(self, name: *str*) → Union[*str*, None]

Parameters **name** (*str*) –

Return type Union[*str*, None]

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

Parameters

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

index(self, 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(self)
make sure all classdefs are in order

__str__(self)
convert the OD into a string

find(self, 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(*self*)
get a list of the class names in parent-child order

add_import(*self*, *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__
```

19.5.2 Submodules

stubber.basicgit

simple Git module, where needed via powershell

Module Contents

Functions

<code>_run_git(cmd: List[str], repo: Optional[str] = None, expect_stderr=False, capture_output=True)</code>	run a external (git) command in the repo's folder and deal with some of the errors
<code>clone(remote_repo: str, path: pathlib.Path, shallow=False) → bool</code>	git clone --depth 0 <remote> <directory>
<code>get_tag(repo: Optional[str] = None, abbreviate: bool = True) → Union[str, None]</code>	get the most recent git version tag of a local repo
<code>checkout_tag(tag: str, repo: Optional[str] = None) → bool</code>	checkout a specific git tag
<code>checkout_commit(commit_hash: str, repo: Optional[str] = None) → bool</code>	Checkout a specific commit
<code>switch_tag(tag: str, repo: Optional[str] = None) → bool</code>	get the most recent git version tag of a local repo"
<code>switch_branch(branch: str, repo: Optional[str] = None) → bool</code>	get the most recent git version tag of a local repo"
<code>fetch(repo: str) → bool</code>	fetches a repo
<code>pull(repo: str, branch='main') → bool</code>	pull a repo origin into main

```
stubber.basicgit._run_git(cmd: List[str], repo: Optional[str] = None, expect_stderr=False, capture_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[str]*) –

`stubber.basicgit.clone(remote_repo: str, path: pathlib.Path, shallow=False) → bool`
 git clone -depth 0 <remote> <directory>

Parameters

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

Return type *bool*

`stubber.basicgit.get_tag(repo: Optional[str] = None, abbreviate: bool = True) → Union[str, None]`
 get the most recent git version tag of a local repo repo should be in the form of : repo = “./micropython”
 returns the tag or None

Parameters

- **repo** (*Optional[str]*) –
- **abbreviate** (*bool*) –

Return type *Union[str, None]*

`stubber.basicgit.checkout_tag(tag: str, repo: Optional[str] = None) → bool`
 checkout a specific git tag

Parameters

- **tag** (*str*) –
- **repo** (*Optional[str]*) –

Return type *bool*

`stubber.basicgit.checkout_commit(commit_hash: str, repo: Optional[str] = None) → bool`
 Checkout a specific commit

Parameters

- **commit_hash** (*str*) –
- **repo** (*Optional[str]*) –

Return type *bool*

`stubber.basicgit.switch_tag(tag: str, repo: Optional[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[str]*) –

Return type *bool*

`stubber.basicgit.switch_branch(branch: str, repo: Optional[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[str]*) –

Return type *bool*

`stubber.basicgit.fetch(repo: str) → bool`

fetches a repo repo should be in the form of : path/.git repo = ‘../micropython/.git’ returns True on success

Parameters `repo (str)` –

Return type `bool`

`stubber.basicgit.pull(repo: 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 (str)` –

Return type `bool`

`stubber.downloader`

Download files from a public github repo

Module Contents

Functions

<code>download_file(url: str, module: str, folder: str = '/')</code>	download a file from a public github repo
<code>download_files(repo, frozen_modules, savepath)</code>	download multiple files from a public github repo

Attributes

<code>log</code>

`stubber.downloader.log`

`stubber.downloader.download_file(url: str, module: str, folder: str = '/')`

download a file from a public github repo

Parameters

- `url (str)` –
- `module (str)` –
- `folder (str)` –

`stubber.downloader.download_files(repo, frozen_modules, savepath)`

download multiple files from a public github repo

stubber.get_cpython

Download or update the micropython 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=None, family: str = 'core') Download MicroPython compatibility modules

Attributes

log

stubber.get_cpython.log

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=None, *, repo=None, version='3.2.24') Loboris frozen modules

Attributes

FAMILY

PORT

log

```

stubber.get_lobo.FAMILY = loboris
stubber.get_lobo.PORT = esp32_lobo
stubber.get_lobo.log
stubber.get_lobo.get_frozen(stub_path=None, *, repo=None, version='3.2.24')
    Loboris frozen modules

```

stubber.get_mpy

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

<code>get_frozen(stub_path: str, version: str, mpy_path: str = None, lib_path: str = None)</code>	get and parse the to-be-frozen .py modules for micropython to extract the static type information
<code>get_frozen_folders(stub_path: str, mpy_path: str, lib_path: str, version: str)</code>	get and parse the to-be-frozen .py modules for micropython to extract the static type information
<code>get_target_names(path: str) → tuple</code>	get path to port and board names from a path
<code>read_micropython_lib_commits(filename='data/micropython_lib_commits')</code>	Return a list with the micropython version and matching micropython-lib commit-hashes
<code>get_frozen_from_manifest(manifests, stub_path: str, mpy_path: str, lib_path: str, version: str)</code>	get and parse the to-be-frozen .py modules for micropython to extract the static type information

Attributes

<code>log</code>
<code>FAMILY</code>
<code>mpy_path</code>

```

stubber.get_mpy.log
stubber.get_mpy.FAMILY = micropython
stubber.get_mpy.get_frozen(stub_path: str, version: str, mpy_path: str = None, lib_path: str = 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

Parameters

- `stub_path` (*str*) –

- **version** (*str*) –
- **mpy_path** (*str*) –
- **lib_path** (*str*) –

`stubber.get_mpy.get_frozen_folders(stub_path: str, mpy_path: str, lib_path: 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_path** (*str*) –
- **mpy_path** (*str*) –
- **lib_path** (*str*) –
- **version** (*str*) –

`stubber.get_mpy.get_target_names(path: str) → tuple`

get path to port and board names from a path

Parameters *path* (*str*) –

Return type *tuple*

`stubber.get_mpy.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.

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

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

`stubber.get_mpy.get_frozen_from_manifest(manifests, stub_path: str, mpy_path: str, lib_path: 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 through the manifest.py introduced in MicroPython 1.12 - manifest.py is used for board specific and daily builds - manifest_release.py is used for the release builds

Parameters

- **stub_path** (*str*) –
- **mpy_path** (*str*) –
- **lib_path** (*str*) –
- **version** (*str*) –

`stubber.get_mpy.mpy_path = ./micropython`

`stubber.makemanifest_2`

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=None, opt=0)

freeze_as_str(path)

<i>freeze_mpy</i> (path, script=None, opt=0)	Freeze the input (see above), which must be .mpy files that are
--	---

<i>freeze</i> (path, script=None, opt=0)	Freeze the input, automatically determining its type. A .py script
--	--

<i>include</i> (manifest, **kwargs)	Include another manifest.
-------------------------------------	---------------------------

<i>convert_path</i> (path: str) → str	Perform variable substitution in path
---------------------------------------	---------------------------------------

<i>freeze_internal</i> (path: str, script: str, opt=None)	Copy the to-be-frozen module to the destination folder to be stubbed.
---	---

Attributes

log

path_vars

stub_dir

stubber.makemaniifest_2.log

stubber.makemaniifest_2.path_vars

exception stubber.makemaniifest_2.FreezeError

Bases: `Exception`

Common base class for all non-exit exceptions.

class stubber.makemaniifest_2.IncludeOptions(**kwargs)

defaults(self, **kwargs)

__getattr__(self, name)

stubber.makemaniifest_2.freeze_as_mpy(path, script=None, opt=0)

stubber.makemaniifest_2.freeze_as_str(path)

stubber.makemaniifest_2.freeze_mpy(path, script=None, opt=0)

Freeze the input (see above), which must be .mpy files that are frozen directly.

`stubber.makemani fest_2.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.makemani fest_2.include(manifest, **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.`

`stubber.makemani fest_2.convert_path(path: str) → str`

Perform variable substitution in path

Parameters `path (str)` –

Return type `str`

`stubber.makemani fest_2.stub_dir :str =`

`stubber.makemani fest_2.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.minify

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

Module Contents

Functions

<code>edit_lines</code> (content, edits, diff=False)	Edit string by list of edits
<code>minify_script</code> (source_script: pathlib.Path, keep_report=True, diff=False) → str	minifies createstubs.py
<code>minify</code> (source: Union[str, pathlib.Path], target: Union[str, pathlib.Path], keep_report: bool = True, diff: bool = False, cross_compile: bool = False)	

Attributes

<code>python_minifier</code>

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 represents the 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

Pre/Post Processing for createstubs.py

Module Contents

Functions

<i>do_post_processing</i> (stub_paths: List[pathlib.Path], pyi: bool, black: bool)	Common post processing
<i>stubber_cli</i> (ctx, debug=False)	
<i>cli_init</i> (mpy: bool, mpy_lib: bool, path: Union[str, pathlib.Path])	Clone the micropython repos locally to be able to generate frozen-stubs and doc-stubs.
<i>cli_stub</i> (source: Union[str, pathlib.Path])	Create or update .pyi type hint files for all .py files in SOURCE path.
<i>cli_minify</i> (ctx, source: Union[str, pathlib.Path], target: Union[str, pathlib.Path], keep_report: bool, diff: bool, cross_compile: bool, all: bool) → int	Creates a minified version of the SOURCE micropython file in TARGET (file or folder).
<i>cli_get_frozen</i> (stub_folder: str = STUB_FOLDER, mpy_folder: str = MPY_FOLDER, mpy_lib_folder: str = MPY_LIB_FOLDER, version: str = "", pyi: bool = True, black: bool = True)	Get the frozen modules for the checked out version of MicroPython
<i>cli_get_lobo</i> (stub_folder: str = STUB_FOLDER, pyi: bool = True, black: bool = True)	Get the frozen modules for the Loboris v3.2.24 fork of MicroPython
<i>cli_get_core</i> (stub_folder: str = STUB_FOLDER, pyi: bool = True, black: bool = True)	Get the core (CPython compat) modules for both MicroPython and Pycopy.
<i>cli_docstubs</i> (source: str = "", target: str = './all-stubs', verbose: bool = False, black: bool = True, base-name='micropython')	Read the Micropython library documentation files and use them to build stubs that can be used for static type-checking.

Attributes

`STUB_FOLDER`

`MPY_FOLDER`

`MPY_LIB_FOLDER`

`log`

`stubber.stubber.STUB_FOLDER = ./all-stubs`

`stubber.stubber.MPY_FOLDER = ./micropython`

`stubber.stubber.MPY_LIB_FOLDER = ./micropython-lib`

`stubber.stubber.log`

`stubber.stubber.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.stubber.stubber_cli(ctx, debug=False)`

`stubber.stubber.cli_init(mpy: bool, mpy_lib: bool, path: Union[str, pathlib.Path])`
Clone the micropython repos locally to be able to generate frozen-stubs and doc-stubs.

Parameters

- `mpy` (`bool`) –
- `mpy_lib` (`bool`) –
- `path` (`Union[str, pathlib.Path]`) –

`stubber.stubber.cli_stub(source: Union[str, pathlib.Path])`

Create or update .pyi type hint files for all .py files in SOURCE path.

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

`stubber.stubber.cli_minify(ctx, source: Union[str, pathlib.Path], target: Union[str, pathlib.Path], keep_report: bool, diff: bool, cross_compile: bool, all: bool) → int`
Creates a minified version of the SOURCE micropython file in TARGET (file or folder).

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.stubber.cli_get_frozen(stub_folder: str = STUB_FOLDER, mpy_folder: str = MPY_FOLDER, mpy_lib_folder: str = MPY_LIB_FOLDER, version: str = "", pyi: bool = True, black: bool = True)`

Get the frozen modules for the checked out version of MicroPython

Parameters

- **stub_folder** (*str*) –
- **mpy_folder** (*str*) –
- **mpy_lib_folder** (*str*) –
- **version** (*str*) –
- **pyi** (*bool*) –
- **black** (*bool*) –

`stubber.stubber.cli_get_lobo(stub_folder: str = STUB_FOLDER, pyi: bool = True, black: bool = True)`

Get the frozen modules for the Loberis v3.2.24 fork of MicroPython

Parameters

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

`stubber.stubber.cli_get_core(stub_folder: str = STUB_FOLDER, pyi: bool = True, black: bool = True)`

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

Parameters

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

`stubber.stubber.cli_docstubs(source: str = "", target: str = './all-stubs', verbose: bool = False, black: bool = True, basename='micropython')`

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

Parameters

- **source** (*str*) –
- **target** (*str*) –
- **verbose** (*bool*) –
- **black** (*bool*) –

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

_color(c, *args) → str

_green(*args) → str

_red(*args) → str

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

Attributes

log

NEW_OUTPUT

SEPERATOR

stubber.stubs_from_docs.log

stubber.stubs_from_docs.NEW_OUTPUT = True

stubber.stubs_from_docs.SEPERATOR = ::

stubber.stubs_from_docs._color(c, *args) → str

Return type str

stubber.stubs_from_docs._green(*args) → str

Return type str

stubber.stubs_from_docs._red(*args) → str

Return type str

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

verbose = False

gather_docs = False

target = .py

property line(self) → 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

property module_names(self) → List[str]

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

Return type List[str]

strip_prefixes(self, 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(self)

read_file(self, filename: pathlib.Path)

Parameters filename (pathlib.Path) –

prepare_output(self)

clean up some trailing spaces and commas

write_file(self, filename: pathlib.Path) → bool

Parameters filename (pathlib.Path) –

Return type bool

at_anchor(self) → bool

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

Return type `bool`

at_heading(*self*) → `bool`
stop at heading

Return type `bool`

parse_docstring(*self*) → `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(*self*, *params*: `str`)

change documentation parameter notation to a supported format that works for linting

Parameters *params* (`str`) –

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

Parameters

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

parse_toc(*self*)

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

parse_module(*self*)

parse a module tag and set the module's docstring

parse_current_module(*self*)

parse_function(*self*)

parse_class(*self*)

get_rst_hint(*self*)

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

parse_method(*self*)

parse_exception(*self*)

parse_name(*self*, *line*: `Optional[str] = None`)

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

Parameters *line* (`Optional[str]`) –

parse_names(*self*, *oneline*: `bool = True`)

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

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

Parameters *oneline* (`bool`) –

parse_data(*self*)

parse(*self*)

`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_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

<code>read_modules</code> (path: <i>pathlib.Path</i> = None) → Set	read text files with modules per firmware.
<code>wrapped</code> (modules: Union[Set, List]) → str	wrap code line at spaces
<code>main</code> ()	helper script

`stubber.update_module_list.read_modules`(path: *pathlib.Path* = None) → Set
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 separator, - and add each module to a set

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

Return type Set

`stubber.update_module_list.wrapped`(modules: Union[Set, List]) → str
wrap code line at spaces

Parameters **modules** (Union[Set, List]) –

Return type str

`stubber.update_module_list.main`()
helper script generate a few lines of code with all modules to be stubbed by createstubs

stubber.utils

Module Contents

Functions

<code>clean_version</code> (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
<code>stubfolder</code> (path: str) → str	return path in the stub folder
<code>cleanup</code> (modules_folder: pathlib.Path, all_pyi: bool = False)	Q&D cleanup
<code>generate_pyi_from_file</code> (file: pathlib.Path) → bool	Generate a .pyi stubfile from a single .py module using mypy/stubgen
<code>fix_umqtt_init</code> (modules_path: pathlib.Path)	
<code>generate_pyi_files</code> (modules_folder: pathlib.Path) → bool	generate typeshed files for all scripts in a folder using mypy/stubgen
<code>manifest</code> (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
<code>make_manifest</code> (folder: pathlib.Path, family: str, port: str, version: str, release: str = "", stubtype: str = "", board: str = "") → bool	Create a <i>module.json</i> manifest listing all files/stubs in this folder and subfolders.
<code>read_exclusion_file</code> (path: pathlib.Path = None) → List[str]	Read a .exclusion file to determine which files should not be automatically re-generated
<code>should_ignore</code> (file: str, exclusions: List[str]) → bool	Check if a file matches a line in the exclusion list.

Attributes

<code>log</code>
<code>STUB_FOLDER</code>
<code>STUBGEN_OPT</code>

stubber.utils.log

stubber.utils.STUB_FOLDER = ./all-stubs

stubber.utils.STUBGEN_OPT

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*) –

`stubber.utils.stubfolder(path: str) → str`
 return path in the stub folder

Parameters `path` (*str*) –

Return type *str*

`stubber.utils.cleanup(modules_folder: pathlib.Path, all_pyi: bool = False)`
 Q&D cleanup

Parameters

- **modules_folder** (*pathlib.Path*) –
- **all_pyi** (*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.fix_umqtt_init(modules_path: pathlib.Path)`

Parameters `modules_path` (*pathlib.Path*) –

`stubber.utils.generate_pyi_files(modules_folder: pathlib.Path) → bool`
 generate typeshed files for all scripts in a folder using mypy/stubgen

Parameters `modules_folder` (*pathlib.Path*) –

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.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.read_exclusion_file(path: pathlib.Path = None) → List[str]`

Read a *.exclusion* file to determine which files should not be automatically re-generated in *.GitIgnore* format

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

Return type *List*[*str*]

`stubber.utils.should_ignore(file: str, exclusions: List[str]) → bool`

Check if a file matches a line in the exclusion list.

Parameters

- **file** (*str*) –
- **exclusions** (*List*[*str*]) –

Return type *bool*

19.6 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
```

19.6.1 Module Contents

Classes

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

Functions

<i>stdout_write_bytes(b)</i>
<i>execfile(filename, device='/dev/ttyACM0', baudrate=115200, user='micro', password='python')</i>
<i>filesystem_command(pyb, args)</i>
<i>main()</i>

Attributes

<i>stdout</i>
<i>_injected_import_hook_code</i>

`pyboard.stdout`

`pyboard.stdout_write_bytes(b)`

exception `pyboard.PyboardError`

Bases: `Exception`

Common base class for all non-exit exceptions.

```

class pyboard.TelnetToSerial(ip, user, password, read_timeout=None)
    __del__(self)
    close(self)
    read(self, size=1)
    write(self, data)
    inWaiting(self)

class pyboard.ProcessToSerial(cmd)
    Execute a process and emulate serial connection using its stdin/stdout.
    close(self)
    read(self, size=1)
    write(self, data)
    inWaiting(self)

class 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(self)
    read(self, size=1)
    write(self, data)
    inWaiting(self)

class pyboard.Pyboard(device, baudrate=115200, user='micro', password='python', wait=0, exclusive=True)
    close(self)
    read_until(self, min_num_bytes, ending, timeout=10, data_consumer=None)
    enter_raw_repl(self, soft_reset=True)
    exit_raw_repl(self)
    follow(self, timeout, data_consumer=None)
    raw_paste_write(self, command_bytes)
    exec_raw_no_follow(self, command)
    exec_raw(self, command, timeout=10, data_consumer=None)
    eval(self, expression)
    exec_(self, command, data_consumer=None)
    execfile(self, filename)
    get_time(self)
    fs_ls(self, src)
    fs_cat(self, src, chunk_size=256)
    fs_get(self, src, dest, chunk_size=256)
    fs_put(self, src, dest, chunk_size=256)
    fs_mkdir(self, dir)

```

`fs_rmdir(self, dir)`

`fs_rm(self, src)`

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

`pyboard.filesystem_command(pyb, args)`

`pyboard._injected_import_hook_code = Multiline-String`

```

1 import uos, uio
2 class _FS:
3     class File(uio.IOBase):
4         def __init__(self):
5             self.off = 0
6         def ioctl(self, request, arg):
7             return 0
8         def readinto(self, buf):
9             buf[:] = memoryview(_injected_buf)[self.off:self.off + len(buf)]
10            self.off += len(buf)
11            return len(buf)
12    mount = umount = chdir = lambda *args: None
13    def stat(self, path):
14        if path == '_injected.mpy':
15            return tuple(0 for _ in range(10))
16        else:
17            raise OSError(-2) # ENOENT
18    def open(self, path, mode):
19        return self.File()
20    uos.mount(_FS(), '/')
21    uos.chdir('/')
22    from _injected import *
23    uos.umount('/')
24    del _injected_buf, _FS

```

`pyboard.main()`

19.7 visitors

19.7.1 Submodules

`visitors._apply_stubber_annotations`

Module Contents

Classes

FunctionAnnotation

TypeCollector

Collect type annotations from a stub module.

continues on next page

Table 44 – continued from previous page

Annotations

ApplyStubberAnnotationsVisitor

Apply type annotations to a source module using the given stub modules.

Functions

_get_import_alias_names(import_aliases: Sequence[libcst.ImportAlias]) → Set[str]

_get_import_names(imports: Sequence[Union[libcst.Import, libcst.ImportFrom]]) → Set[str]

visitors._apply_stubber_annotations._**get_import_alias_names**(import_aliases: Sequence[libcst.ImportAlias]) → Set[str]

Parameters **import_aliases** (Sequence[libcst.ImportAlias]) –

Return type Set[str]

visitors._apply_stubber_annotations._**get_import_names**(imports: Sequence[Union[libcst.Import, libcst.ImportFrom]]) → Set[str]

Parameters **imports** (Sequence[Union[libcst.Import, libcst.ImportFrom]]) –

Return type Set[str]

class visitors._apply_stubber_annotations.**FunctionAnnotation**

parameters :libcst.Parameters

returns :Optional[libcst.Annotation]

class visitors._apply_stubber_annotations.**TypeCollector**(existing_imports: Set[str], context: libcst.codemod._context.CodemodContext)

Bases: libcst.CSTVisitor

Collect type annotations from a stub module.

Parameters

- **existing_imports** (Set[str]) –
- **context** (libcst.codemod._context.CodemodContext) –

visit_ClassDef(self, node: libcst.ClassDef) → None

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

Return type None

leave_ClassDef(self, original_node: libcst.ClassDef) → None

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

Return type None

```

visit_FunctionDef(self, node: libcst.FunctionDef) → bool

    Parameters node (libcst.FunctionDef) –
    Return type bool

leave_FunctionDef(self, original_node: libcst.FunctionDef) → None

    Parameters original_node (libcst.FunctionDef) –
    Return type None

visit_AnnAssign(self, node: libcst.AnnAssign) → bool

    Parameters node (libcst.AnnAssign) –
    Return type bool

leave_AnnAssign(self, original_node: libcst.AnnAssign) → None

    Parameters original_node (libcst.AnnAssign) –
    Return type None

visit_ImportFrom(self, node: libcst.ImportFrom) → None

    Parameters node (libcst.ImportFrom) –
    Return type None

_add_annotation_to_imports(self, annotation: libcst.Attribute) → Union[libcst.Name, libcst.Attribute]

    Parameters annotation (libcst.Attribute) –
    Return type Union[libcst.Name, libcst.Attribute]

_handle_Index(self, slice: libcst.Index, node: libcst.Subscript) → libcst.Subscript

    Parameters
        • slice (libcst.Index) –
        • node (libcst.Subscript) –
    Return type libcst.Subscript

_handle_Subscript(self, node: libcst.Subscript) → libcst.Subscript

    Parameters node (libcst.Subscript) –
    Return type libcst.Subscript

_create_import_from_annotation(self, returns: libcst.Annotation) → libcst.Annotation

    Parameters returns (libcst.Annotation) –
    Return type libcst.Annotation

_import_parameter_annotations(self, parameters: libcst.Parameters) → libcst.Parameters

```

Parameters `parameters` (`libcst.Parameters`) –

Return type `libcst.Parameters`

```
class visitors._apply_stubber_annotations.Annotations
    function_annotations :Dict[str, FunctionAnnotation]
    attribute_annotations :Dict[str, libcst.Annotation]
    class_definitions :Dict[str, libcst.ClassDef]
class visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor(context:
    libcst.codemod._context.CodemodContext,
    annotations: Optional[Annotations] = None,
    overwrite_existing_annotations: bool = False)
```

Bases: `libcst.codemod._visitor.ContextAwareTransformer`

Apply type annotations to a source module using the given stub modules. You can also pass in explicit annotations for functions and attributes and pass in new class definitions that need to be added to the source module.

This is one of the transforms that is available automatically to you when running a codemod. To use it in this manner, import `ApplyStubberAnnotationsVisitor` and then call the static `store_stub_in_context()` method, giving it the current context (found as `self.context` for all subclasses of `Codemod`), the stub module from which you wish to add annotations.

For example, you can store the type annotation `int` for `x` using:

```
stub_module = parse_module("x: int = ...")
ApplyStubberAnnotationsVisitor.store_stub_in_context(self.context, stub_module)
```

You can apply the type annotation using:

```
source_module = parse_module("x = 1")
ApplyStubberAnnotationsVisitor.transform_module(source_module)
```

This will produce the following code:

```
x: int = 1
```

If the function or attribute already has a type annotation, it will not be overwritten.

To overwrite existing annotations when applying annotations from a stub, use the keyword argument `overwrite_existing_annotations=True` when constructing the codemod or when calling `store_stub_in_context`.

Parameters

- **context** (`libcst.codemod._context.CodemodContext`) –
- **annotations** (`Optional[Annotations]`) –
- **overwrite_existing_annotations** (`bool`) –

CONTEXT_KEY = `ApplyStubberAnnotationsVisitor`

static store_stub_in_context(*context: libcst.codemod._context.CodemodContext, stub: libcst.Module, overwrite_existing_annotations: bool = False*) → *None*

Store a stub module in the CodemodContext so that type annotations from the stub can be applied in a later invocation of this class.

If the `overwrite_existing_annotations` flag is `True`, the codemod will overwrite any existing annotations.

If you call this function multiple times, only the last values of `stub` and `overwrite_existing_annotations` will take effect.

Parameters

- **context** (*libcst.codemod._context.CodemodContext*) –
- **stub** (*libcst.Module*) –
- **overwrite_existing_annotations** (*bool*) –

Return type *None*

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

Collect type annotations from all stubs and apply them to `tree`.

Gather existing imports from `tree` so that we don't add duplicate imports.

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

Return type *libcst.Module*

_qualifier_name(*self*) → *str*

Return type *str*

_annotate_single_target(*self, node: libcst.Assign, updated_node: libcst.Assign*) → *Union[libcst.Assign, libcst.AnnAssign]*

Parameters

- **node** (*libcst.Assign*) –
- **updated_node** (*libcst.Assign*) –

Return type *Union[libcst.Assign, libcst.AnnAssign]*

_split_module(*self, module: libcst.Module, updated_module: libcst.Module*) → *Tuple[List[Union[libcst.SimpleStatementLine, libcst.BaseCompoundStatement]], List[Union[libcst.SimpleStatementLine, libcst.BaseCompoundStatement]]]*

Parameters

- **module** (*libcst.Module*) –
- **updated_module** (*libcst.Module*) –

Return type *Tuple[List[Union[libcst.SimpleStatementLine, libcst.BaseCompoundStatement]], List[Union[libcst.SimpleStatementLine, libcst.BaseCompoundStatement]]]*

_add_to_toplevel_annotations(*self, name: str*) → *None*

Parameters **name** (*str*) –

Return type *None*

_update_parameters(*self*, *annotations*: [FunctionAnnotation](#), *updated_node*: *libcst.FunctionDef*) → *libcst.Parameters*

Parameters

- **annotations** ([FunctionAnnotation](#)) –
- **updated_node** (*libcst.FunctionDef*) –

Return type *libcst.Parameters*

_insert_empty_line(*self*, *statements*: *List[Union[libcst.SimpleStatementLine, libcst.BaseCompoundStatement]]*) → *List[Union[libcst.SimpleStatementLine, libcst.BaseCompoundStatement]]*

Parameters **statements** (*List[Union[libcst.SimpleStatementLine, libcst.BaseCompoundStatement]]*) –

Return type *List[Union[libcst.SimpleStatementLine, libcst.BaseCompoundStatement]]*

visit_ClassDef(*self*, *node*: *libcst.ClassDef*) → [None](#)

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

Return type [None](#)

leave_ClassDef(*self*, *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(*self*, *node*: *libcst.FunctionDef*) → [bool](#)

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

Return type [bool](#)

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

Parameters

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

Return type *libcst.FunctionDef*

leave_Assign(*self*, *original_node*: *libcst.Assign*, *updated_node*: *libcst.Assign*) → *Union[libcst.Assign, libcst.AnnAssign]*

Parameters

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

Return type Union[libcst.Assign, libcst.AnnAssign]

leave_ImportFrom(self, original_node: libcst.ImportFrom, updated_node: libcst.ImportFrom) → libcst.ImportFrom

Parameters

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

Return type libcst.ImportFrom

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

Parameters

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

Return type libcst.Module

19.8 commands

19.8.1 Submodules

commands.constant_folding

Module Contents

Classes

ConvertConstantCommand

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

class commands.constant_folding.**ConvertConstantCommand**(context: libcst.codemod.CodemodContext, string: str, constant: 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) –
- **string** (str) –
- **constant** (str) –

DESCRIPTION :str = Converts raw strings to constant accesses.

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`

leave_SimpleString(*self*, *original_node*: *`libcst.SimpleString`*, *updated_node*: *`libcst.SimpleString`*) → *`Union[libcst.SimpleString, libcst.Name]`*

Parameters

- **original_node** (*`libcst.SimpleString`*) –
- **updated_node** (*`libcst.SimpleString`*) –

Return type *`Union[libcst.SimpleString, libcst.Name]`*

`commands.noop`

Module Contents

Classes

`NOOPCommand`

A Codemod which can be invoked on the command-line

class `commands.noop.NOOPCommand`(*context*: *`libcst.codemod._context.CodemodContext`*)

Bases: *`libcst.codemod.CodemodCommand`*

A Codemod which can be invoked on the command-line using the `libcst.tool codemod` utility. It behaves like any other codemod in that it can be instantiated and run identically to a `Codemod`. However, it provides support for providing help text and command-line arguments to `libcst.tool codemod` as well as facilities for automatically running certain common transforms after executing your `transform_module_impl()`.

The following list of transforms are automatically run at this time:

- `AddImportsVisitor` (adds needed imports to a module).
- `RemoveImportsVisitor` (removes unreferenced imports from a module).

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

DESCRIPTION :str = Does absolutely nothing.

transform_module_impl(*self*, *tree*: *`libcst.Module`*) → *`libcst.Module`*

Override this with your transform. You should take in the tree, optionally mutate it and then return the mutated version. The module reference and all calculated metadata are available for the lifetime of this function.

Parameters `tree` (*`libcst.Module`*) –

Return type *`libcst.Module`*

commands.samples

Module Contents

commands.samples.simple_stub = Multiline-String

```

1  """
2  Module: 'machine' on micropython-esp32-1.15
3  """
4  # MCU: {'ver': '1.15', 'port': 'esp32', 'arch': 'xtensawin', 'sysname':
   ↳ 'esp32', 'release': '1.15.0', 'name': 'micropython', 'mpy': 10757,
   ↳ 'version': '1.15.0', 'machine': 'ESP32 module (spiram) with ESP32', 'build
   ↳ ': '', 'nodename': 'esp32', 'platform': 'esp32', 'family': 'micropython'}
5  # Stubber: 1.3.11
6  from typing import Any
7
8  class Signal:
9      ''
10     def __init__(self):
11         pass
12
13     def off(self) -> Any:
14         pass
15
16     def on(self) -> Any:
17         pass
18
19     def value(self) -> Any:
20         pass
21
22     def time_pulse_us() -> Any:
23         pass
24
25     def unique_id() -> Any:
26         pass
27
28     def wake_reason() -> Any:
29         pass

```

commands.samples.rich_source = Multiline-String

```

1  def time_pulse_us(pin:Pin, pulse_level:int, timeout_us:int=1000000, /) ->↳
   ↳ int:
2      """
3      Time a pulse on the given *pin*, and return the duration of the pulse in
4      microseconds. The *pulse_level* argument should be 0 to time a low↳
   ↳ pulse
5      or 1 to time a high pulse.
6
7      If the current input value of the pin is different to *pulse_level*,
8      the function first (*) waits until the pin input becomes equal to↳
   ↳ *pulse_level*,

```

(continues on next page)

(continued from previous page)

```

9      then (**) times the duration that the pin is equal to *pulse_level*.
10     If the pin is already equal to *pulse_level* then timing starts
↳straight away.

11
12     The function will return -2 if there was timeout waiting for condition
↳marked
13     (*) above, and -1 if there was timeout during the main measurement,
↳marked (**)
14     above. The timeout is the same for both cases and given by *timeout_us*
↳(which
15     is in microseconds).
16     """
17     ...
18
19
20 class Signal(Pin):
21     """The Signal class is a simple extension of the Pin class. Unlike Pin,
↳which can be only in "absolute"
22     0 and 1 states, a Signal can be in "asserted" (on) or "deasserted"
↳(off) states, while being inverted (active-low) or not.
23     In other words, it adds logical inversion support to Pin functionality.
↳While this may seem a simple addition, it is exactly what
24     is needed to support wide array of simple digital devices in a way
↳portable across different boards, which is one of the major
25     MicroPython goals. Regardless of whether different users have an active-
↳high or active-low LED, a normally open or normally closed
26     relay - you can develop a single, nicely looking application which
↳works with each of them, and capture hardware configuration
27     differences in few lines in the config file of your app.
28
29     """
30     def __init__(self, pin_obj:Pin, *,invert:bool=False):
31         """ Create a Signal object. There're two ways to create it:
32         By wrapping existing Pin object - universal method which works for
↳any board.
33         By passing required Pin parameters directly to Signal constructor,
↳skipping the need to create intermediate Pin object. Available on many,
↳but not all boards.
34         The arguments are:
35         pin_obj is existing Pin object.
36         pin_arguments are the same arguments as can be passed to Pin
↳constructor.
37         invert - if True, the signal will be inverted (active low).
38         """
39         pass
40
41
42     def off(self) -> None:
43         """ Activate signal.
44         """
45         pass
46

```

(continues on next page)

(continued from previous page)

```

47
48     def on(self) -> None:
49         """ Deactivate signal.
50         """
51         pass
52
53     def value(self) -> None:
54         """ This method allows to set and get the value of the signal,
↳ depending on whether the argument x is supplied or not.
55         If the argument is omitted then this method gets the signal
↳ level, 1 meaning signal is asserted (active) and 0 - signal inactive.
56         If the argument is supplied then this method sets the signal
↳ level. The argument x can be anything that converts to a boolean.
57         If it converts to True, the signal is active, otherwise it is
↳ inactive.
58         Correspondence between signal being active and actual logic
↳ level on the underlying pin depends on whether signal is inverted (active-
↳ low) or not.
59         For non-inverted signal, active status corresponds to logical 1,
↳ inactive - to logical 0. For inverted/active-low signal, active status
↳ corresponds to logical 0,
60         while inactive - to logical 1.
61         """
62         pass

```

19.9 basics

19.9.1 Module Contents

Classes

<i>TypingCollector</i>	The low-level base visitor class for traversing a CST. This should be used in
<i>TypingTransformer</i>	The low-level base visitor class for traversing a CST and creating an

Attributes

<i>source_tree</i>
<i>info_tree</i>
<i>visitor</i>
<i>transformer</i>

continues on next page

Table 49 – continued from previous page

modified_tree

class basics.**TypingCollector**

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_ClassDef(*self*, *node*: `libcst.ClassDef`) → `Optional[bool]`

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

Return type `Optional[bool]`

leave_ClassDef(*self*, *node*: `libcst.ClassDef`) → `None`

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

Return type `None`

visit_FunctionDef(*self*, *node*: `libcst.FunctionDef`) → `Optional[bool]`

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

Return type `Optional[bool]`

leave_FunctionDef(*self*, *node*: `libcst.FunctionDef`) → `None`

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

Return type `None`

class basics.**TypingTransformer**(*annotations*)

Bases: `libcst.CSTTransformer`

The low-level base visitor class for traversing a CST and creating an updated copy of the original 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, and possibly returning a new node in its place.

When visiting nodes using a `CSTTransformer`, the return value of `visit()` will be a new tree with any changes made in `on_leave()` calls reflected in its children.

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

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

Return type `Optional[bool]`

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

Parameters

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

Return type libbst.CSTNode

visit_FunctionDef(*self*, *node*: libbst.FunctionDef) → Optional[bool]

Parameters *node* (libbst.FunctionDef) –

Return type Optional[bool]

leave_FunctionDef(*self*, *original_node*: libbst.FunctionDef, *updated_node*: libbst.FunctionDef) → libbst.CSTNode

Parameters

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

Return type libbst.CSTNode

basics.source_tree

basics.info_tree

basics.visitor

basics.transformer

basics.modified_tree

19.10 ata_script

19.10.1 Module Contents

Functions

prRed(skk)

prGreen(skk)

Attributes

context

visitor

stubs_dir

sources_dir

stubs_dict

continues on next page

Table 51 – continued from previous page

<i>sources_dict</i>
<i>stubs_pathlist</i>
<i>rel_path</i>
<i>sources_pathlist</i>
<i>rel_path</i>
<i>stub</i>

```

ata_script.prRed(skk)
ata_script.prGreen(skk)
ata_script.context
ata_script.visitor
ata_script.stubs_dir = all-stubs/micropython-v1_16-documentation/
ata_script.sources_dir = dev-stubs/micropython-v1_16-esp32/
ata_script.stubs_dict
ata_script.sources_dict
ata_script.stubs_pathlist
ata_script.rel_path
ata_script.sources_pathlist
ata_script.rel_path
ata_script.stub

```

19.11 samples

19.11.1 Module Contents

`samples.simple_stub = Multiline-String`

```

1  """
2  Module: 'machine' on micropython-esp32-1.15
3  """
4  # MCU: {'ver': '1.15', 'port': 'esp32', 'arch': 'xtensawin', 'sysname':
    ↳ 'esp32', 'release': '1.15.0', 'name': 'micropython', 'mpy': 10757,
    ↳ 'version': '1.15.0', 'machine': 'ESP32 module (spiram) with ESP32', 'build
    ↳ ': '', 'nodename': 'esp32', 'platform': 'esp32', 'family': 'micropython'}
5  # Stubber: 1.3.11
6  from typing import Any
7

```

(continues on next page)

(continued from previous page)

```

8 class Signal:
9     ''
10    def __init__(self):
11        pass
12
13    def off(self) -> Any:
14        pass
15
16    def on(self) -> Any:
17        pass
18
19    def value(self) -> Any:
20        pass
21
22    def time_pulse_us() -> Any:
23        pass
24
25    def unique_id() -> Any:
26        pass
27
28    def wake_reason() -> Any:
29        pass

```

`samples.rich_source = Multiline-String`

```

1 def time_pulse_us(pin:Pin, pulse_level:int, timeout_us:int=1000000, /) ->
↳int:
2     """
3     Time a pulse on the given *pin*, and return the duration of the pulse in
4     microseconds. The *pulse_level* argument should be 0 to time a low
↳pulse
5     or 1 to time a high pulse.
6
7     If the current input value of the pin is different to *pulse_level*,
8     the function first (*) waits until the pin input becomes equal to
↳pulse_level*,
9     then (**) times the duration that the pin is equal to *pulse_level*.
10    If the pin is already equal to *pulse_level* then timing starts
↳straight away.
11
12    The function will return -2 if there was timeout waiting for condition
↳marked
13    (*) above, and -1 if there was timeout during the main measurement,
↳marked (**)
14    above. The timeout is the same for both cases and given by *timeout_us*
↳(which
15    is in microseconds).
16    """
17    ...
18
19

```

(continues on next page)

(continued from previous page)

```

20 class Signal(Pin):
21     """The Signal class is a simple extension of the Pin class. Unlike Pin,
↳ which can be only in "absolute"
22     0 and 1 states, a Signal can be in "asserted" (on) or "deasserted"
↳ (off) states, while being inverted (active-low) or not.
23     In other words, it adds logical inversion support to Pin functionality.
↳ While this may seem a simple addition, it is exactly what
24     is needed to support wide array of simple digital devices in a way
↳ portable across different boards, which is one of the major
25     MicroPython goals. Regardless of whether different users have an active-
↳ high or active-low LED, a normally open or normally closed
26     relay - you can develop a single, nicely looking application which
↳ works with each of them, and capture hardware configuration
27     differences in few lines in the config file of your app.
28
29     """
30     def __init__(self, pin_obj:Pin, *,invert:bool=False):
31         """ Create a Signal object. There're two ways to create it:
32         By wrapping existing Pin object - universal method which works for
↳ any board.
33         By passing required Pin parameters directly to Signal constructor,
↳ skipping the need to create intermediate Pin object. Available on many,
↳ but not all boards.
34         The arguments are:
35         pin_obj is existing Pin object.
36         pin_arguments are the same arguments as can be passed to Pin
↳ constructor.
37         invert - if True, the signal will be inverted (active low).
38         """
39         pass
40
41
42     def off(self) -> None:
43         """ Activate signal.
44         """
45         pass
46
47
48     def on(self) -> None:
49         """ Deactivate signal.
50         """
51         pass
52
53     def value(self) -> None:
54         """ This method allows to set and get the value of the signal,
↳ depending on whether the argument x is supplied or not.
55         If the argument is omitted then this method gets the signal
↳ level, 1 meaning signal is asserted (active) and 0 - signal inactive.
56         If the argument is supplied then this method sets the signal
↳ level. The argument x can be anything that converts to a boolean.
57         If it converts to True, the signal is active, otherwise it is
↳ inactive.

```

(continues on next page)

(continued from previous page)

```

58         Correspondence between signal being active and actual logic_
↪ level on the underlying pin depends on whether signal is inverted (active-
↪ low) or not.
59         For non-inverted signal, active status corresponds to logical 1,
↪ inactive - to logical 0. For inverted/active-low signal, active status_
↪ corresponds to logical 0,
60         while inactive - to logical 1.
61         """
62         pass

```

19.12 simple

19.12.1 Module Contents

`simple.py_source = Multiline-String`

```

1  'moduledoc'
2  class Signal:
3      ''
4      def value(self):
5          ''
6          pass
7
8  class Foo:
9      ''
10     ''
11     def __init__(self):
12         ''
13         pass

```

`simple.config`

`simple.source_tree`

`simple.classdef`

`simple.body`

`simple.expr`

`simple.expr`

`simple.expr`

`simple.new_docstr`

`simple.newtree`

19.13 docstrings

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

a

`ata_script`, 104

b

`basics`, 102

c

`commands`, 98

`commands.constant_folding`, 98

`commands.noop`, 99

`commands.samples`, 100

`createtubs`, 55

`createtubs_db`, 52

`createtubs_mem`, 49

d

`docstrings`, 109

p

`pyboard`, 89

s

`samples`, 105

`simple`, 108

`stub_lvgl`, 52

`stubber`, 57

`stubber.basicgit`, 70

`stubber.downloader`, 72

`stubber.get_cpython`, 73

`stubber.get_lobo`, 73

`stubber.get_mpy`, 74

`stubber.makemaniifest_2`, 75

`stubber.minify`, 78

`stubber.rst`, 57

`stubber.rst.classsort`, 57

`stubber.rst.lookup`, 58

`stubber.rst.output_dict`, 58

`stubber.rst.report_return`, 61

`stubber.rst.rst_utils`, 62

`stubber.stubber`, 79

`stubber.stubs_from_docs`, 82

`stubber.update_module_list`, 86

`stubber.utils`, 87

v

`visitors`, 92

`visitors._apply_stubber_annotations`, 92

Symbols

- `__MAX_CLASS_LEVEL` (in module `createstubs`), 56
- `__MAX_CLASS_LEVEL` (in module `createstubs_db`), 53
- `__MAX_CLASS_LEVEL` (in module `createstubs_mem`), 50
- `__add__` () (stubber.rst.SourceDict method), 66
- `__add__` () (stubber.rst.output_dict.SourceDict method), 59
- `__all__` (in module `stubber.rst`), 70
- `__del__` () (pyboard.TelnetToSerial method), 91
- `__getattr__` () (stubber.makemanifest_2.IncludeOptions method), 76
- `__str__` () (stubber.rst.ModuleSourceDict method), 67
- `__str__` () (stubber.rst.SourceDict method), 66
- `__str__` () (stubber.rst.output_dict.ModuleSourceDict method), 60
- `__str__` () (stubber.rst.output_dict.SourceDict method), 59
- `__version__` (in module `createstubs`), 55
- `__version__` (in module `createstubs_db`), 53
- `__version__` (in module `createstubs_mem`), 50
- `_add_annotation_to_imports` () (visitors._apply_stubber_annotations.TypeCollector method), 94
- `_add_to_toplevel_annotations` () (visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor method), 96
- `_annotate_single_target` () (visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor method), 96
- `_color` () (in module `stubber.stubs_from_docs`), 84
- `_create_import_from_annotation` () (visitors._apply_stubber_annotations.TypeCollector method), 94
- `_get_import_alias_names` () (in module visitors._apply_stubber_annotations), 93
- `_get_import_names` () (in module visitors._apply_stubber_annotations), 93
- `_green` () (in module `stubber.stubs_from_docs`), 84
- `_handle_Index` () (visitors._apply_stubber_annotations.TypeCollector method), 94
- `_handle_Subscript` () (visitors._apply_stubber_annotations.TypeCollector method), 94
- `_import_parameter_annotations` () (visitors._apply_stubber_annotations.TypeCollector method), 94
- `_info` () (in module `createstubs`), 57
- `_info` () (in module `createstubs_db`), 54
- `_info` () (in module `createstubs_mem`), 51
- `_injected_import_hook_code` (in module `pyboard`), 92
- `_insert_empty_line` () (visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor method), 97
- `_log` (in module `createstubs`), 57
- `_log` (in module `createstubs_db`), 55
- `_log` (in module `createstubs_mem`), 51
- `_qualifier_name` () (visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor method), 96
- `_red` () (in module `stubber.stubs_from_docs`), 84
- `_run_git` () (in module `stubber.basicgit`), 70
- `_split_module` () (visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor method), 96
- `_type_from_context` () (in module `stubber.rst`), 69
- `_type_from_context` () (in module `stubber.rst.rst_utils`), 64
- `_update_parameters` () (visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor method), 96

A

- `add_args` () (commands.constant_folding.ConvertConstantCommand static method), 98
- `add_comment` () (stubber.rst.output_dict.SourceDict method), 59
- `add_comment` () (stubber.rst.SourceDict method), 66
- `add_constant` () (stubber.rst.output_dict.SourceDict method), 60
- `add_constant` () (stubber.rst.SourceDict method), 67
- `add_constant_smart` () (stubber.rst.output_dict.SourceDict method), 60

- 60
- `add_constant_smart()` (*stubber.rst.SourceDict method*), 67
- `add_docstr()` (*stubber.rst.output_dict.SourceDict method*), 59
- `add_docstr()` (*stubber.rst.SourceDict method*), 66
- `add_import()` (*stubber.rst.ModuleSourceDict method*), 68
- `add_import()` (*stubber.rst.output_dict.ModuleSourceDict method*), 61
- `add_line()` (*stubber.rst.output_dict.SourceDict method*), 60
- `add_line()` (*stubber.rst.SourceDict method*), 67
- `add_modules()` (*createstubs.Stubber method*), 56
- `add_modules()` (*createstubs_db.Stubber method*), 53
- `add_modules()` (*createstubs_mem.Stubber method*), 50
- `Annotations` (*class in visitors._apply_stubber_annotations*), 95
- `ApplyStubberAnnotationsVisitor` (*class in visitors._apply_stubber_annotations*), 95
- `at_anchor()` (*stubber.stubs_from_docs.RSTReader method*), 84
- `at_heading()` (*stubber.stubs_from_docs.RSTReader method*), 85
- `ata_script` *module*, 104
- `attribute_annotations` (*visitors._apply_stubber_annotations.Annotations attribute*), 95
- B**
- `basics` *module*, 102
- `body` (*in module simple*), 108
- C**
- `checkout_commit()` (*in module stubber.basicgit*), 71
- `checkout_tag()` (*in module stubber.basicgit*), 71
- `CHILD_PARENT_CLASS` (*in module stubber.rst*), 66
- `CHILD_PARENT_CLASS` (*in module stubber.rst.lookup*), 58
- `class_definitions` (*visitors._apply_stubber_annotations.Annotations attribute*), 95
- `classdef` (*in module simple*), 108
- `classes()` (*stubber.rst.ModuleSourceDict method*), 67
- `classes()` (*stubber.rst.output_dict.ModuleSourceDict method*), 60
- `ClassSourceDict` (*class in stubber.rst*), 68
- `ClassSourceDict` (*class in stubber.rst.output_dict*), 61
- `clean()` (*createstubs.Stubber method*), 56
- `clean()` (*createstubs_db.Stubber method*), 54
- `clean()` (*createstubs_mem.Stubber method*), 51
- `clean_version()` (*in module stubber.utils*), 87
- `cleanup()` (*in module stubber.utils*), 88
- `cli_docstubs()` (*in module stubber.stubber*), 81
- `cli_get_core()` (*in module stubber.stubber*), 81
- `cli_get_frozen()` (*in module stubber.stubber*), 81
- `cli_get_lobo()` (*in module stubber.stubber*), 81
- `cli_init()` (*in module stubber.stubber*), 80
- `cli_minify()` (*in module stubber.stubber*), 80
- `cli_stub()` (*in module stubber.stubber*), 80
- `clone()` (*in module stubber.basicgit*), 70
- `close()` (*pyboard.ProcessPtyToTerminal method*), 91
- `close()` (*pyboard.ProcessToSerial method*), 91
- `close()` (*pyboard.Pyboard method*), 91
- `close()` (*pyboard.TelnetToSerial method*), 91
- `commands` *module*, 98
- `commands.constant_folding` *module*, 98
- `commands.noop` *module*, 99
- `commands.samples` *module*, 100
- `compound_candidates()` (*in module stubber.rst*), 68
- `compound_candidates()` (*in module stubber.rst.rst_utils*), 63
- `config` (*in module simple*), 108
- `context` (*in module ata_script*), 105
- `CONTEXT_KEY` (*visitors._apply_stubber_annotations.ApplyStubberAnnotations attribute*), 95
- `convert_path()` (*in module stubber.makemanifest_2*), 77
- `ConvertConstantCommand` (*class in commands.constant_folding*), 98
- `create_all_stubs()` (*createstubs.Stubber method*), 56
- `create_all_stubs()` (*createstubs_db.Stubber method*), 54
- `create_all_stubs()` (*createstubs_mem.Stubber method*), 50
- `create_module_stub()` (*createstubs.Stubber method*), 56
- `create_module_stub()` (*createstubs_db.Stubber method*), 54
- `create_module_stub()` (*createstubs_mem.Stubber method*), 50
- `create_one_stub()` (*createstubs.Stubber method*), 56
- `create_one_stub()` (*createstubs_db.Stubber method*), 54
- `create_one_stub()` (*createstubs_mem.Stubber method*), 50
- `create_update_class()` (*stubber.stubs_from_docs.RSTReader method*), 85
- `createstubs` *module*, 55
- `createstubs_db` *module*, 52

createstubs_mem
module, 49

D

defaults() (stubber.makemanifest_2.IncludeOptions
method), 76

DESCRIPTION (commands.constant_folding.ConvertConstant
attribute), 98

DESCRIPTION (commands.noop.NOOPCommand at-
tribute), 99

distill_return() (in module stubber.rst), 69

distill_return() (in module stubber.rst.rst_utils), 64

do_post_processing() (in module stubber.stubber),
80

docstrings
module, 109

download_file() (in module stubber.downloader), 72

download_files() (in module stubber.downloader), 72

E

edit_lines() (in module stubber.minify), 78

ENOENT (in module createstubs), 55

ENOENT (in module createstubs_db), 53

ENOENT (in module createstubs_mem), 50

ensure_folder() (in module createstubs), 56

ensure_folder() (in module createstubs_db), 54

ensure_folder() (in module createstubs_mem), 51

enter_raw_repl() (pyboard.Pyboard method), 91

eval() (pyboard.Pyboard method), 91

exec_() (pyboard.Pyboard method), 91

exec_raw() (pyboard.Pyboard method), 91

exec_raw_no_follow() (pyboard.Pyboard method), 91

execfile() (in module pyboard), 92

execfile() (pyboard.Pyboard method), 91

exit_raw_repl() (pyboard.Pyboard method), 91

expr (in module simple), 108

F

FAMILY (in module stubber.get_lobo), 73

FAMILY (in module stubber.get_mpy), 74

fetch() (in module stubber.basicgit), 71

filesystem_command() (in module pyboard), 92

find() (stubber.rst.ModuleSourceDict method), 67

find() (stubber.rst.output_dict.ModuleSourceDict
method), 60

find() (stubber.rst.output_dict.SourceDict method), 60

find() (stubber.rst.SourceDict method), 67

fix_parameters() (stub-
ber.stubs_from_docs.RSTReader method),
85

fix_umqtt_init() (in module stubber.utils), 88

flat_fwid (createstubs.Stubber property), 56

flat_fwid (createstubs_db.Stubber property), 54

flat_fwid (createstubs_mem.Stubber property), 51

follow() (pyboard.Pyboard method), 91

freeze() (in module stubber.makemanifest_2), 76

freeze_as_mpy() (in module stubber.makemanifest_2),
76

freeze_as_str() (in module stubber.makemanifest_2),
76

freeze_internal() (in module stub-
ber.makemanifest_2), 77

freeze_mpy() (in module stubber.makemanifest_2), 76
FreezeError, 76

fs_cat() (pyboard.Pyboard method), 91

fs_get() (pyboard.Pyboard method), 91

fs_ls() (pyboard.Pyboard method), 91

fs_mkdir() (pyboard.Pyboard method), 91

fs_put() (pyboard.Pyboard method), 91

fs_rm() (pyboard.Pyboard method), 92

fs_rmdir() (pyboard.Pyboard method), 91

function_annotations (visi-
tors._apply_stubber_annotations.Annotations
attribute), 95

FunctionAnnotation (class in visi-
tors._apply_stubber_annotations), 93

FunctionSourceDict (class in stubber.rst), 68

FunctionSourceDict (class in stubber.rst.output_dict),
61

G

gather_docs (stubber.stubs_from_docs.RSTReader at-
tribute), 84

generate_from_rst() (in module stub-
ber.stubs_from_docs), 85

generate_pyi_files() (in module stubber.utils), 88

generate_pyi_from_file() (in module stubber.utils),
88

get_core() (in module stubber.get_cpython), 73

get_frozen() (in module stubber.get_lobo), 74

get_frozen() (in module stubber.get_mpy), 74

get_frozen_folders() (in module stubber.get_mpy),
75

get_frozen_from_manifest() (in module stub-
ber.get_mpy), 75

get_obj_attributes() (createstubs.Stubber method),
56

get_obj_attributes() (createstubs_db.Stubber
method), 53

get_obj_attributes() (createstubs_mem.Stubber
method), 50

get_root() (in module createstubs), 57

get_root() (in module createstubs_db), 54

get_root() (in module createstubs_mem), 51

get_rst_hint() (stubber.stubs_from_docs.RSTReader
method), 85

get_tag() (in module stubber.basicgit), 71

get_target_names() (in module stubber.get_mpy), 75

`get_time()` (*pyboard.Pyboard* method), 91

I

`include()` (in module *stubber.makemanifest_2*), 77

`IncludeOptions` (class in *stubber.makemanifest_2*), 76

`index()` (*stubber.rst.output_dict.SourceDict* method), 60

`index()` (*stubber.rst.SourceDict* method), 67

`info_tree` (in module *basics*), 104

`inWaiting()` (*pyboard.ProcessPtyToTerminal* method), 91

`inWaiting()` (*pyboard.ProcessToSerial* method), 91

`inWaiting()` (*pyboard.TelnetToSerial* method), 91

`isMicroPython()` (in module *createstubs*), 57

`isMicroPython()` (in module *createstubs_db*), 55

`isMicroPython()` (in module *createstubs_mem*), 51

L

`leave_AnnAssign()` (*visitors._apply_stubber_annotations.TypeCollector* method), 94

`leave_Assign()` (*visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor* method), 97

`leave_class()` (*stubber.stubs_from_docs.RSTReader* method), 84

`leave_ClassDef()` (*basics.TypeingCollector* method), 103

`leave_ClassDef()` (*basics.TypeingTransformer* method), 103

`leave_ClassDef()` (*visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor* method), 97

`leave_ClassDef()` (*visitors._apply_stubber_annotations.TypeCollector* method), 93

`leave_FunctionDef()` (*basics.TypeingCollector* method), 103

`leave_FunctionDef()` (*basics.TypeingTransformer* method), 104

`leave_FunctionDef()` (*visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor* method), 97

`leave_FunctionDef()` (*visitors._apply_stubber_annotations.TypeCollector* method), 94

`leave_ImportFrom()` (*visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor* method), 98

`leave_Module()` (*visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor* method), 98

`leave_SimpleString()` (*commands.constant_folding.ConvertConstantCommand* method), 99

`line` (*stubber.stubs_from_docs.RSTReader* property), 84

`log` (in module *stubber.downloader*), 72

`log` (in module *stubber.get_cpython*), 73

`log` (in module *stubber.get_lobo*), 74

`log` (in module *stubber.get_mpy*), 74

`log` (in module *stubber.makemanifest_2*), 76

`log` (in module *stubber.stubber*), 80

`log` (in module *stubber.stubs_from_docs*), 83

`log` (in module *stubber.utils*), 87

`LOOKUP_LIST` (in module *stubber.rst*), 66

`LOOKUP_LIST` (in module *stubber.rst.lookup*), 58

M

`main()` (in module *createstubs*), 57

`main()` (in module *createstubs_mem*), 51

`main()` (in module *pyboard*), 92

`main()` (in module *stub_lvgl*), 52

`main()` (in module *stubber.update_module_list*), 86

`main_esp8266()` (in module *createstubs_db*), 55

`make_manifest()` (in module *stubber.utils*), 89

`manifest()` (in module *stubber.utils*), 88

`minify()` (in module *stubber.minify*), 78

`minify_script()` (in module *stubber.minify*), 78

`modified_tree` (in module *basics*), 104

module

ata_script, 104

basics, 102

commands, 98

commands.constant_folding, 98

commands.noop, 99

commands.samples, 100

createstubs, 55

createstubs_db, 52

createstubs_mem, 49

docstrings, 109

pyboard, 89

samples, 105

simple, 108

stub_lvgl, 52

stubber, 57

stubber.basicgit, 70

stubber.downloader, 72

stubber.get_cpython, 73

stubber.get_lobo, 73

stubber.get_mpy, 74

stubber.makemanifest_2, 75

stubber.minify, 78

stubber.rst, 57

stubber.rst.classsort, 57

stubber.rst.lookup, 58

stubber.rst.output_dict, 58

stubber.rst.report_return, 61

stubber.rst.rst_utils, 62

stubber.stubber, 79

stubber.stubs_from_docs, 82
 stubber.update_module_list, 86
 stubber.utils, 87
 visitors, 92
 visitors._apply_stubber_annotations, 92
 MODULE_GLUE (in module stubber.rst), 66
 MODULE_GLUE (in module stubber.rst.lookup), 58
 module_names (stubber.stubs_from_docs.RSTReader
 property), 84
 ModuleSourceDict (class in stubber.rst), 67
 ModuleSourceDict (class in stubber.rst.output_dict), 60
 MPY_FOLDER (in module stubber.stubber), 80
 MPY_LIB_FOLDER (in module stubber.stubber), 80
 mpy_path (in module stubber.get_mpy), 75

N

new_docstr (in module simple), 108
 NEW_OUTPUT (in module stubber.stubs_from_docs), 84
 newtree (in module simple), 108
 NONE_VERBS (in module stubber.rst), 66
 NONE_VERBS (in module stubber.rst.lookup), 58
 NOOPCommand (class in commands.noop), 99

O

object_candidates() (in module stubber.rst), 69
 object_candidates() (in module stubber.rst.rst_utils),
 63

P

PARAM_FIXES (in module stubber.rst), 66
 PARAM_FIXES (in module stubber.rst.lookup), 58
 parameters (visitors._apply_stubber_annotations.FunctionAnnotation
 attribute), 93
 parse() (stubber.stubs_from_docs.RSTReader method),
 85
 parse_class() (stubber.stubs_from_docs.RSTReader
 method), 85
 parse_current_module() (stub-
 ber.stubs_from_docs.RSTReader method),
 85
 parse_data() (stubber.stubs_from_docs.RSTReader
 method), 85
 parse_docstring() (stub-
 ber.stubs_from_docs.RSTReader method),
 85
 parse_exception() (stub-
 ber.stubs_from_docs.RSTReader method),
 85
 parse_function() (stub-
 ber.stubs_from_docs.RSTReader method),
 85
 parse_method() (stubber.stubs_from_docs.RSTReader
 method), 85

parse_module() (stubber.stubs_from_docs.RSTReader
 method), 85
 parse_name() (stubber.stubs_from_docs.RSTReader
 method), 85
 parse_names() (stubber.stubs_from_docs.RSTReader
 method), 85
 parse_toc() (stubber.stubs_from_docs.RSTReader
 method), 85
 path_vars (in module stubber.makemanifest_2), 76
 PORT (in module stubber.get_lobo), 74
 prepare_output() (stub-
 ber.stubs_from_docs.RSTReader method),
 84
 prGreen() (in module ata_script), 105
 process() (in module stubber.rst.report_return), 61
 ProcessPtyToTerminal (class in pyboard), 91
 ProcessToSerial (class in pyboard), 91
 prRed() (in module ata_script), 105
 pull() (in module stubber.basicgit), 72
 py_source (in module simple), 108
 pyboard
 module, 89
 Pyboard (class in pyboard), 91
 PyboardError, 90
 python_minifier (in module stubber.minify), 78

R

raw_paste_write() (pyboard.Pyboard method), 91
 read() (pyboard.ProcessPtyToTerminal method), 91
 read() (pyboard.ProcessToSerial method), 91
 read() (pyboard.TelnetToSerial method), 91
 read_exclusion_file() (in module stubber.utils), 89
 read_file() (stubber.stubs_from_docs.RSTReader
 method), 84
 read_micropython_lib_commits() (in module stub-
 ber.get_mpy), 75
 read_modules() (in module stub-
 ber.update_module_list), 86
 read_path() (in module createstubs), 57
 read_path() (in module createstubs_db), 54
 read_path() (in module createstubs_mem), 51
 read_until() (pyboard.Pyboard method), 91
 rel_path (in module ata_script), 105
 report() (createstubs.Stubber method), 56
 report() (createstubs_db.Stubber method), 54
 report() (createstubs_mem.Stubber method), 51
 resetWDT() (in module createstubs), 56
 resetWDT() (in module createstubs_db), 53
 resetWDT() (in module createstubs_mem), 50
 return_type_from_context() (in module stub-
 ber.rst), 69
 return_type_from_context() (in module stub-
 ber.rst.rst_utils), 64

returns (visitors._apply_stubber_annotations.FunctionAnnotationsVisitor attribute), 93
 rich_source (in module commands.samples), 100
 rich_source (in module samples), 106
 RST_DOC_FIXES (in module stubber.rst), 66
 RST_DOC_FIXES (in module stubber.rst.lookup), 58
 RSTReader (class in stubber.stubs_from_docs), 84

S

samples
 module, 105
 SEPARATOR (in module stubber.stubs_from_docs), 84
 should_ignore() (in module stubber.utils), 89
 show_help() (in module createstubs), 57
 show_help() (in module createstubs_db), 54
 show_help() (in module createstubs_mem), 51
 simple
 module, 108
 simple_candidates() (in module stubber.rst), 68
 simple_candidates() (in module stubber.rst.rst_utils), 63
 simple_stub (in module commands.samples), 100
 simple_stub (in module samples), 105
 sort() (stubber.rst.ModuleSourceDict method), 67
 sort() (stubber.rst.output_dict.ModuleSourceDict method), 60
 sort_classes() (in module stubber.rst), 66
 sort_classes() (in module stubber.rst.classsort), 57
 source_tree (in module basics), 104
 source_tree (in module simple), 108
 SourceDict (class in stubber.rst), 66
 SourceDict (class in stubber.rst.output_dict), 59
 sources_dict (in module ata_script), 105
 sources_dir (in module ata_script), 105
 sources_pathlist (in module ata_script), 105
 stdout (in module pyboard), 90
 stdout_write_bytes() (in module pyboard), 90
 store_stub_in_context() (visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor static method), 95
 strip_prefixes() (stubber.stubs_from_docs.RSTReader method), 84
 stub (in module ata_script), 105
 stub_dir (in module stubber.makemanifest_2), 77
 STUB_FOLDER (in module stubber.stubber), 80
 STUB_FOLDER (in module stubber.utils), 87
 stub_lvgl
 module, 52
 stubber
 module, 57
 Stubber (class in createstubs), 56
 Stubber (class in createstubs_db), 53
 Stubber (class in createstubs_mem), 50
 stubber.basicgit
 module, 70
 stubber.downloader
 module, 72
 stubber.get_cpython
 module, 73
 stubber.get_lobo
 module, 73
 stubber.get_mpy
 module, 74
 stubber.makemanifest_2
 module, 75
 stubber.minify
 module, 78
 stubber.rst
 module, 57
 stubber.rst.classsort
 module, 57
 stubber.rst.lookup
 module, 58
 stubber.rst.output_dict
 module, 58
 stubber.rst.report_return
 module, 61
 stubber.rst.rst_utils
 module, 62
 stubber.stubber
 module, 79
 stubber.stubs_from_docs
 module, 82
 stubber.update_module_list
 module, 86
 stubber.utils
 module, 87
 stubber_cli() (in module stubber.stubber), 80
 stubfolder() (in module stubber.utils), 88
 STUBGEN_OPT (in module stubber.utils), 87
 stubs_dict (in module ata_script), 105
 stubs_dir (in module ata_script), 105
 stubs_pathlist (in module ata_script), 105
 switch_branch() (in module stubber.basicgit), 71
 switch_tag() (in module stubber.basicgit), 71

T

target (stubber.stubs_from_docs.RSTReader attribute), 84
 TelnetToSerial (class in pyboard), 91
 transform_module_impl() (commands.noop.NOOPCommand method), 99
 transform_module_impl() (visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor method), 96
 transformer (in module basics), 104

TypeCollector (class in visitors._apply_stubber_annotations), 93
 write_object_stub() (createstubs_mem.Stubber method), 50
 TYPING_IMPORT (in module stubber.rst), 70
 TYPING_IMPORT (in module stubber.rst.rst_utils), 63
 TypingCollector (class in basics), 103
 TypingTransformer (class in basics), 103

V

verbose (stubber.stubs_from_docs.RSTReader attribute), 84
 visit_AnnAssign() (visitors._apply_stubber_annotations.TypeCollector method), 94
 visit_ClassDef() (basics.TypingCollector method), 103
 visit_ClassDef() (basics.TypingTransformer method), 103
 visit_ClassDef() (visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor method), 97
 visit_ClassDef() (visitors._apply_stubber_annotations.TypeCollector method), 93
 visit_FunctionDef() (basics.TypingCollector method), 103
 visit_FunctionDef() (basics.TypingTransformer method), 104
 visit_FunctionDef() (visitors._apply_stubber_annotations.ApplyStubberAnnotationsVisitor method), 97
 visit_FunctionDef() (visitors._apply_stubber_annotations.TypeCollector method), 94
 visit_ImportFrom() (visitors._apply_stubber_annotations.TypeCollector method), 94
 visitor (in module ata_script), 105
 visitor (in module basics), 104
 visitors
 module, 92
 visitors._apply_stubber_annotations
 module, 92

W

wrapped() (in module stubber.update_module_list), 86
 write() (pyboard.ProcessPtyToTerminal method), 91
 write() (pyboard.ProcessToSerial method), 91
 write() (pyboard.TelnetToSerial method), 91
 write_file() (stubber.stubs_from_docs.RSTReader method), 84
 write_object_stub() (createstubs.Stubber method), 56
 write_object_stub() (createstubs_db.Stubber method), 54