
multidoc

Release 0.0.1

G.H. Garrett

Apr 29, 2022

MODULES:

1	multidoc.parsing	3
1.1	Functions	3
1.2	Helper data models	3
1.3	Core data models	3
1.4	Reference	4
2	multidoc.template	11
3	multidoc.generate	13
4	multidoc.regex	15
5	multidoc.error	19
6	Indices and tables	21
	Python Module Index	23
	Index	25

Creating and maintaining an API Reference for multiple programming languages consumes unnecessary time. multidoc attempts to solve this through the use of jinja2 templating and pydantic data modelling. At the core, is the API Declaration:

Listing 1: api/hello.yaml

```
summary: This is the hello module...
notes: It's probably not that impressive

functions:
  - name: hello
    returns:
      name: world
      type: str
      description: Returns the world!
```

With the API Declaration in hand, we can add replacement tags to the C++ source code:

Listing 2: cpp_src/foo.h

```
///! @get_docstring(__doc__)
#include <string>

namespace hello {

///! @get_docstring(world)
std::string world();

}
```

In the pybind11 exposed module, docstrings can also be added:

Listing 3: pybind_src/foo.cpp

```
#include <pybind11/pybind11.h>
#include <pybind_src/docstrings.h>
#include <cpp/foo.h>

namespace py = pybind11;

PYBIND11_MODULE(hello, m) {
    m.doc() = hello::get_docstring("__doc__");
    m.def("world", &hello::world, hello::get_docstring("world"));
}
```

```
>>> generate_documented(src="pybind_src", dest=".", local={"py": True}, api="api/module.
↳yaml")
>>> generate_documented(src="cpp_src", dest=".", local={"cpp": True}, api="api/module.
↳yaml"))
```


MULTIDOC.PARSING

1.1 Functions

<i>yaml2dict</i> (path[, include_name_error])	Yaml file parser.
---	-------------------

1.2 Helper data models

<i>Parameter</i> (*[, name[, type, description]])	Parameter docstring pydantic.BaseModel data structure.
<i>Returns</i> (*[, name, type, description])	Returns docstring pydantic.BaseModel data structure.
<i>Config</i> (*[, name, version])	Multidoc module config pydantic.BaseModel data structure.

1.3 Core data models

<i>Function</i> (*[, name[, short_summary, ...]])	Function docstring pydantic.BaseModel data structure.
<i>Class</i> (*[, name[, short_summary, ...]])	Class docstring pydantic.BaseModel data structure.
<i>Constant</i> (*[, summary[, extended_summary, ...]])	Constant docstring pydantic.BaseModel data structure.
<i>FileBased</i> ()	FileBased declaration pydantic.BaseModel data structure.
<i>Module</i> (*[, config, summary, ...])	Module docstring pydantic.BaseModel data structure.
<i>Package</i> (*[, config, summary, ...])	Module docstring pydantic.BaseModel data structure.

1.4 Reference

```
class multidoc.parsing.Class(*, name: str, short_summary: str = None, deprecation_warning: str = None,
    extended_summary: str = None, parameters:
        List[multidoc.parsing.models.Parameter] = None, attributes:
        List[multidoc.parsing.models.Parameter] = None, properties:
        List[multidoc.parsing.models.Property] = None, yields:
        List[multidoc.parsing.models.Yields] = None, other_parameters:
        List[multidoc.parsing.models.Parameter] = None, raises:
        List[multidoc.parsing.models.Raises] = None, warns:
        List[multidoc.parsing.models.Raises] = None, warnings: str = None, see_also:
        str = None, notes: str = None, references: str = None, examples: str = None,
        methods: List[multidoc.parsing.models.Function] = None, autoclass:
        multidoc.parsing.models.AutoClassConfig = None)
```

Class docstring pydantic.BaseModel data structure.

name

Type *str*

short_summary

Type Optional[*str*] # test

deprecation_warning

Type Optional[*str*]

extended_summary

Type Optional[*str*]

parameters

Type Optional[List[*Parameter*]]

returns

Type Optional[*Returns*] or Optional[List[*Returns*]]

yields

Type Optional[List[*Yields*] or *Yields*]

other_parameters

Type Optional[List[*Parameter*]]

raises

Type Optional[List[*Raises*] or *Raises*]

warns

Type Optional[List[*Raises*] or *Raises*]

warnings

Type Optional[*str*]

see_also

Type Optional[str]

notes

Type Optional[str]

references

Type Optional[str]

examples

Type Optional[str]

methods

Type Optional[str]

class multidoc.parsing.**Config**(*, name: str = None, version: str = None)

Multidoc module config pydantic.BaseModel data structure.

name

Type Optional[str]

version

Type Optional[str]

class multidoc.parsing.**Constant**(*, summary: str, extended_summary: str = None, see_also: str = None, references: str = None, examples: str = None)

Constant docstring pydantic.BaseModel data structure.

summary

Type str

extended_summary

Type Optional[str]

see_also

Type Optional[str]

references

Type Optional[str]

examples

Type Optional[str]

class multidoc.parsing.**FileBased**

FileBased declaration pydantic.BaseModel data structure.

classmethod parse_yaml(path, **kwargs)

Parameters

- path –
- local –

```
class multidoc.parsing.Function(*, name: str, short_summary: str = None, deprecation_warning: str =
    None, extended_summary: str = None, parameters:
    List[multidoc.parsing.models.Parameter] = None, returns:
    multidoc.parsing.models>Returns = None, yields:
    List[multidoc.parsing.models.Yields] = None, other_parameters:
    List[multidoc.parsing.models.Parameter] = None, raises:
    List[multidoc.parsing.models.Raises] = None, warns:
    List[multidoc.parsing.models.Raises] = None, warnings: str = None,
    see_also: str = None, notes: str = None, references: str = None, examples:
    str = None)
```

Function docstring pydantic.BaseModel data structure.

name

Type str

short_summary

Type Optional[str] # test

deprecation_warning

Type Optional[str]

extended_summary

Type Optional[str]

parameters

Type Optional[List[Parameter]]

returns

Type Optional>Returns or Optional[List>Returns]]

yields

Type Optional[List[Yields] or Yields]

other_parameters

Type Optional[List[Parameter]]

raises

Type Optional[List[Raises] or Raises]

warns

Type Optional[List[Raises] or Raises]

warnings

Type Optional[str]

see_also

Type Optional[str]

notes

Type Optional[str]

references

Type Optional[str]

examples

Type Optional[str]

```
class multidoc.parsing.Module(*, config: multidoc.parsing.models.Config = None, summary: str = None,
    extended_summary: str = None, routine_listings: str = None, see_also: str =
    None, notes: str = None, references: str = None, examples: str = None,
    enums: List[multidoc.parsing.models.Enum] = None, classes:
    List[multidoc.parsing.models.Class] = None, functions:
    List[multidoc.parsing.models.Function] = None, constants:
    List[multidoc.parsing.models.Constant] = None)
```

Module docstring pydantic.BaseModel data structure.

config

Type Optional[Config]

summary

Type Optional[str]

extended_summary

Type Optional[str]

routine_listings

Type Optional[str]

see_also

Type Optional[str]

notes

Type Optional[str]

references

Type Optional[str]

examples

Type Optional[str]

classes

Type Optional[List[Class]]

functions

Type Optional[List[Function]]

constants

Type Optional[List[Constant]]

```
class multidoc.parsing.Package(*, config: multidoc.parsing.models.Config = None, summary: str = None,
    extended_summary: str = None, routine_listings: str = None, see_also: str
    = None, notes: str = None, references: str = None, examples: str = None,
    enums: List[multidoc.parsing.models.Enum] = None, classes:
    List[multidoc.parsing.models.Class] = None, functions:
    List[multidoc.parsing.models.Function] = None, constants:
    List[multidoc.parsing.models.Constant] = None, modules: List[str] = None)
```

Module docstring pydantic.BaseModel data structure.

config

Type Optional[Config]

summary

Type Optional[str]

extended_summary

Type Optional[str]

routine_listings

Type Optional[str]

see_also

Type Optional[str]

notes

Type Optional[str]

references

Type Optional[str]

examples

Type Optional[str]

classes

Type Optional[List[Class]]

functions

Type Optional[List[Function]]

constants

Type Optional[List[Constant]]

modules

Type Optional[List[str]]

```
class multidoc.parsing.Parameter(*, name: str, type: str = None, description: str = None)
```

Parameter docstring pydantic.BaseModel data structure.

name

Type str

type

Type Optional[str]

description

Type Optional[str]

Examples

Listing 1: Used in *Function* parameters.

```
functions:
- name: foo_function
  parameters:
  - name: bar_parameter
    type: Any
    description: "The bar parameter"
```

Listing 2: Used in *Class* method parameters.

```
classes:
- name: FooClass
  methods:
  - name: bar_method
    parameters:
    - name: bar_parameter
      type: Any
      description: "The bar parameter"
```

class multidoc.parsing.**Returns**(**name*: str = None, *type*: str = None, *description*: str = None)

Returns docstring pydantic.BaseModel data structure.

name

Type Optional[str]

type

Type Optional[str]

description

Type str

multidoc.parsing.parse_api_declaration(*path*: str, *parent*=None, **kwargs)

Parameters

- **path** –
- **local** –
- **parent** –

multidoc.parsing.yaml2dict(*path*, *include_name_error*=False, **kwargs)

Yaml file parser.

Parameters

- **path** (*os.PathLike* or *str*) – Path of the yaml file to be loaded.
- **_locals** (*dict[str, Any]*) – List of definitions to parse in the yaml files. See examples for how they affect yaml loading.
- **include_name_error** (*bool*, *default=True*) – Include tag evaluations that return a `NameError`

Examples

Given the following example yaml file:

Listing 3: example.yaml

```
package:
  name: name-cpp      # [cpp]
  name: name-py       # [py]

modules:
  - module
  - module-py         # [py]
  - module-cpp        # [cpp]
  - module-not-cpp    # [not cpp]
  - module-not-py     # [not py]
  - module-both       # [py or cpp]
  - module-both       # [py and cpp]
```

```
>>> yaml2dict("../tests/example.yaml")
{'package': None, 'modules': ['module']}
```

```
>>> yaml2dict("../tests/example.yaml", cpp=True)
{'package': {'name': 'name-cpp'}, 'modules': ['module', 'module-cpp']}
```

```
>>> yaml2dict("../tests/example.yaml", py=True)
{'package': {'name': 'name-py'}, 'modules': ['module', 'module-py']}
```

```
>>> yaml2dict("../tests/example.yaml", cpp=True, py=False)
{'package': {'name': 'name-cpp'}, 'modules': ['module', 'module-cpp', 'module-not-py
→']}
```

Returns yaml loaded into memory as Python dict, parsed for definitions.

Return type `dict`

MULTIDOC.TEMPLATE

`multidoc.template.get_docstring_template(**kwargs)`

Parameters `kwargs` (*dict[str, Any]*) – One of the supported `multidoc.template.__languages__` must be set to true in the ***kwargs*** in order to facilitate successful docstring deduction.

`multidoc.template.get_enum_member_template(**kwargs)`

Parameters `kwargs` (*dict[str, Any]*) – One of the supported `multidoc.template.__languages__` must be set to true in the ***kwargs*** in order to facilitate successful docstring deduction.

`multidoc.template.get_property_template(**kwargs)`

Parameters `kwargs` (*dict[str, Any]*) – One of the supported `multidoc.template.__languages__` must be set to true in the ***kwargs*** in order to facilitate successful docstring deduction.

MULTIDOC.GENERATE

`multidoc.generate.generate_docstring_header`(*api_declaration*, *destination*, *template_path*='/home/docs/checkouts/readthedocs.org/user_builds/multidoc/che

Parameters

- `api_declaration` (*APIDeclaration*) –
- `destination` –
- `template_path` –

MULTIDOC.REGEX

multidoc.regex.p_cpp_tag

```
p_cpp_tag = re.compile(
    r"(?P<indent>[^\S\r\n]+)?" # Matches named group indent from start of line.
    ↪until comment.
    r"\./!" # Matches ./! as user designed indentation of.
    ↪docstring.
    r"\s+" # Allows any number of whitespaces between start.
    ↪of comment and @.
    r"@get_docstring(" # Matches 'get_docstring(' exactly.
    r"(\s+)?" # Allows any number (zero include) of whitespaces.
    ↪from '( until 'class' or 'func' name.
    r"(" # Start of (optional) class regex group.
    r"(?P<cls>[\w]+)" # Match a named group 'class' for a word of any.
    ↪length.
    r"\." # . match for separation between class name.
    ↪and method name.
    r"(?P<method>[\w]+)" # Match a named group 'method' for a word of any.
    ↪length.
    r")?" # End of class regex group.
    r"(?P<func>\w+)?" # Match (optional) named group 'func' for a word.
    ↪of any length.
    r"(" # Start of overload matching group.
    r"(\s+)?" # Any number of whitespaces before comma.
    r"," # Matches comma separating cls.method/func.
    ↪from overload variable.
    r"(\s+)?" # Any number (optional) of whitespaces after.
    ↪comma.
    r"(overload(\s+)?=(\s+)?" # matches optional format of overload with.
    ↪overload key "overload = n"
    r")?"
    r"(?P<variant>[0-9]+)" # Named group 'variant', int value between 0 and 9.
    ↪([0-9]), unlimited times (+)
    r")?"
    r"(\s+)?" # Optional whitespaces between closing parenthesis.
    r"\)",
    flags=re.MULTILINE)
```

Tip: See <https://regex101.com/r/XJNQ8S/1> for desired effect.

Type `re.compile()`

`multidoc.regex.p_python_tag`

```
p_python_tag = re.compile(
    r"(?P<indent>[^\S\r\n]+)?" # Matches named group indent from start of line.
    ↪until comment.
    r"#" # Matches # as user designed indentation of.
    ↪docstring.
    r"\s+" # Allows any number of whitespaces between start.
    ↪of comment and @.
    r"@get_docstring(" # Matches '@get_docstring(' exactly.
    r"(\s+)?" # Allows any number (zero include) of whitespaces.
    ↪from '( until 'class' or 'func' name.
    r"(" # Start of (optional) class regex group.
    r"(?P<cls>[\w]+)" # Match a named group 'class' for a word of any.
    ↪length.
    r"\." # . match for separation between class name.
    ↪and method name.
    r"(?P<method>[\w]+)" # Match a named group 'method' for a word of any.
    ↪length.
    r")?" # End of class regex group.
    r"(?P<func>\w+)?" # Match (optional) named group 'func' for a word.
    ↪of any length.
    r"(" # Start of overload matching group.
    r"(\s+)?" # Any number of whitespaces before comma.
    r"," # Matches comma separating cls.method/func.
    ↪from overload variable.
    r"(\s+)?" # Any number (optional) of whitespaces after.
    ↪comma.
    r"(overload(\s+)?=(\s+)?" # matches optional format of overload with.
    ↪overload key "overload = n"
    r")?"
    r"(?P<variant>[0-9]+)" # Named group 'variant', int value between 0 and 9.
    ↪([0-9]), unlimited times (+)
    r")?"
    r"(\s+)?" # Optional whitespaces between closing parenthesis.
    r"\)",
    flags=re.MULTILINE)
```

Type `re.compile()`

`multidoc.regex.p_api_tag`

```
p_api_tag = re.compile(r".*#\s*\[(?P<expr>.*)\]")
```

Type `re.compile()`

`multidoc.regex.p_package_file`

```
p_api_tag = re.compile(r".*__package__\.yaml|.yml)")
```

Type `re.compile()`

multidoc.regex.p_module_file

```
p_module_file = re.compile(r".*(?P<module>\w+)\.(yaml|.yml)")
```

Type `re.compile()`

MULTIDOC.ERROR

exception multidoc.error.ClassNotDeclaredError

Class not declared in API Declaration.

exception multidoc.error.FunctionNotDeclaredError

Function not declared in API Declaration.

exception multidoc.error.MethodNotDeclaredError

Method not declared in API Declaration.

Examples

Listing 1: example-1.yaml

```
classes:
- name: ExampleClass
  methods:
  - name: constructor
  - name: foo_method
  # there is no bar_method !
```

Listing 2: ExampleClass.hpp

```
#include <string>

//! @get_docstring(ExampleClass.__docstring__)
class ExampleClass {
public:

    //! @get_docstring(ExampleClass.constructor)
    ExampleClass();

    //! @get_docstring(ExampleClass.foo_method)
    std::string foo_method();

    //! @get_docstring(ExampleClass.bar_method)
    std::string bar_method();

}
```

```
>>> from multidoc.parsing import parse_api_declaration
>>> from multidoc.testing import TESTING_DIR
>>> import os
>>> s = parse_api_declaration(os.path.join(TESTING_DIR, "example-1.yaml"))
```

exception `multidoc.error.OverloadNotFoundError`

Overload not declared in API Declaration.

INDICES AND TABLES

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

PYTHON MODULE INDEX

m

- `multidoc.error`, [19](#)
- `multidoc.generate`, [13](#)
- `multidoc.parsing`, [4](#)
- `multidoc.regex`, [15](#)
- `multidoc.template`, [11](#)

C

Class (*class in multidoc.parsing*), 4
 classes (*multidoc.parsing.Module attribute*), 7
 classes (*multidoc.parsing.Package attribute*), 8
 ClassNotDeclaredError, 19
 Config (*class in multidoc.parsing*), 5
 config (*multidoc.parsing.Module attribute*), 7
 config (*multidoc.parsing.Package attribute*), 8
 Constant (*class in multidoc.parsing*), 5
 constants (*multidoc.parsing.Module attribute*), 7
 constants (*multidoc.parsing.Package attribute*), 8

D

deprecation_warning (*multidoc.parsing.Class attribute*), 4
 deprecation_warning (*multidoc.parsing.Function attribute*), 6
 description (*multidoc.parsing.Parameter attribute*), 9
 description (*multidoc.parsing>Returns attribute*), 9

E

examples (*multidoc.parsing.Class attribute*), 5
 examples (*multidoc.parsing.Constant attribute*), 5
 examples (*multidoc.parsing.Function attribute*), 7
 examples (*multidoc.parsing.Module attribute*), 7
 examples (*multidoc.parsing.Package attribute*), 8
 extended_summary (*multidoc.parsing.Class attribute*), 4
 extended_summary (*multidoc.parsing.Constant attribute*), 5
 extended_summary (*multidoc.parsing.Function attribute*), 6
 extended_summary (*multidoc.parsing.Module attribute*), 7
 extended_summary (*multidoc.parsing.Package attribute*), 8

F

FileBased (*class in multidoc.parsing*), 5
 Function (*class in multidoc.parsing*), 5
 FunctionNotDeclaredError, 19
 functions (*multidoc.parsing.Module attribute*), 7

functions (*multidoc.parsing.Package attribute*), 8

G

generate_docstring_header() (*in module multidoc.generate*), 13
 get_docstring_template() (*in module multidoc.template*), 11
 get_enum_member_template() (*in module multidoc.template*), 11
 get_property_template() (*in module multidoc.template*), 11

M

MethodNotDeclaredError, 19
 methods (*multidoc.parsing.Class attribute*), 5
 module
 multidoc.error, 19
 multidoc.generate, 13
 multidoc.parsing, 4
 multidoc.regex, 15
 multidoc.template, 11
 Module (*class in multidoc.parsing*), 7
 modules (*multidoc.parsing.Package attribute*), 8
 multidoc.error
 module, 19
 multidoc.generate
 module, 13
 multidoc.parsing
 module, 4
 multidoc.regex
 module, 15
 multidoc.template
 module, 11

N

name (*multidoc.parsing.Class attribute*), 4
 name (*multidoc.parsing.Config attribute*), 5
 name (*multidoc.parsing.Function attribute*), 6
 name (*multidoc.parsing.Parameter attribute*), 8
 name (*multidoc.parsing>Returns attribute*), 9
 notes (*multidoc.parsing.Class attribute*), 5
 notes (*multidoc.parsing.Function attribute*), 6

notes (*multidoc.parsing.Module* attribute), 7
 notes (*multidoc.parsing.Package* attribute), 8

O

other_parameters (*multidoc.parsing.Class* attribute), 4
 other_parameters (*multidoc.parsing.Function* attribute), 6
 OverloadNotFoundError, 20

P

p_api_tag (*in module multidoc.regex*), 16
 p_cpp_tag (*in module multidoc.regex*), 15
 p_module_file (*in module multidoc.regex*), 17
 p_package_file (*in module multidoc.regex*), 16
 p_python_tag (*in module multidoc.regex*), 16
 Package (*class in multidoc.parsing*), 7
 Parameter (*class in multidoc.parsing*), 8
 parameters (*multidoc.parsing.Class* attribute), 4
 parameters (*multidoc.parsing.Function* attribute), 6
 parse_api_declaration() (*in module multidoc.parsing*), 9
 parse_yaml() (*multidoc.parsing.FileBased* class method), 5

R

raises (*multidoc.parsing.Class* attribute), 4
 raises (*multidoc.parsing.Function* attribute), 6
 references (*multidoc.parsing.Class* attribute), 5
 references (*multidoc.parsing.Constant* attribute), 5
 references (*multidoc.parsing.Function* attribute), 6
 references (*multidoc.parsing.Module* attribute), 7
 references (*multidoc.parsing.Package* attribute), 8
 Returns (*class in multidoc.parsing*), 9
 returns (*multidoc.parsing.Class* attribute), 4
 returns (*multidoc.parsing.Function* attribute), 6
 routine_listings (*multidoc.parsing.Module* attribute), 7
 routine_listings (*multidoc.parsing.Package* attribute), 8

S

see_also (*multidoc.parsing.Class* attribute), 4
 see_also (*multidoc.parsing.Constant* attribute), 5
 see_also (*multidoc.parsing.Function* attribute), 6
 see_also (*multidoc.parsing.Module* attribute), 7
 see_also (*multidoc.parsing.Package* attribute), 8
 short_summary (*multidoc.parsing.Class* attribute), 4
 short_summary (*multidoc.parsing.Function* attribute), 6
 summary (*multidoc.parsing.Constant* attribute), 5
 summary (*multidoc.parsing.Module* attribute), 7
 summary (*multidoc.parsing.Package* attribute), 8

T

type (*multidoc.parsing.Parameter* attribute), 8
 type (*multidoc.parsing>Returns* attribute), 9

V

version (*multidoc.parsing.Config* attribute), 5

W

warnings (*multidoc.parsing.Class* attribute), 4
 warnings (*multidoc.parsing.Function* attribute), 6
 warns (*multidoc.parsing.Class* attribute), 4
 warns (*multidoc.parsing.Function* attribute), 6

Y

yaml2dict() (*in module multidoc.parsing*), 9
 yields (*multidoc.parsing.Class* attribute), 4
 yields (*multidoc.parsing.Function* attribute), 6