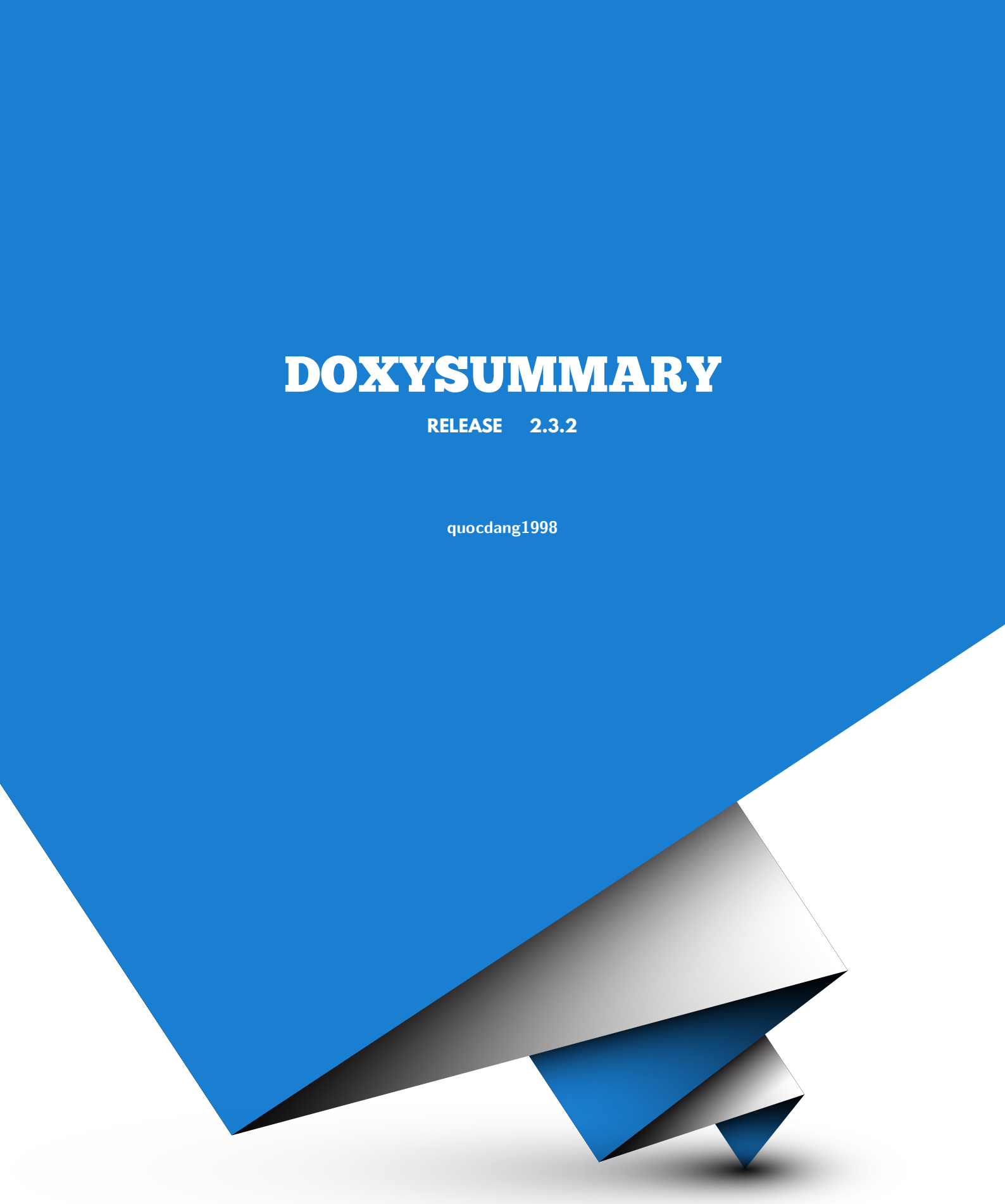
DOXYSUMMARY

RELEASE 2.3.2

quocdang1998



April, 2024

CONTENTS

1

INTRODUCTION

Doxysummary is a Sphinx extension for creating autosummary with entries from xml files generated by Doxygen.

2

INSTALLATION

Install from source:

```
$ git clone https://github.com/quocdang1998/doxysummary.git
$ cd doxysummary
$ pip install .
```

Run example:

```
$ pip install --upgrade sphinx-rtd-theme
$ cd example
$ doxygen Doxyfile
$ make html
```

3

USER GUIDE

1. In **conf.py**, add **sphinx_doxysummary** to the list of extensions

```
extensions = [...
    'sphinx_doxysummary',
    ...
]
```

2. In **conf.py**, set the config variable **doxygen_xml** to list of parent directories containing xml files, separated by ","

```
doxygen_xml = ['./xml1', '../project2/xml']  # each directory corresponds to one Doxygen project
```

3. In the input rst file, add the following directive:

```
.. doxysummary::
   :toctree: generated

   spam::Foo
   Bar
```

3.1 Configuration

DoxySummary provides these following config variables:

doxygen_xml (mandatory)
: Paths to Doxygen XML directories. Each directory is a project.

doxysummary_generate
: Automatically generate rst source files based on template. Default: **True**.

3.2 Alias

DoxySummary allows users to replace the display name of an entry in the summary table through aliasing.

```
.. doxysummary::
   :toctree: generated

   an_object_with_a_very_long_name "short_name"
```

Note: Alias should not have any space in between.

Note: Putting a ~ before an entry helps removing the scope name. However, this feature is overridden by the alias.

3.3 Change template

Users can write their own template for generated files with doxysummary (similar to the option in autosummary).

```
.. doxysummary::
   :toctree: generated
   :template: mycpp.rst

   Foo
```

Warning: Recursively auto-generating rst source files is not supported. Having directive **doxysummary** inside of template file will return error.

3.4 Scope

For multiple objects belong to the same scope (like namespace, class, scoped enum), the option scope can help to shorten the typed name.

```
.. doxysummary::
   :toctree: generated
   :scope: mynamespace

   MyClass
   my_function
   my_variable
```

Note: The display name in the summary table is the fullscope name. To display only the non-scoped name, use aslias or ~ instead.

3.5 Function Overloading

C++ allows many functions with different argument types to share the same name. If a function is overloaded, its argument type (with or without argument name) must be declared.

```
.. doxysummary::
   :toctree: generated

   my_function(int)
   my_function(char, char)
   my_function(std::vector<double> &)
```

4

PACKAGES

The process of creating doxysummary table can be divided into 3 stages:

1. `Generate XML tree` (when Sphinx initialize Builder instance)
2. `Generate rst files` (when Sphinx initialize Builder instance)
3. `Processing directives` (when Sphinx read the directive `.. doxysummary::`)

4.1 Generate XML tree

The module **`sphinx_doxysummary.xmltree`** is used to create a map from item names to their corresponding Doxygen descriptions. The result is saved to the module variable **`xml_tree`**, which is called when Sphinx parses rst files. When the summary table is constructed, summaries of the items are retrieved from their Doxygen descriptions stored in the variable.

Note that because function overloading is allowed in C++, the generated map is one-to-many (i.e. a name is mapped to a list of all possible descriptions sharing the same name).

`DoxygenItem`	Item read from Doxygen generated xml.
`process_generate_xmltree`	Create a tree of name -> **`DoxygenItem`**.
`xml_tree`	Map of item names to a list of corresponding DoxygenItem objects.

A. DoxygenItem

```
class sphinx_doxysummary.xmltree.DoxygenItem(refid: str, name: str, kind: str)
```

Item read from Doxygen generated xml.

`name`

Name of the item.

Type
str

`kind`

Kind of the item.

Type
str

`refid`

Reference ID of the item.

Type
str

summary

Brief description of the item.

Type
str

args

Arguments of a function in the form of a list of pairs (argtype, argname).

Type
List[Tuple[str, str]]

return_type

Return type of the function.

Type
str

overload

Wether function is overloaded by another function or not.

Type
bool

__init__(refid: str, name: str, kind: str)

Parameters
- **refid** (**str**) – Reference ID of the item within the xml document.
- **name** (**str**) – Name of the item (full scope).
- **kind** (**str**) – Kind of the item (one of the following values: class, struct, function, variable, enum, enumvalue, typedef, define, namespace, file).

check_args(args: str) → bool

Check if arguments of a prototype / declaration match the arguments of the item.

Parameters
args (**str**) – Arguments of the prototype / declaration in form of a string and enclosed in parentheses.

Examples

```
>>> d = DoxygenItem(refid, name, kind)
>>> ...  # set summary, args and return type
>>> d.check_args('(char a, char b)')
>>> d.check_args('(double *, int)')
```

property has_summary: bool

Check if the item has a summary or not.

set_args(args: List[Tuple[str, str]])

Set arguments of the item if the item is a function.

Parameters
args (**List[Tuple[str, str]]**) – Arguments of the function.

Raises
ValueError – When the item is not a function.

```
set_return_type(return_type: str)
```

Set return type of the item if the item is a function.

Parameters
return_type (**str**) – Return type of the function.

Raises
ValueError – When the item is not a function.

```
set_summary(summary: str)
```

Set summary of the item.

Parameters
summary (**str**) – Brief description of the item.

B. process_generate_xmltree

```
sphinx_doxysummary.xmltree.process_generate_xmltree(app: Sphinx) → None
```

Create a tree of name -> **DoxygenItem**.

This process parses through all xml files in all Doxygen projects which are declared in the config variable **doxygen_xml**, and creates a look-up table (i.e. a map of item full scope name to its corresponding DexygenItem objects) stored in the extern variable **xml_tree**.

Parameters
app (**Sphinx**) – Sphinx.Builder.

Raises
ValueError – When the behavior of Doxygen-created XML elements are not as expected.

Notes

- This process should be executed at the initialization of the building process of Sphinx.
- The name saved in **xml_tree** is the full scope name of item.

C. xml_tree

```
sphinx_doxysummary.xmltree.xml_tree = {}
```

Map of item names to a list of corresponding DoxygenItem objects.

4.2 Generate rst files

The process of auto-generating rst files which is based on the provided template, is executed after parsing XML data.

Along with `process_generate_xmltree`, this process is called at the beginning of building the documentation (not when Sphinx is parsing rst source files) to avoid internal linking problems.

DoxySummaryEntry	Simple class representing an entry in "doxysummary" directive.
DoxySummaryRenderer	Renderer generating rst files based on templates.
process_generate_files	Process generating rst files.

A. DoxySummaryEntry

```
class sphinx_doxysummary.generate.DoxySummaryEntry(filename: str, name: str, template: str =
'cppbase.rst', toctree: str = '', scope: str = '', alias: str | None = None)
```

Simple class representing an entry in “doxysummary” directive.

filename

Name of rst file source.

Type
str

toctree

Directory in which the rst file will be generated.

Type
str

template

Name of the template for generating rst file. Default is the template **cppbase.rst** in installation directory of this package.

Type
str

name

Content of the entry. If the entry is a function, the return type is removed. Only the function name and the arguments are retained.

Type
str

scope

Scope of the name. The true name in Doxygen XML of the item is “scope::name”.

Type
str

alias

Alias of the entry. If there is an alias, the displayname and the title of the generated file are the alias.

Type
str

```
__init__(filename: str, name: str, template: str = 'cppbase.rst', toctree: str = '',
scope: str = '', alias: str | None = None)
```

Parameters

- **filename** (**str**) – Name of the rst file containing the entry.
- **template** (**str**) – Name of template for generating the entry.
- **name** (**str**) – Content of the entry.
- **toctree** (**str, optional**) – Directory to generate automatically rst files. The default is ‘’.
- **scope** (**str, optional**) – Current scope of the entry. The default is ‘’.
- **alias** (**str**) – Alias of the entry. The default is **None**.

```
property fullname: str
```

Get the fullname (scope + name) of the entry.

Returns
Fullname of the entry.
Return type
str

B. DoxySummaryRenderer

```
class sphinx_doxysummary.generate.DoxySummaryRenderer(app: Sphinx)
```

Renderer generating rst files based on templates.

```
env
```

Enviroment containing path to possible templates to match.

Type
jinja2.sandbox.SandboxedEnvironment

```
__init__(app: Sphinx) → None
```

Parameters
app (**Sphinx**) – Sphinx application.
Raises
ValueError – When **app** is not a sphinx.Builder.

```
render(template_name: str, context: Dict[str, Any]) → str
```

Render a string based on a template.

The renderer will first find template in source directory, then in the template path declared in **conf.py**, and finally in the template directory of this package.

Parameters
- **template_name** (**str**) – File name of the template.
- **context** (**Dict[str, Any]**) – Python **dict** of keyword-value to be fetched in the template.

Returns
Content of the template with matched keywords.
Return type
str

C. process_generate_files

```
sphinx_doxysummary.generate.process_generate_files(app: Sphinx) → None
```

Process generating rst files. This function must be called at initialization of Sphinx's building process.

Parameters
app (**Sphinx**) – Sphinx Buider.
Raises
ValueError – Kind of item not found in the package template library.

4.3 Processing directives

The third step is to instruct Sphinx how to process the rst directives **doxysummary** and translate it to html/latex/... output.

DoxySummary	Class represents the directive **doxysummary** when Sphinx parses inputs.

A. DoxySummary

```
class sphinx_doxysummary.directive.DoxySummary(name, arguments, options, content, lineno,
content_offset, block_text, state, state_machine)
```

Class represents the directive **doxysummary** when Sphinx parses inputs.

```
final_argument_whitespace = False
```

May the final argument contain whitespace?

```
has_content = True
```

May the directive have content?

```
option_spec: dict[str, Callable[[str], Any]] = {'scope': <function unchanged>,
'template': <function unchanged_required>, 'toctree': <function unchanged>}
```

Dictionary of options of the directive.

```
optional_arguments = 0
```

Number of optional arguments after the required arguments.

```
required_arguments = 0
```

Number of required directive arguments.

```
run() → List[Node]
```

Method called after Sphinx has read the directive **doxysummary**.

It retrieves the full name of each entry, creates a summary table with alias and brief description from Doxygen created XML files, and builds a hidden toctree linking to the generated files at the beginning of Sphinx build process.

Raises

ValueError – When a line having more than 1 alias.

Returns

List of docutils nodes to be added to the document.

Return type

List[docutils.nodes.Node]

Notes

The **name** variable in the method **run** represents the full scoped name without return type and with arguments.

4.4 Utils

This module contains util functions.

tokenize_arg	Split a C++ argument into its components.
compare_type	Check if type of 2 arguments are the same.
getFisrtChildByTagName	Search the first child of XML element by tagname.
split_name	Split any item declaration into a list of return type, item name and arguments.
fullname_to_filename	Convert special characters in item fullname to valid filename characters.

A. tokenize_arg

```
sphinx_doxysummary.utils.tokenize_arg(argument: str) → List[Set[str]]
```

Split a C++ argument into its components.

Parameters
argument (**str**) – One argument of the function.
Returns
List of set of tokens from the argument.
Return type
List[Set[str]]

Examples

```
>>> tokenize_arg('int argc')  # type + var-name
[{'argc', 'int'}]
>>> tokenize_arg('const char *a')  # with specifier
[{'char'}, {'*'}, {'a'}]
>>> tokenize_arg('char&& a')  # reference to r-value
[{'char'}, {'&'}, {'&'}, {'a'}]
>>> tokenize_arg('const std::vector< double *, int > & x_')  # template
[{'const', 'std::vector<double *, int>'}, {'&'}, {'x_'}]
```

B. compare_type

```
sphinx_doxysummary.utils.compare_type(arg1: str, arg2: str) → bool
```

Check if type of 2 arguments are the same.

Parameters
- **arg1** (**str**) – First argument.
- **arg2** (**str**) – Second argument.

Returns
True if 2 arguments have the same type.

Return type
bool

Examples

```
>>> compare_type('const int a', 'int const')  # normal case
True
>>> compare_type('const int * a', 'int *')  # without specifier
False
>>> compare_type('const int &', 'int const &')  # change specifier position
True
>>> compare_type('std::vector<const double *> &', 'std::vector<double const> &')  # template
False
>>> compare_type('void', '')
True
```

C. getFisrtChildByTagName

```
sphinx_doxysummary.utils.getFisrtChildByTagName(element: _Element, tag: str) →
List[_Element]
```

Search the first child of XML element by tagname.

Parameters
- **element** (**etree._Element**) – XML element to search.
- **tag** (**str**) – Name of the tag to search.

Returns
result – List of first-level children with tagname to search for.

Return type
List[lxml.etree._Element]

D. split_name

```
sphinx_doxysummary.utils.split_name(name: str) → List[str]
```

Split any item declaration into a list of return type, item name and arguments.

Parameters
name (**str**) – Declarartion of the item.

Returns
List of return type - item name - arguments

Return type
List[str]

Examples

```
>>> split_name('spam::Shrub')
['', 'spam::Shrub', '']
>>> split_name('void hello::hello_world(int a, const std::vector<int> & b)')
['void', 'hello::hello_world', '(int a, const std::vector<int> & b)']
>>> split_name('int * get(int * array)')
['int *', 'get', '(int * array)']
>>> split_name('spam::Spam operator * (spam::Spam & x, spam::Spam & y)')
['spam::Spam', 'operator *', '(spam::Spam & x, spam::Spam & y)']
>>> split_name('void* spam::Spam::operator ->* ()')
['void *', 'spam::Spam::operator ->*', '()']
```

E. fullname_to_filename

sphinx_doxysummary.utils.fullname_to_filename(item_name: str, suffix: str)

Convert special characters in item fullname to valid filename characters.

Parameters

- **item_name** (**str**) – Full name of item in C++ code (can be declaration or prototype).
- **suffix** (**str**) – File suffix.

Returns

Name of the file.

Return type

str

5

LICENCE

```
MIT License

Copyright (c) 2022 quocdang1998

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

INDEX

Symbols

A

C

D

E

F

G

H

K

N

O

P

R

S

T

X