sphinx-argparse Documentation

Release 0.2.2

Alex Rudakov and Devon Ryan

Mar 15, 2018

Contents

sphinx-argparse is an extension for sphinx that allows for easy generation of documentation for command line tools using the python argparse library.

CHAPTER 1

Installation

This extension is tested on python 2.7 and 3.3+.

The package is available in the Python Package Index:

```
pip install sphinx-argparse
```

Enable the extension in your sphinx config:

```
extensions += ['sphinxarg.ext']
```

CHAPTER 2

Basic usage

This extension adds the “argparse” directive:

```
.. argparse::
   :module: my.module
   :func: my_func_that_returns_a_parser
   :prog: fancytool
```

The *module*, *func* and *prog* options are required.

func is a function that returns an instance of the *argparse.ArgumentParser* class.

Alternatively, one can use :ref: like this:

```
.. argparse::
   :ref: my.module.my_func_that_returns_a_parser
   :prog: fancytool
```

In this case :ref: points directly to argument parser instance.

For this directive to work, you should point it to the function that will return a pre-filled *ArgumentParser*. Something like:

```
def my_func_that_return_parser():
    parser = argparse.ArgumentParser()
    parser.add_argument('foo', default=False, help='foo help')
    parser.add_argument('bar', default=False)

    subparsers = parser.add_subparsers()

    subparser = subparsers.add_parser('install', help='install help')
    subparser.add_argument('ref', type=str, help='foo1 help')
    subparser.add_argument('--upgrade', action='store_true', default=False, help=
↪'foo2 help')

    return parser
```

Note: We will use this example as a reference for every example in this document.

To document a file that is not part of a module, use :filename:

```
.. argparse::
   :filename: script.py
   :func: my_func_that_returns_a_parser
   :prog: script.py
```

The 'filename' option could be absolute path or a relative path under current working dir.

:module: Module name, where the function is located

:func: Function name

:ref: A combination of :module: and :func:

:filename: A file name, in cases where the file to be documented is not part of a module.

:prog: The name of your tool (or how it should appear in the documentation). For example, if you run your script as *./boo –some args* then :prog: will be "boo"

That's it. Directives will render positional arguments, options and sub-commands.

Sub-commands are limited to one level. But, you can always output help for subcommands separately:

```
.. argparse::
   :module: my.module
   :func: my_func_that_return_parser
   :prog: fancytool
   :path: install
```

This will render same doc for "install" subcommand.

Nesting level is unlimited:

```
.. argparse::
   :module: my.module
   :func: my_func_that_return_parser
   :prog: fancytool
   :path: install subcomand1 subcommand2 subcommand3
```

2.1 Other useful directives

nodefault Do not show any default values.

nodefaultconst Like nodefault:, except it applies only to arguments of types *store_const*, *store_true* and *store_false*.

nosubcommands Do not show subcommands.

noepilogue Do not parse the epilogue, which can be useful if it contains text that could be incorrectly parse as reStructuredText.

nodescription Do not parse the description, which can be useful if it contains text that could be incorrectly parse as reStructuredText.

CHAPTER 3

Extending results of *argparse* directives

You can add extra content or even replace some parts of the documentation generated by the *argparse* directive. For example, any content you put inside directives (you must follow ReStructuredText identation rules) will be inserted just before the argument and option list:

```
.. argparse::
    :module: my.module
    :func: my_func_that_return_parser
    :prog: fancytool

    My content here that will be inserted right before the argument list.

    Also any valid markup...
    ************************

    ... may `be` *applied* here

    including::

         any directives you usually use.
```

Also, there is an option to insert custom content into a specific argument/option/subcommand/argument-group description. Just create a name:definition pair, where the name is an argument/option/subcommand/argument-group name and the definition is any reStructured markup. Changes to options/arguments appearing in multiple action groups can either be targeted (i.e., only one instance of the argument is changed) or general (i.e., all instances are modified).:

```
.. argparse::
    :module: my.module
    :func: my_func_that_return_parser
    :prog: fancytool

    My content here that will be inserted right before the argument list.

    foo
         This text will go right after the "foo" positional argument help.
```

(continues on next page)

(continued from previous page)

```
    install
         This text will go right after the "install" subcommand help and before its␣
↪arguments.

         --upgrade -u
             This text will go after the upgrade option of the install subcommand.␣
↪Nesting is unlimited.
             Note the space between --upgrade and -u, which differs from the comma␣
↪that would normally
             be used.

     --output -o
         Content appended to the --output option, regardless of the argument group.
```

You can also add classifiers, which will change how these definitions are incorporated:

```
.. argparse::
    :module: my.module
    :func: my_func_that_return_parser
    :prog: fancytool

    My content that will be inserted right before the argument list.

    foo : @before
         This text will go before the "foo" positional argument help.

    install : @replace
         This text will replace the "install" subcommand help/description.

         --upgrade : @after
             The after directive is the default, so you needn't specify it.
```

@before Insert content before the parsed help/description message of the argument/option/subcommand/argument-group.

@after Insert content after the parsed help/description message of argument/option/subcommand/argument-group. This is the default.

@replace Replace content of help/description message of argument/option/subcommand/argument-group.

CHAPTER 4

Examples

4.1 Example documentation structure

Here is an example structure for the documentation of a complex command with many subcommands. You are free to use any structure, but this may be a good starting point.

File “index.rst”:

```
.. toctree::
   :maxdepth: 2

   cmd
```

File “cmd.rst”:

```
Command line utilities
**********************

.. toctree::
   :maxdepth: 1

   cmd_main
   cmd_subcommand
```

File “cmd_main.rst”:

```
Fancytool command
*****************

.. argparse::
   :module: my.module
   :func: my_func_that_returns_a_parser
   :prog: fancytool
```

(continues on next page)

(continued from previous page)

```
   subcommand
        Here we add a reference to subcommand, to simplify navigation.
        See :doc:`cmd_subcommand`
```

File “cmd_subcommand.rst”:

```
Subcommand command
**********************

.. argparse::
   :module: my.module
   :func: my_func_that_return_parser
   :prog: fancytool
   :path: subcommand
```

4.2 Source of example file

This file will be used in all generated examples.

```
import argparse

parser = argparse.ArgumentParser()

subparsers = parser.add_subparsers()

my_command1 = subparsers.add_parser('apply', help='Execute provision script, collect␣
↪all resources and apply them.')

my_command1.add_argument('path', help='Specify path to provision script. provision.py␣
↪in current'
                                      'directory by default. Also may include url.',␣
↪default='provision.py')
my_command1.add_argument('-r', '--rollback', action='store_true', default=False, help=
↪'If specified will rollback all'

↪'resources applied.')
my_command1.add_argument('--tree', action='store_true', default=False, help='Print␣
↪resource tree')
my_command1.add_argument('--dry', action='store_true', default=False, help='Just␣
↪print changes list')
my_command1.add_argument('--force', action='store_true', default=False, help='Apply␣
↪without confirmation')
my_command1.add_argument('default_string', default='I am a default', help='Ensure␣
↪variables are filled in %(prog)s (default %(default)s)')

my_command2 = subparsers.add_parser('game', help='Decision games')
my_command2.add_argument('move', choices=['rock', 'paper', 'scissors'], help='Choices␣
↪for argument example')
my_command2.add_argument('--opt', choices=['rock', 'paper', 'scissors'], help=
↪'Choices for option example')

optional = my_command2.add_argument_group('Group 1')

optional.add_argument('--addition', choices=['Spock', 'lizard'], help='Extra choices␣
↪for additional group.')
```

(continues on next page)

(continued from previous page)

```
optional.add_argument('--lorem_ipsum',
                      help='Lorem ipsum dolor sit amet, consectetur adipiscing elit,␣
↪sed do eiusmod '
                           'tempor incididunt ut labore et dolore magna aliqua. Ut␣
↪enim ad minim veniam, '
                           'quis nostrud exercitation ullamco laboris nisi ut aliquip␣
↪ex ea commodo '
                           'consequat. Duis aute irure dolor in reprehenderit in␣
↪voluptate velit esse '
                           'cillum dolore eu fugiat nulla pariatur. Excepteur sint␣
↪occaecat cupidatat '
                           'non proident, sunt in culpa qui officia deserunt mollit␣
↪anim id est laborum.')
```

4.3 Generated sample 1 - command with subcommands

4.3.1 Directive

Source:

```
.. argparse::
   :filename: ../test/sample.py
   :func: parser
   :prog: sample
```

4.3.2 Output

```
usage: sample [-h] {apply,game} ...
```

Sub-commands:

apply

Execute provision script, collect all resources and apply them.

```
sample apply [-h] [-r] [--tree] [--dry] [--force] path default_string
```

Positional Arguments

path Specify path to provision script. provision.py in currentdirectory by default. Also may include url.

Default: “provision.py”

default_string Ensure variables are filled in (default “I am a default”)

Default: “I am a default”

Named Arguments

-r, --rollback If specified will rollback allresources applied.

Default: False

--tree Print resource tree

Default: False

--dry Just print changes list

Default: False

--force Apply without confirmation

Default: False

game

Decision games

```
sample game [-h] [--opt {rock,paper,scissors}] [--addition {Spock,lizard}]
            [--lorem_ipsum LOREM_IPSUM]
            {rock,paper,scissors}
```

Positional Arguments

move Possible choices: rock, paper, scissors

Choices for argument example

Named Arguments

--opt Possible choices: rock, paper, scissors

Choices for option example

Group 1

--addition Possible choices: Spock, lizard

Extra choices for additional group.

--lorem_ipsum Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

4.4 Generated sample 2 - subcommand

4.4.1 Directive

Source:

```
.. argparse::
   :filename../test/sample.py
   :func: parser
   :prog: sample
   :path: game
```

4.4.2 Output

```
usage: sample game [-h] [--opt {rock,paper,scissors}]
                   [--addition {Spock,lizard}] [--lorem_ipsum LOREM_IPSUM]
                   {rock,paper,scissors}
```

Positional Arguments

move Possible choices: rock, paper, scissors

Choices for argument example

Named Arguments

--opt Possible choices: rock, paper, scissors

Choices for option example

Group 1

--addition Possible choices: Spock, lizard

Extra choices for additional group.

--lorem_ipsum Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

CHAPTER 5

Miscellaneous

5.1 Text wrapping in argument tables

A common issue with the default html output is that the table within which options are displayed is designed such that the option descriptions are each held on one line. Any even remotely lengthy description then causes the viewer to need to scroll left/right to view the entire text. This is typically undesirable and the fix is described fully here.

The short synopsis is below:

1. Create a new CSS file (likely under *_static*) and point to it in *html_static_path* and *html_context* (or a template in the *templates_path*) in *conf.py*.
2. In that CSS file, add the following code:

```
.wy-table-responsive table td {
    white-space: normal !important;
}
.wy-table-responsive {
    overflow: visible !important;
}
```

5.2 Linking to action groups

As of version 0.2.0, action groups (e.g., "Optional arguments", "Required arguments", and subcommands) can be included in tables of contents and external links. The anchor name is the same as the title name (e.g., "Optional arguments"). In cases where titles are duplicated, as is often the case when subcommands are used, *_repeatX*, where *X* is a number, is prepended to duplicate anchor names to ensure that they can all be uniquely linked.

CHAPTER 6

Markdown

As of version 0.2.0, markdown (rather than only reStructuredText) can be included inside directives as nested content. While markdown is much easier to write, please note that it is also less powerful. An example is below:

```
.. argparse::
    :filename: ../test/sample.py
    :func: parser
    :prog: sample
    :markdown:

    Header 1
    ========

    [I'm a link to google](http://www.google.com)

    ## Sub-heading

    ```
    This
     is
      a
      fenced
     code
    block
    ```
```

The above example renders as follows:

A random paragraph

6.1 Heading 1

I'm a link to google

6.1.1 Sub heading

```
This
 is
  a
  fenced
 code
block
```

```
usage: sample [-h] {apply,game} ...
```

6.2 Sub-commands:

6.2.1 apply

Execute provision script, collect all resources and apply them.

```
sample apply [-h] [-r] [--tree] [--dry] [--force] path default_string
```

Positional Arguments

path Specify path to provision script. provision.py in currentdirectory by default. Also may include url.

Default: "provision.py"

default_string Ensure variables are filled in (default "I am a default")

Default: "I am a default"

Named Arguments

-r, --rollback If specified will rollback allresources applied.

Default: False

--tree Print resource tree

Default: False

--dry Just print changes list

Default: False

--force Apply without confirmation

Default: False

6.2.2 game

Decision games

```
sample game [-h] [--opt {rock,paper,scissors}] [--addition {Spock,lizard}]
            [--lorem_ipsum LOREM_IPSUM]
            {rock,paper,scissors}
```

Positional Arguments

move Possible choices: rock, paper, scissors

Choices for argument example

Named Arguments

--opt Possible choices: rock, paper, scissors

Choices for option example

Group 1

--addition Possible choices: Spock, lizard

Extra choices for additional group.

--lorem_ipsum Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

The CommonMark-py is used internally to parse Markdown. Consequently, only Markdown supported by CommonMark-py will be rendered.

You must explicitly use the *:markdown:* flag, otherwise all content inside directives will be parsed as reStructuredText.

6.3 A note on headers

If the Markdown you nest includes headings, then the first one **MUST** be level 1. Subsequent headings can be at lower levels and then rendered correctly.

6.4 Hard line breaks

Sphinx strips white-space from the end of lines prior to handing it to this package. Because of that, hard line breaks can not currently be rendered.

6.5 Replacing/appending/prepending content

When markdown is used as nested content, it's not possible to create dictionary entries like in reStructuredText to modify program option descriptions. This is because CommonMark-py does not support dictionary entries.

6.6 MarkDown in program descriptions and option help

In addition to using MarkDown in nested content, one can also use MarkDown directly in program descriptions and option help messages. For example:

```
import argparse

def blah():
    parser = argparse.ArgumentParser(description="""
### Example of MarkDown inside programs

[I'm a link](http://www.google.com)
""")
    parser.add_argument('cmd', help='execute a `command`')
    return parser
```

To render this as MarkDown rather than reStructuredText, use the *markdownhelp* option:

```
.. argparse::
    :filename: ../test/sample2.py
    :func: blah
    :prog: sample
    :markdownhelp:
```

This will then be rendered as:

6.7 Example of MarkDown inside programs

I'm a link

```
usage: sample [-h] cmd
```

6.7.1 Positional Arguments

cmd execute a `command`

CHAPTER 7

Change log

7.1 0.2.2

- CommonMark is now only imported if absolutely required. This should fix failures on read the docs. Thanks to @Chilipp for fixing this!

7.2 0.2.1

- Stopped importing *sphinx.util.compat*, which was causing issues like that seen in #65

7.3 0.2.0

- Section titles can now be used in tables of contents and linked to. The title itself is also used as the anchor. In the case of repeated names *_replicateX*, where *X* is a number, is prepended to ensure that all titles are uniquely linkable. This was bug #46.
- The positional (aka required) and named (aka optional) option sections are now named "Positional Arguments" and "Named Arguments", for the sake of clarity (e.g., named arguments can be required). This was issue #58.
- Fixed quoting of default strings (issue #59).
- Added the *:noepilogue:* and *:nodescription:* options, thanks to @arewm.
- Added the *:nosubcommand:* option, thanks to @arewm.

7.4 0.1.17

- Fixed handling of argument groups (this was bug #49). Thanks to @croth1 for reporting this bug. Note that now position arguments (also known as required arguments) within argument groups are now also handled correctly.

7.5 0.1.16

- Added a *:nodefaultconst:* directive, which is similar to the *:nodefault:* directive, but applies only to *store_true*, *store_false*, and *store_const* (e.g., it will hide the "=True" part in the output, since that can be misleading to users).
- Fixed various typos (thanks to users mikeantonacci, brondsem, and tony)
- Format specifiers (e.g., *%(prog)s* and *%(default)s*) are now filled in (if possible) in help sections. If there's a missing keyword, then nothing will be filled in. This was issue #27.
- The package is now a bit more robust to incorrectly spelling module names (#39, courtesy of Gabriel Falcão)
- Added support for argparse groups (thanks to Fidel Ramirez)

7.6 0.1.15

- Fixed malformed docutils DOM in manpages (Matt Boyer)

7.7 0.1.14

- Support for aliasing arguments #22 (Campbell Barton)
- Support for nested arguments #23 (Campbell Barton)
- Support for subcommand descriptions #24 (Campbell Barton)
- Improved parsing of content of *epilog* and *description* #25 (Louis - https://github.com/paternal)
- Added 'passparser' option (David Hoese)

7.8 0.1.13

- Bugfix: Choices are not always strings (Robert Langlois)
- Polished small mistakes in usage documentation (Dean Malmgren)
- Started to improve man-pages support (Zygmunt Krynicki)

7.9 0.1.12

- Improved error reporting (James Anderson)

7.10 0.1.11

- Fixed stupid bug, prevented things working on py3 (Alex Rudakov)
- added tox configuration for tests

7.11 0.1.10

- Remove the ugly new line in the end of usage string (Vadim Markovtsev)
- Issue #9 Display argument choises (Proposed by Felix-neko, done by Alex Rudakov)
- **ref** syntax for specifying path to parser instance. Issue #7 (Proposed by David Cottrell, Implemented by Alex Rudakov)
- Updated docs to read the docs theme

7.12 0.1.9

Fix problem with python version comparison, when python reports it as "2.7.5+" (Alex Rudakov)

7.13 0.1.8

Argparse is not required anymore separate module as of python 2.7 (Mike Gleen)

7.14 0.1.7

– Nothing – Created by accident.

7.15 0.1.6

Adding :nodefault: directive that skips default values for options (Stephen Tridgell)

7.16 0.1.5

Fix issue: epilog is ignored (James Anderson - https://github.com/jamesra)

7.17 0.1.4

Fix issue #3: ==SUPPRESS== in option list with no default value

7.18 0.1.2

Fix issue with subcommands (by Tony Narlock - https://github.com/tony)

7.19 0.1.1

Initial version

CHAPTER 8

Contribute

Any help is welcome!

Most wanted:

- Additional features
- Bug fixes
- Examples

Contributions are gratefully accepted through github pull-request. Please report bugs as issues on github.

Don't forget to run tests before committing:

```
py.test
```

CHAPTER 9

References

9.1 Similar projects

- https://pythonhosted.org/sphinxcontrib-autoprogram/ (See for comparison: https://github.com/ribozz/sphinx-argparse/issues/16)