FriendlyShell Documentation

Release 1.0.9

Kevin S. Phillips

Jun 23, 2019

Contents:

1 friendlyshell **3**
1.1 friendlyshell package 3

2 Overview **9**

3 Indices and tables **11**

Python Module Index **13**

Index **15**

CHAPTER 1

friendlyshell

1.1 friendlyshell package

1.1.1 Submodules

friendlyshell.base_shell module

Common shell interaction logic shared between different shells

class friendlyshell.base_shell.**BaseShell**(**args*, ***kwargs*)
Bases: object

Common base class for all Friendly Shells

Defines basic IO and interactive shell logic common to all Friendly Shells

static alias_native_shell()
Gets the shorthand character for the 'native_shell' command

Return type str

static debug(*message*, **args*, ***_kwargs*)
Displays an internal-use-only debug message to verbose log file

Default implementation hides all debug output. Use a logging mixin class to customize this behavior.

See *friendlyshell.basic_logger_mixin.BasicLoggerMixin* for examples.

Parameters **message** (*str*) – text to be displayed

do_close()
Terminates the currently running shell

do_exit()
Terminates the command interpreter

do_native_shell(*cmd*)
Executes a shell command within the Friendly Shell environment

static error(*message*, **args*, ***_kwargs*)
Displays a critical error message to the default output stream

Default implementation just directs output to stdout. Use a logging mixin class to customize this behavior.

See *friendlyshell.basic_logger_mixin.BasicLoggerMixin* for examples.

Parameters **message** (*str*) – text to be displayed

static help_close()
Extended help for close method

static info(*message*, **args*, ***_kwargs*)
Displays an info message to the default output stream

Default implementation just directs output to stdout. Use a logging mixin class to customize this behavior.

See *friendlyshell.basic_logger_mixin.BasicLoggerMixin* for examples.

Parameters **message** (*str*) – text to be displayed

return_code
error / return code generated by operations run by this shell

Return type int

run(**_args*, ***kwargs*)
Main entry point function that launches our command line interpreter

This method will wait for input to be given via the command line, and process each command provided until a request to terminate the shell is given.

Parameters

- **input_stream** – optional Python input stream object where commands should be loaded from. Typically this will be a file-like object containing commands to be run, but any input stream object should work. If not provided, input will be read from stdin using `input()`
- **parent** – Optional parent shell which owns this shell. If none provided this shell is assumed to be a parent or first level shell session with no ancestry

run_subshell(*subshell*)
Launches a child process for another shell under this one

Parameters **subshell** – the new Friendly Shell to be launched

static warning(*message*, **args*, ***_kwargs*)
Displays a non-critical warning message to the default output stream

Default implementation just directs output to stdout. Use a logging mixin class to customize this behavior.

See *friendlyshell.basic_logger_mixin.BasicLoggerMixin* for examples.

param str message text to be displayed

friendlyshell.basic_logger_mixin module

Mixin for displaying output from a friendly shell

class friendlyshell.basic_logger_mixin.**BasicLoggerMixin**(**args*, ***kwargs*)
Bases: object

Mixin class, to be combined with a Friendly Shell, to direct log output to a default output stream

The assumption here is that all Friendly Shell derived classes are going to use the print(), warning() and error() methods on this class to interact with the shell, and those methods in turn will use the Python logging API to delegate output to. By default those methods should direct their output to stdout, however if a user has s need to redriect the output elsewhere - like, when running a shell in a non-interactive or headless environment - then they can easily do so by simply re-configuring the default logger for the library

Helpful links relating to logging

https://docs.python.org/2/library/logging.html#logrecord-attributes https://docs.python.org/2/library/logging.html#logging-levels

debug(*message*, **args*, ***kwargs*)
Displays an internal-use-only debug message to verbose log file

Parameters **message** (*str*) – text to be displayed

error(*message*, **args*, ***kwargs*)
Displays a critical error message to the default output stream

Parameters **message** (*str*) – text to be displayed

info(*message*, **args*, ***kwargs*)
Displays an info message to the default output stream

Parameters **message** (*str*) – text to be displayed

warning(*message*, **args*, ***kwargs*)
Displays a non-critical warning message to the default output stream

Parameters **message** (*str*) – text to be displayed

friendlyshell.basic_shell module

friendlyshell.command_complete_mixin module

Mixin class that adds command completion to a friendly shell

class friendlyshell.command_complete_mixin.**CommandCompleteMixin**(**args*, ***kwargs*)
Bases: object

Mixin to be added to any friendly shell to add command completion

do_clear_history()
Clears the history of previously used commands from this shell

friendlyshell.command_complete_mixin.**auto_complete_manager**(**args*, ***kwds*)
Context manager for enabling command line auto-completion

This context manager can be used to ensure that command completion for a Friendly Shell runner can have it's own self-defined auto-completion and command history while not affecting the global completion sub-system that may already be configured prior to running the Friendly Shell. Through the use of a context manager, we ensure the state of the command completion subsystem will be restored regardless of how the context manager was terminated.

Parameters

- **key** (*str*) – descriptor for keyboard key to use for auto completion trigger
- **callback** – method point for the callback to run when completion key is pressed
- **history_file** (*str*) – optional path to the history file to use for storing previous commands run by this shell. If not provided, history will not be saved.

friendlyshell.command_parsers module

Pre-defined command line parsers supported by Friendly Shell APIs

All Friendly Shell command lines are expected to begin with a command name, followed by 0 or more input parameters as shown below:

<command_token>[<parameter_tokens>]

The command portion of each line must map command names to Python class methods that are prefixed with the characters "do_" (ie: "do_exit()"). Because of this commands must meet the same naming restrictions as Python class methods. Thus most Friendly Shells line parsers will want to leverage the `meth` default_command_parser function to parse the first token of their command lines. Also, for convenience all command token parsers are expected to return a named token with the label 'command' associated with it.

The optional list of parameter tokens may satisfy whatever restrictions are desired by a particular command. Since they need not be mapped to Python class methods they do not need to be bound by the same naming rules as the command token. Finally, the list of input parameters - when provided - are expected to return a named token with the label 'params', which will define a list of 1 or more input parameters to be provided to the command.

Below you will find some helper functions that provide pre-built grammars for some common command and parameter token styles so users of the library need not always write their own.

`friendlyshell.command_parsers.`**`default_command_token`**`()`
token parser that satisfies the requirements of the FShell interpreter

Meets the default naming requirements of the Friendly Shell interpreter, ensuring that command names may be safely mapped to Python class methods. The resulting token will be labelled 'command' as required by the Friendly Shell APIs.

Return type `pyparsing.Parser`

`friendlyshell.command_parsers.`**`default_line_parser`**`()`
Gets the default command line parser used by the Friendly Shell APIs

Return type `pyparsing.Parser`

`friendlyshell.command_parsers.`**`quoted_space_sep_params_token`**`()`
Gets a token parser that supports

This parser expects all commands to take the following form: <command_name>[<optional_param1> <optional_param2>...]<eol>

The <command_name> maps to a method of this class or a descendent of it with the same name but with a prefix of "do_". As a result tokens for commands must adhere to the same restrictions as a typical Python class method.

Commands may optionally accept parameters if desired. Parameters are provided on the same line as the command and are separated by spaces. Each parameter may use any printable character since they are translated to Python strings during translation and thus need not be restricted to the same criteria as commands. Also, if a parameter needs to contain embedded spaces then it must be wrapped in quotation marks. Single or double quotes will work. Further, if you need to embed quotation marks within your string simply wrap the inner quotes with outer quotes of the opposite style.

The command token can be accessed by named attribute 'command', and the list of any parameters provided on the line can be acessed by the named attribute 'params'.

TODO: Add support for escaping quote characters when embedded in strings delimited with the same quote style, as in ""hello" world"

Return type `pyparsing.Parser`

friendlyshell.shell_help_mixin module

friendlyshell.version module

1.1.2 Module contents

Package initialization...

CHAPTER 2

Overview

Framework for writing interactive Python command line interfaces, similar to the ‘cmd’ built in class.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

friendlyshell, 7
friendlyshell.base_shell, 3
friendlyshell.basic_logger_mixin, 4
friendlyshell.command_complete_mixin, 5
friendlyshell.command_parsers, 6
friendlyshell.version, 7

Index

A

alias_native_shell() (*friendlyshell.base_shell.BaseShell static method*), 3
auto_complete_manager() (*in module friendlyshell.command_complete_mixin*), 5

B

BaseShell (*class in friendlyshell.base_shell*), 3
BasicLoggerMixin (*class in friendlyshell.basic_logger_mixin*), 4

C

CommandCompleteMixin (*class in friendlyshell.command_complete_mixin*), 5

D

debug() (*friendlyshell.base_shell.BaseShell static method*), 3
debug() (*friendlyshell.basic_logger_mixin.BasicLoggerMixin method*), 5
default_command_token() (*in module friendlyshell.command_parsers*), 6
default_line_parser() (*in module friendlyshell.command_parsers*), 6
do_clear_history() (*friendlyshell.command_complete_mixin.CommandCompleteMixin method*), 5
do_close() (*friendlyshell.base_shell.BaseShell method*), 3
do_exit() (*friendlyshell.base_shell.BaseShell method*), 3
do_native_shell() (*friendlyshell.base_shell.BaseShell method*), 3

E

error() (*friendlyshell.base_shell.BaseShell static method*), 3
error() (*friendlyshell.basic_logger_mixin.BasicLoggerMixin method*), 5

F

friendlyshell (*module*), 7
friendlyshell.base_shell (*module*), 3
friendlyshell.basic_logger_mixin (*module*), 4
friendlyshell.command_complete_mixin (*module*), 5
friendlyshell.command_parsers (*module*), 6
friendlyshell.version (*module*), 7

H

help_close() (*friendlyshell.base_shell.BaseShell static method*), 4

I

info() (*friendlyshell.base_shell.BaseShell static method*), 4
info() (*friendlyshell.basic_logger_mixin.BasicLoggerMixin method*), 5

Q

quoted_space_sep_params_token() (*in module friendlyshell.command_parsers*), 6

R

return_code (*friendlyshell.base_shell.BaseShell attribute*), 4
run() (*friendlyshell.base_shell.BaseShell method*), 4
run_subshell() (*friendlyshell.base_shell.BaseShell method*), 4

W

warning() (*friendlyshell.base_shell.BaseShell static method*), 4
warning() (*friendlyshell.basic_logger_mixin.BasicLoggerMixin method*), 5