
phial Documentation

Release 0.10.1

James Seden Smith

Nov 24, 2022

CONTENTS:

1	phial package	3
1.1	Submodules	3
1.2	phial.bot module	3
1.3	phial.scheduler module	8
1.4	phial.wrappers module	11
2	Indices and tables	15
	Python Module Index	17
	Index	19

phial is a Slack bot framework, modelled loosely on [Flask](#)

PHIAL PACKAGE

1.1 Submodules

1.2 phial.bot module

The core of phial.

```
class phial.bot.Phial(token, config={'autoReconnect': True, 'baseHelpText': 'All available commands:',  
                                   'hotReload': False, 'loopDelay': 0.001, 'maxThreads': 4, 'prefix': '!',  
                                   'registerHelpCommand': True})
```

Bases: `object`

The Phial class acts as the main interface to Slack.

It handles registraion and execution of user defined commands, as well as providing a wrapper around `slackclient.SlackClient` to make sending messages to Slack simpler.

```
add_command(pattern, func, case_sensitive=False, help_text_override=None,  
            hide_from_help_command=False)
```

Registers a command with the bot.

This method can be used as a decorator via `command()`

Parameters

- **pattern** (`str`) – The pattern that a `Message`'s text must match for the command to be invoked.
- **func** (`Callable[... Union[None, str, Response, Attachment]]`) – The fuction to be executed when the command is invoked
- **case_sensitive** (`bool`) – Whether the `pattern` should respect case sensitivity.

Defaults to `False`

- **help_text_override** (`Optional[str]`) – Text that should be used as a description of the command using the inbuilt help function.

If not overridden the command's docstring will be used as the help text.

Defaults to `None`

- **hide_from_help_command** (`Optional[bool]`) – A flag to specify whether or not the inbuilt help command should hide this command from the list it generates.

Defaults to `False`

Raises `ValueError` – If command with the same pattern is already registered

Example

```
def hello():
    return "world"
bot.add_command('hello', world)
```

Is the same as

```
@bot.command('hello')
def hello():
    return "world"
```

Return type `None`

`add_fallback_command(func)`

Add a fallback command.

Registers a ‘fallback’ function to run when a user tries to execute a command that doesn’t exist.

This method can be used as a decorator via `fallback_command()`

Parameters `func` (`Callable[[Message], Union[None, str, Response, Attachment]]`) – The function to be executed when the user tries to execute a command that doesn’t exist

Example

```
def error_handler(message: Message) -> str:
    return "Oops that command doesn't seem to exist"

bot.add_fallback_command(error_handler)
```

Is the same as

```
@bot.fallback_command()
def error_handler(message: Message) -> str:
    return "Oops that command doesn't seem to exist"
```

Return type `None`

`add_middleware(func)`

Adds a middleware function to the bot.

Middleware functions get passed every message the bot receives from slack before the bot process the message itself. Returning `None` from a middleware function will prevent the bot from processing it.

This method can be used as a decorator via `middleware()`.

Parameters `middleware_func` – The function to be added to the middleware pipeline

Example

```
def intercept(message):
    return message
bot.add_middleware(intercept)
```

Is the same as

```
@bot.middleware()
def intercept(message):
    return message
```

Return type `None`

add_scheduled(*schedule*, *func*)

Adds a scheduled function to the bot.

This method can be used as a decorator via *scheduled()*.

Parameters

- **schedule** (*Schedule*) – The schedule used to run the function
- **scheduled_func** – The function to be run in accordance to the schedule

Example

```
def scheduled_beep():
    bot.send_message(Response(text="Beep",
                             channel="channel-id">))
bot.add_scheduled(Schedule().every().day(), scheduled_beep)
```

Is the same as

```
@bot.scheduled(Schedule().every().day())
def scheduled_beep():
    bot.send_message(Response(text="Beep",
                             channel="channel-id">))
```

Return type `None`

alias(*pattern*)

A decorator that is used to register an alias for a command.

Parameters **pattern** (*str*) – The pattern that a Message's text must match for the command to be invoked.

Example

```
@bot.command('hello')
@bot.alias('goodbye')
def hello():
    return "world"
```

Return type `Callable`

command(*pattern*, *case_sensitive=False*, *help_text_override=None*, *hide_from_help_command=False*)

Registers a command with the bot.

This command is a decorator version of [add_command\(\)](#)

Parameters

- **pattern** (`str`) – The pattern that a `Message`’s text must match for the command to be invoked.
- **case_sensitive** (`bool`) – Whether the `pattern` should respect case sensitivity.

Defaults to `False`

- **help_text_override** (`Optional[str]`) – Text that should be used as a description of the command using the inbuilt help function.

If not overridden the command’s docstring will be used as the help text.

Defaults to `None`

- **hide_from_help_command** (`Optional[bool]`) – A flag to specify whether or not the inbuilt help command should hide this command from the list it generates.

Defaults to `False`

Example

```
@bot.command('hello')
def hello():
    return "world"

@bot.command('caseSensitive', case_sensitive=True)
def case_sensitive():
    return "You typed caseSensitive"
```

Return type `Callable`

```
default_config = {'autoReconnect': True, 'baseHelpText': 'All available
commands:', 'hotReload': False, 'loopDelay': 0.001, 'maxThreads': 4, 'prefix':
'!', 'registerHelpCommand': True}
```

Default configuration

fallback_command()

A decorator to add a fallback command.

See [add_fallback_command\(\)](#) for more information on fallback commands

Example

```
@bot.fallback_command()
def error_handler(message: Message) -> str:
    return "Oops that command doesn't seem to exist"
```

Return type `Callable`

middleware()

A decorator to add a middleware function to the bot.

See [add_middleware\(\)](#) for more information about middleware

Example

```
@bot.middleware()
def intercept(messaage):
    return message
```

Return type `Callable`

run()

Run the bot.

Return type `None`

scheduled(schedule)

A decorator that is used to register a scheduled function.

See [add_scheduled\(\)](#) for more information on scheduled jobs.

Parameters `schedule` ([Schedule](#)) – The schedule used to determine when the function should be run

Example

```
@bot.scheduled(Schedule().every().day())
def scheduled_beep():
    bot.send_message(Response(text="Beep",
                             channel="channel-id">))
```

Return type `Callable`

send_message(message)

Sends a message to Slack.

Parameters `message` ([Response](#)) – The message to be sent to Slack

Return type `None`

send_reaction(*response*)

Sends a reaction to a Slack Message.

Parameters **response** (*Response*) – Response containing the reaction to be sent to Slack

Return type *None*

upload_attachment(*attachment*)

Upload a file to Slack.

Parameters **attachment** (*Attachment*) – The attachment to be uploaded to Slack

Return type *None*

1.3 phial.scheduler module

The classes related to scheduling of regular jobs in phial.

class `phial.scheduler.Schedule`

Bases: *object*

A schedule stores the relative time for something to happen.

It can be used to compute when the next instance of an event should occur.

at(*hour, minute, second=0*)

Specifies the time of day the next occurrence will happen.

NOTE: ‘at’ can only be used with *day()*.

```
schedule = Schedule().every().day().at(12,00)
```

Parameters

- **hour** (*int*) – The hour of day the next event should happen, when combined with the minute
- **minute** (*int*) – The minute of day the next event should happen, when combined with the hour
- **second** (*int*) – The second of day the next event should happen, when combined with the hour and minute. Defaults to 0

Return type *Schedule*

day()

Adds a day to the relative time till the next event.

```
schedule = Schedule().every().day()
```

Return type *Schedule*

days(*value*)

Set the days till the next instance of the event.

Adds the specified number of days to the relative time till the next event.

```
schedule = Schedule().every().days(2)
```

Parameters `value` (`int`) – The number of days to wait between events

Return type `Schedule`

`every()`

Syntactic sugar to make schedule declaration more readable.

Syntactic sugar to allow the declaration of schedules to be more like an English sentence.

```
schedule = Schedule().every().day()
```

Return type `Schedule`

`get_next_run_time(last_run)`

Get the next time the job should run.

Calculates the next time to run, based on the last time the event was run.

Parameters `last_run` (`datetime`) – The last time the event happened

Return type `datetime`

Returns A `datetime` of when the event should next happen

`hour()`

Adds an hour to the relative time till the next event.

```
schedule = Schedule().every().hour()
```

Return type `Schedule`

`hours(value)`

Sets the hours till the next instance of the event.

Adds the specified number of hours to the relative time till the next event.

```
schedule = Schedule().every().hours(2)
```

Parameters `value` (`int`) – The number of hours to wait between events

Return type `Schedule`

`minute()`

Adds a minute to the relative time till the next event.

```
schedule = Schedule().every().minute()
```

Return type `Schedule`

minutes(*value*)

Sets the minutes till the next instance of the event.

Adds the specified number of minutes to the relative time till the next event.

```
schedule = Schedule().every().minutes(2)
```

Parameters **value** (*int*) – The number of minutes to wait between events

Return type *Schedule*

second()

Adds a second to the relative time till the next event.

```
schedule = Schedule().every().second()
```

Return type *Schedule*

seconds(*value*)

Sets the seconds till the next instance of the event.

Adds the specified number of seconds to the relative time till the next event.

```
schedule = Schedule().every().seconds(2)
```

Parameters **value** (*int*) – The number of seconds to wait between events

Return type *Schedule*

class `phial.scheduler.ScheduledJob`(*schedule, func*)

Bases: *object*

A function with a schedule.

run()

Runs the function and calculates + stores the next run time.

Return type *None*

should_run()

Checks whether the function needs to be run based on the schedule.

Return type *bool*

Returns A *bool* of whether or not to run

class `phial.scheduler.Scheduler`

Bases: *object*

A store for Scheduled Jobs.

add_job(*job*)

Adds a scheuled job to the scheduler.

Parameters **job** (*ScheduledJob*) – The job to be added to the scheduler

Return type *None*

run_pending()

Runs any pending scheduled jobs.

Runs any ScheduledJobs in the store, where `job.should_run()` returns true

Return type `None`

1.4 phial.wrappers module

Contains models for phial to use.

class `phial.wrappers.Attachment`(*channel, filename, content*)

Bases: `object`

A file to be uploaded to Slack.

Parameters

- **channel** (`str`) – The Slack channel ID the file will be sent to
- **filename** (`str`) – The filename of the file
- **content** (`IO`) – The file to send to Slack. Open file using `open('<file>', 'rb')`

Example

```
Attachment('channel', 'file_name', open('file', 'rb'))
```

class `phial.wrappers.Command`(*pattern, func, case_sensitive=False, help_text_override=None, hide_from_help_command=False*)

Bases: `object`

An action executable from Slack.

Parameters

- **pattern** (`str`) – A string that a Slack Message must match to trigger execution
- **func** (`Callable[... , Union[None, str, Response, Attachment]]`) – The function that will be triggered when the command is invoked
- **case_sensitive** (`bool`) – Whether the `pattern` should enforce case sensitivity
- **help_text_override** (`Optional[str]`) – Overrides the function's docstring in the standard help command
- **hide_from_help_command** (`Optional[bool]`) – Prevents function from being displayed by the standard help command

property help_text: `Optional[str]`

A description of the command's function.

Return type `Optional[str]`

pattern_matches(*message*)

Check if message should invoke the command.

Checks whether the text of a `Message` matches the command's pattern, or any of it's aliased patterns.

Return type `Optional[Dict[str, str]]`

class `phial.wrappers.Message`(*text, channel, user, timestamp, team, bot_id=None*)

Bases: `object`

A representation of a Slack message.

Parameters

- **text** (`str`) – The message contents
- **channel** (`str`) – The Slack channel ID the message was sent from
- **user** (`str`) – The user who sent the message
- **timestamp** (`str`) – The message’s timestamp
- **team** (`Optional[str]`) – The Team ID of the Slack workspace the message was sent from
- **bot_id** (`Optional[str]`) – If the message was sent by a bot the ID of that bot. Defaults to `None`.

`phial.wrappers.PhialResponse`

A union of all response types phial can use

alias of `Union[None, str, Response, Attachment]`

class `phial.wrappers.Response`(*channel, text=None, original_ts=None, reaction=None, ephemeral=False, user=None, attachments=None*)

Bases: `object`

A response to be sent to Slack.

When returned in a command function will send a message, or reaction to Slack depending on contents.

Parameters

- **channel** (`str`) – The Slack channel ID the response will be sent to
- **text** (`Optional[str]`) – The response contents
- **original_ts** (`Optional[str]`) – The timestamp of the original message. If populated will put the text response in a thread
- **reaction** – A valid slack emoji name. NOTE: will only work when `original_ts` is populated
- **attachments** (`Optional[List[Dict[str, Union[str, int, float, bool, List]]]]`) – Any Slack Message Attachments
- **ephemeral** (`bool`) – Whether to send the message as an ephemeral message
- **user** (`Optional[str]`) – The user id to display the ephemeral message to

Examples

The following would send a message to a slack channel when executed

```
@slackbot.command('ping')
def ping():
    return Response(text="Pong", channel='channel_id')
```

The following would send a reply to a message in a thread

```
@slackbot.command('hello')
def hello():
    return Response(text="hi",
                    channel='channel_id',
                    original_ts='original_ts')
```

The following would send a reaction to a message

```
@slackbot.command('react')
def react():
    return Response(reaction="x",
                    channel='channel_id',
                    original_ts='original_ts')
```


INDICES AND TABLES

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

PYTHON MODULE INDEX

p

`phial.bot`, [3](#)
`phial.scheduler`, [8](#)
`phial.wrappers`, [11](#)

INDEX

A

`add_command()` (*phial.bot.Phal* method), 3
`add_fallback_command()` (*phial.bot.Phal* method), 4
`add_job()` (*phial.scheduler.Scheduler* method), 10
`add_middleware()` (*phial.bot.Phal* method), 4
`add_scheduled()` (*phial.bot.Phal* method), 5
`alias()` (*phial.bot.Phal* method), 5
`at()` (*phial.scheduler.Schedule* method), 8
Attachment (class in *phial.wrappers*), 11

C

Command (class in *phial.wrappers*), 11
`command()` (*phial.bot.Phal* method), 6

D

`day()` (*phial.scheduler.Schedule* method), 8
`days()` (*phial.scheduler.Schedule* method), 8
`default_config` (*phial.bot.Phal* attribute), 6

E

`every()` (*phial.scheduler.Schedule* method), 9

F

`fallback_command()` (*phial.bot.Phal* method), 6

G

`get_next_run_time()` (*phial.scheduler.Schedule* method), 9

H

`help_text` (*phial.wrappers.Command* property), 11
`hour()` (*phial.scheduler.Schedule* method), 9
`hours()` (*phial.scheduler.Schedule* method), 9

M

Message (class in *phial.wrappers*), 11
`middleware()` (*phial.bot.Phal* method), 7
`minute()` (*phial.scheduler.Schedule* method), 9
`minutes()` (*phial.scheduler.Schedule* method), 9
module
 phial.bot, 3

phial.scheduler, 8
 phial.wrappers, 11

P

`pattern_matches()` (*phial.wrappers.Command* method), 11
Phial (class in *phial.bot*), 3
phial.bot
 module, 3
phial.scheduler
 module, 8
phial.wrappers
 module, 11
PhialResponse (in module *phial.wrappers*), 12

R

Response (class in *phial.wrappers*), 12
`run()` (*phial.bot.Phal* method), 7
`run()` (*phial.scheduler.ScheduledJob* method), 10
`run_pending()` (*phial.scheduler.Scheduler* method), 10

S

Schedule (class in *phial.scheduler*), 8
`scheduled()` (*phial.bot.Phal* method), 7
ScheduledJob (class in *phial.scheduler*), 10
Scheduler (class in *phial.scheduler*), 10
`second()` (*phial.scheduler.Schedule* method), 10
`seconds()` (*phial.scheduler.Schedule* method), 10
`send_message()` (*phial.bot.Phal* method), 7
`send_reaction()` (*phial.bot.Phal* method), 7
`should_run()` (*phial.scheduler.ScheduledJob* method), 10

U

`upload_attachment()` (*phial.bot.Phal* method), 8