Sparkbot Documentation

Dalton Durst

Aug 07, 2018

Contents:

Welcome to the documentation for SparkBot! If you're looking for the fastest way to get running, check out *Quickstart*!

CHAPTER 1

Quickstart

This document will lead you through the steps to run the base Sparkbot instance.

1.1 Get a token from Webex Teams

Head over to Cisco Webex Teams for Developer's My Apps portal and click the Add button to create a new bot. Go through the steps to create a bot. Once you're finished, copy the Bot's Access Token somewhere safe. We'll need it later in this process.

1.2 Dependencies

First you'll need to install the prerequisites for running SparkBot.

SparkBot requires the following software:

- Python 3.5 or higher
- Reverse proxy, such as nginx, for its webhook receiver. We'll be using ngrok in this quickstart.

1.2.1 Ubuntu 16.04

To install the prerequisites on Ubuntu 16.04:

```
sudo apt install python3 python3-virtualenv python3-pip nginx
```

1.3 Clone the source

Clone the bot's source to your desired location. From here on, we'll assume that the bot's source code is located in `~/sparkbot`, but you can change the path as you need.

1.4 Copy run.py.example

`run.py.example` contains the code needed to run SparkBot. It's also where you'll create new commands for the bot. We'll copy it to `run.py` now:

```
cp ~/sparkbot/run.py.example ~/sparkbot/run.py
```

1.5 Set up a virtualenv

Create and activate a Python(3.5+) virtualenv for the bot:

```
python3 -m virtualenv ~/sparkbotEnv
source ~/sparkbotEnv/bin/activate
```

Now we can install the required Python packages:

```
pip install -r ~/sparkbot/requirements.txt
pip install gunicorn
```

1.6 Use ngrok for a temporary reverse proxy

ngrok is a great service for setting up temporary public URLs. We'll be using it to quickly test our bot configuration. Download ngrok, then run `ngrok http 8000` to get it running.

1.7 Run the bot

We can now test the bot. Sparkbot requires that a few environment variables be set, so we'll `export` them before we run:

```
cd ~/sparkbot
source ~/sparkbotEnv/bin/activate
export SPARK_ACCESS_TOKEN=[api_token]
export WEBHOOK_URL=[url]
gunicorn run:bot.receiver
```

Replace `[url]` with the URL that points to your webhook endpoint. Since we're using ngrok, put the `https` Forwarding URL here. Replace `[api_token]` with the token that Webex Teams gave you for your bot.

The bot should now be running and, assuming your proxy is working correctly, be able to receive requests directed at it from Webex Teams. Try messaging the bot with `ping` or `help` to see if it will respond.

1.8 Next steps

Now that you've got the bot running, you may want to learn more about *Writing Commands* or *Deploying SparkBot*

CHAPTER 2

Writing Commands

2.1 Introduction

SparkBot provides a very simple interface for writing commands. You will be familiar with it if you have ever used Flask. Here's a simple `ping` command:

```
@MY_BOT.command("ping")
def ping():
    """
    Checks if the bot is running.

    Usage: `ping`

    Returns **pong**.
    """

    return "**pong**"
```

Let's break down what's happening here line-by-line.

First, we have the decorator, `sparkbot.core.SparkBot.command()`, which marks this function as a command for our bot:

```
@MY_BOT.command("ping")
```

`bot` is the SparkBot instance that we're adding this command to. `"ping"` is what we want the user to type in order to invoke this command.

Next, the function definition and docstring:

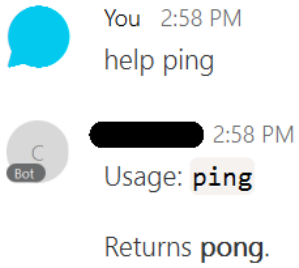
```
def ping():
    """
    Usage: `ping`
```

(continues on next page)

(continued from previous page)

```
    Returns **pong**.
    """
```

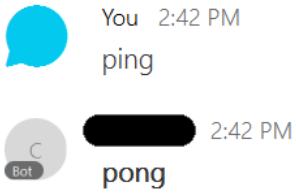
The docstring also serves as the command's help, accessible via the `help [command]` command. It must all be equally spaced (so don't put the description on the same line as the opening quotes like you would in most cases) and is formatted in Markdown. You should stick to the general format of description, usage, returns when writing your docstrings for commands.



Finally, we see how simple it is to send formatted text back to the user:

```
return "**pong**"
```

When we add this to the area below the `# Add commands here` comment and re-run the bot, we can now use the `ping` command:



Note: Commands must always be added to the bot prior to the receiver starting. This means that the bot cannot add or remove commands from *itself*. Changes will always require a restart.

2.2 Taking arguments

In many cases you will want to take arguments to your commands. Sparkbot uses shlex.split to split the message sent by the user into multiple 'tokens' that are given to you in a list. These tokens are split in a similar way to a POSIX shell.

Here's a command that uses this type of input. It returns the first token in the list:

```
@MY_BOT.command("testcommand")
def testcommand(commandline):
    """
    Usage: `testcommand something`

```

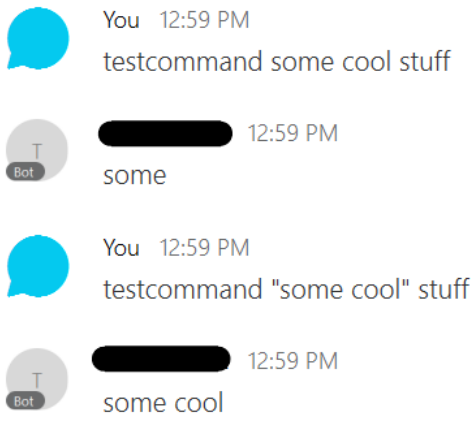
(continues on next page)

(continued from previous page)

```
    A command used for testing. Returns the first word you typed.
    """

    if commandhelpers.minargs(1, commandline):
        return commandline[1]
    else:
        return 'This command requires at least one argument'
```

While the help says that this will only return the first word, this command will also return the first quoted string that's typed as well.



Let's go over this line-by-line:

```
@MY_BOT.command("testcommand")
def testcommand(commandline):
```

As usual, we use the `sparkbot.core.SparkBot.command()` decorator to add this function to our bot's list of commands. However, notice that we defined the function to take the argument `commandline`. This is one of several keywords that SparkBot recognizes. When executing your function, it will find this keyword and send the `commandline` property accordingly.

When the user types `testcommand some cool stuff`, this code receives the following list as its `commandline` argument:

```
['testcommand', 'some', 'cool', 'stuff']
```

Whereas `testcommand "some cool" stuff` will yield the following:

```
['testcommand', 'some cool', 'stuff']
```

Using a helper function, `sparkbot.commandhelpers.minargs()`, we check to make sure we have at least one argument (token) in the commandline. Then, we return either the first token if there is one or more, or an error if there are no tokens:

```
if commandhelpers.minargs(1, commandline):
    return commandline[1]
else:
    return 'This command requires at least one argument'
```

As you can see, you can quickly create a CLI-like interface by iterating over the tokens in this list.

2.3 Replying early

SparkBot allows you to use the `yield` keyword in place of `return` to reply to the user before your command's code has completed. This may be useful if you have a command that will perform a very long operation and you would like to notify the user that it is in progress.

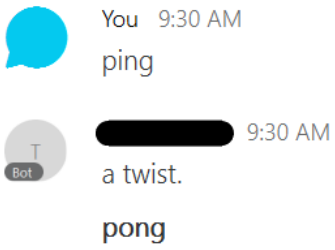
```
@MY_BOT.command("ping")
def ping_callback():
    """
    Usage: `ping`

    Returns **pong**, but with a twist.
    """

    yield "a twist"

    # Some code which runs for a long time

    yield "**pong**"
```



Changed in version 0.1.0: `yield` to reply early has been added as a replacement for the `callback` argument previously used to get a function used for the same purpose. `callback` will be removed in SparkBot version 1.0.0.

2.4 Overriding behavior

SparkBot comes with default behavior that will work well for simple bots. However, you may need to override some of this behavior to provide a richer experience for your users.

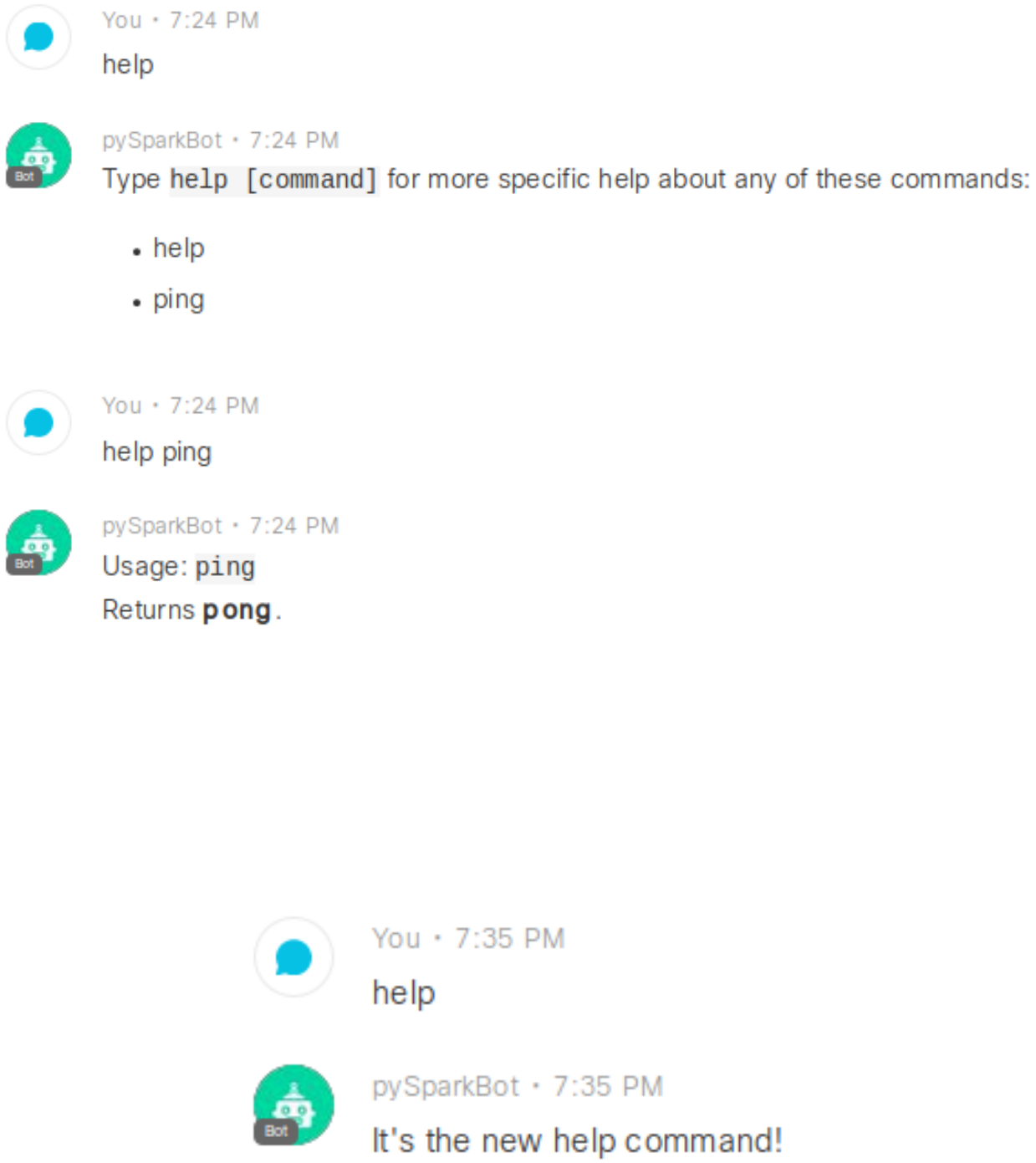
2.4.1 "Help" command

Override

The default SparkBot `help` command is simplistic:

If you want to do something different when your user asks for help, you can add a new command in the same slot as "help":

```
@bot.command("help")
def new_help():
    return "It's the new help command!"
```


You • 7:24 PM
help
pySparkBot • 7:24 PM
Type help [command] for more specific help about any of these commands:
help
ping
You • 7:24 PM
help ping
pySparkBot • 7:24 PM
Usage: ping
Returns pong.
You • 7:35 PM
help
pySparkBot • 7:35 PM
It's the new help command!

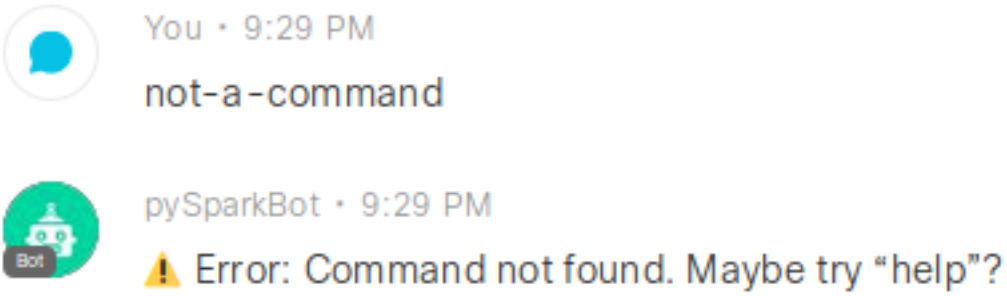
Remove

If you'd prefer to remove the help command altogether, you can do so by calling `SparkBot.remove_help()`.

Note: Similar to adding commands, removing commands must be performed before the bot has started. It is not possible to remove help "in-flight", such as from another command.

2.4.2 "Command not found"

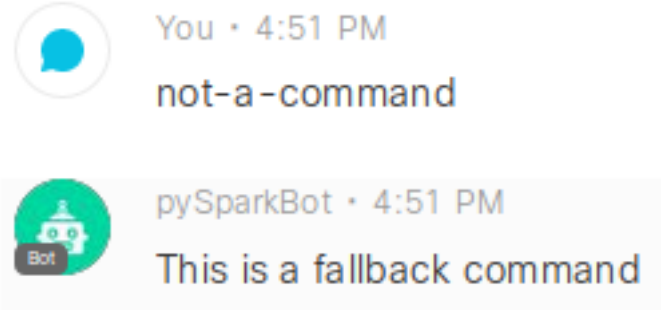
By default, when the user tries to use a command that doesn't exist, they get an error:



It may be desirable for you to do something else (return a more fun error message, give suggestions rather than an error, or maybe use NLP to determine what the user wanted).

You can add a command as a fallback by omitting its command strings and adding the `fallback_command=True` argument to the command decorator:

```
@bot.command(fallback=True)
def fallback():
    return "This is a fallback command"
```



2.5 List of recognized keywords

Keyword	Data
commandline	List containing user's message split into tokens by shlex.split. *Taking arguments*
event	Dictionary containing the event request from Spark.
caller	ciscosparkapi.Person for the user that called this command
room_id	`Str` containing the ID of the room where this command was called

CHAPTER 3

Deploy

When it comes time to deploy your bot to a server, we recommend using gunicorn and nginx. The following information will help you run the bot under gunicorn with nginx as its reverse proxy.

This information is adapted from the Deploying Gunicorn document, you may wish to head to it for more advanced setups.

3.1 Install required system packages

Before starting, it is important to have `nginx` and the appropriate Python 3 packages installed.

3.1.1 Ubuntu 16.04 / 18.04

```
sudo apt install nginx python3 python3-pip python3-virtualenv
```

3.2 Install Python packages in a virtualenv

Create a virtualenv for SparkBot with the required packages. This will keep system-level Python packages separate from your SparkBot packages.

It's a good idea to create a new service account with the bare minimum permissions for Sparkbot:

```
sudo useradd --system --create-home sparkbot
```

Now, log in to the sparkbot user so we can install the virtualenv:

```
sudo -Hu sparkbot /bin/bash
```

Finally, create the virtualenv and install SparkBot into it:

```
python3 -m virtualenv --python=python3 /home/sparkbot/sparkbotenv
source /home/sparkbot/sparkbotenv/bin/activate
pip install git+https://github.com/universalsuperbox/SparkBot.git gunicorn
deactivate
exit
```

3.3 Get your run.py script

This guide assumes that your SparkBot script is called `run.py` and is placed at `/home/sparkbot/run.py`. If your script is named differently, change `run` in `run:bot.receiver` in the `ExecStart` entry to the script's name (without `.py`). If your script is located in a different directory, change the WorkingDirectory.

3.4 Add nginx configuration

We'll use nginx to proxy requests to the bot. You may use this configuration as a template for your reverse proxy for the bot's webhook receiver:

Listing 1: /etc/nginx/conf.d/sparkbot.conf

```
upstream app_server {
  # fail_timeout=0 means we always retry an upstream even if it failed
  # to return a good HTTP response

  # for UNIX domain socket setups
  server unix:/run/sparkbot/socket fail_timeout=0;
}

server {
  # if no Host match, close the connection to prevent host spoofing
  listen 80 default_server;
  return 444;
}

server {
  # use 'listen 80 deferred;' for Linux
  # use 'listen 80 accept_filter=httpready;' for FreeBSD
  listen 80;
  client_max_body_size 4G;

  # set the correct host(s) for your site
  server_name example.com www.example.com;

  keepalive_timeout 5;

  # path for static files
  root /path/to/app/current/public;

  location / {
    # checks for static file, if not found proxy to app
    try_files $uri @proxy_to_app;
  }
```

(continues on next page)

(continued from previous page)

```
  location @proxy_to_app {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header Host $http_host;
    # we don't want nginx trying to do something clever with
    # redirects, we set the Host: header above already.
    proxy_redirect off;
    proxy_pass http://app_server;
  }
}
```

Remember to set the `server_name` property to the FQDN of your server.

It is highly recommended to use HTTPS for this reverse proxy, but setting that up is outside of the scope of this guide.

3.5 Auto-start with systemd

First, we'll add a unit file for the Gunicorn socket. This goes at `/etc/systemd/system/sparkbot.socket`:

Listing 2: /etc/systemd/system/sparkbot.socket

```
[Unit]
Description=SparkBot gunicorn socket

[Socket]
ListenStream=/run/sparkbot/socket

[Install]
WantedBy=sockets.target
```

Next, create the file `/etc/systemd/system/sparkbot.service` with the following content. Once finished, save and close the file then run `systemctl daemon-reload`:

Listing 3: /etc/systemd/system/sparkbot.service

```
[Unit]
Description=Webex Teams chatbot
Requires=sparkbot.socket
After=network.target

[Service]
PIDFile=/run/sparkbot/pid
RuntimeDirectory=sparkbot
Environment="SPARK_ACCESS_TOKEN="
Environment="WEBHOOK_URL="
User=sparkbot
ExecStart=/home/sparkbot/sparkbotenv/bin/gunicorn \
          --bind unix:/run/gunicorn/socket run:bot.receiver
WorkingDirectory=/home/sparkbot/
Restart=on-abort
StandardOutput=journal
ExecReload=/bin/kill -s HUP $MAINPID
ExecStop=/bin/kill -s TERM $MAINPID
PrivateTmp=true
```

(continues on next page)

(continued from previous page)

```
[Install]
WantedBy=multi-user.target
```

Next, run `systemctl edit sparkbot.service` and enter the following, changing the options in curly brackets to match your desired settings:

Listing 4: systemctl edit sparkbot.service

```
[Service]
Environment="SPARK_ACCESS_TOKEN={api_token}"
Environment="WEBHOOK_URL={url}"
```

The values should be the same as the ones you used when you followed *the Quickstart guide*.

Once that's finished, run the following to enable the bot on startup:

```
sudo systemctl daemon-reload
sudo systemctl enable sparkbot.socket
sudo systemctl enable sparkbot.service
sudo systemctl start sparkbot.socket
sudo systemctl start sparkbot.service
```

CHAPTER 4

API Documentation

This page contains information about SparkBot's internals. Bot authors may not find it terribly useful, SparkBot hackers will.

4.1 SparkBot

class sparkbot.**SparkBot**(*spark_api*, *root_url=None*, *logger=None*, *skip_receiver_setup=None*, *custom_receiver_resources={}*)

Bases: object

A bot for Cisco Webex Teams

SparkBot automatically creates a webhook for itself and will delete any other webhooks on its bot account. To do this, it uses the root_url parameter or WEBHOOK_URL in the environment to know its public URL.

SparkBot has a help command built in by default. These may be overridden using the *command()* decorator and providing the "help" argument and a function with your desired behavior on calling help. See *Writing Commands* for more information on writing commands.

Parameters

- **spark_api** (*ciscosparkapi.CiscoSparkAPI*) – CiscoSparkAPI instance that this bot should use
- **root_url** (*str*) – The base URL for the SparkBot webhook receiver. May also be provided as WEBHOOK_URL in the environment.
- **logger** (*logging.Logger*) – Logger that the bot will output to
- **skip_receiver_setup** (*"all", "webhook"*) – Set to "all" or "webhook" to skip setting up a receiver when instancing SparkBot. "all" skips creating the receiver and the webhook. "webhook" creates a receiver but does not register a webhook with Webex Teams.
- **custom_receiver_resources** – dict containing custom resources for the receiver. Pass a dict of Falcon resource(s) with the endpoint you would like them hosted at as the key.

For example, `{"/my_endpoint": EndpointResource}` will serve EndpointResource at `/my_endpoint`.

my_help_all()
Returns a markdown-formatted list of commands that this bot has. This function is used by the default "help" command to show the full list.

my_help(*commandline*)
Returns the help of the command given in `commandline`. Calls *my_help_all()* if no command is given or is `all`. Called by a user by typing `help`. This function is the default "help" command and can be removed using *remove_help()*.

command(*command_strings=[]*, *fallback=False*)
Decorator that adds a command to this bot.

Parameters

- **command_strings** (*list or str*) – Callable name(s) of command. When a bot user types this (these), they call the decorated function. Pass a single string for a single command name. Pass a list of strings to give a command multiple names.
- **fallback** (*bool*) – False by default, not required. If True, sets this command as a "fallback command", used when the user requests a command that does not exist.

Raises

- **CommandSetupError** – Arguments or combination of arguments was incorrect. The error description will have more details.
- **TypeError** – Type of arguments was incorrect.

command_dispatcher(*user_request*)
Executes a command for the user's request.

This method is called by the receiver when a command comes in. It uses the information in the user_request to execute a command (using *execute_command()*) and send its reply back to the user.

Parameters **user_request** (*ciscosparkapi.WebhookEvent*) – Event where the user called the bot

create_callback(*respond*, *room_id*)
Pre-fills room ID in the function given by `respond`

Adds the room ID as the first argument of the function given in `respond`, simplifying the 'callback' experience for bot developers.

Parameters

- **respond** – The method to add the room ID to
- **room_id** – The ID of the room to preset in `respond`

execute_command(*command_str*, ***kwargs*)
Runs the command given by 'command_str' if it exists with the possible arguments in `**kwargs`.

Note that execute_command is "dumb". It does not enforce the return type of a command function. It will happily return anything that the bot writer's command does. Contrast to *command_dispatcher()* which checks whether a command (executed by this function) returns either a Generator or a str.

Parameters

- **command_str** (*str*) – The 'command' that the user wants to run. Should match a command string that has previously been added to the bot.

- **commandline** (*list*) – The user's complete message to the bot parsed into a list of tokens by `shlex.split()`.
- **event** (*ciscosparkapi.WebhookEvent*) – ciscosparkapi.WebhookEvent object describing the request causing this command.
- **caller** (*ciscosparkapi.Person*) – The user who sent the message we're processing.
- **room_id** (*str*) – The ID of the room that the message we're processing was sent in.

Keyword Arguments Each keyword argument sent here will be used as a possible argument for commands. For example, the keyword argument `commandline=` will allow a command function (the ones defined with *command()*) to take an argument by the name `commandline`. The current list can be found at *List of recognized keywords*.

If a logger is defined, SparkBot will raise a warning if an argument is requested but not provided by the `kwargs` given to this function. This means that there is either a typo in the argument on the command function or *command_dispatcher()* has failed to do its job correctly (the first is more likely).

remove_help()
Removes the help command from the bot

This will remove the help command even if it has been overridden.

send_message(*spark_room*, *markdown*)
Sends a message to a Teams room.

Parameters

- **markdown** – Markdown formatted string to send
- **spark_room** – The room that we should send this response to, either CiscoSparkAPI.Room or str containing the room ID

4.2 Default receiver

The receiver waits for a request from Webex Teams and executes *sparkbot.SparkBot.command_dispatcher()* when one comes in.

class sparkbot.receiver.**ReceiverResource**(*bot*)
Bases: `object`

on_post(*req*, *resp*)
Receives messages and passes them to the sparkbot instance in BOT_INSTANCE

sparkbot.receiver.**create**(*bot*, ***kwargs*)
Creates a falcon.API instance with the required behavior for a SparkBot receiver.

Currently the API webhook path is hard-coded to `/sparkbot`

Parameters **bot** – *sparkbot.SparkBot* instance for this API instance to use

Keyword Arguments Additional arguments may be used to specify more resources that should be exposed by the SparkBot receiver. For example, `"/my_webhook"=[Falcon resource]` will expose your Falcon resource at `/my_webhook` on the server.

sparkbot.receiver.**random_bytes**(*length*)
Returns a random bytes array with uppercase and lowercase letters, of length length

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index