bc3cb Documentation

Release 3.5

Dalton Durst

Nov 09, 2017

Contents:

CHAPTER 1

Installation

I used Ubuntu 16.04 to install bc3cb for the first time. It'll probably work with any other OS.

1.1 Requirements

1. A URL for BC3CB. Basecamp will call the bot at this URL. It must be HTTPS.
2. Admin rights on your Basecamp 3 account
3. A server with pip and virtualenv installed

1.2 Procedure

1. Create a new user for bc3cb. Its password can be anything, we'll be disabling remote login anyway. `adduser bc3cb`
2. Remove the new user's login rights by editing /etc/passwd and switching the login shell to one that doesn't allow login.
3. Switch to the user with `su -s /bin/bash bc3cb`
4. Create a virtualenv named bc3cbEnv in its home folder `python3 -m virtualenv ~/bc3cbEnv`
5. Enter the virtualenv to install software `. ~/bc3cbEnv/bin/activate`
6. Install the required packages to run bc3cb `pip install flask requests`
7. Set up a reverse proxy to `[hostname]:5000/basecamp3receiver`. It doesn't matter how you do it or with what software, it just needs to provide a valid HTTPS session (that means you need a valid TLS certificate). I set up nginx to proxy the URL to /receive
8. Set up Basecamp
   (a) Open a project on Basecamp and select "Settings" in the upper right corner.

   (b) Select "Chatbots"
   (c) Pick "Add a new chatbot"
   (d) Give your bot a fun name and set its command URL to your proxy URL

Now all you'll need to do is run `run.py` and your bot will be ready to rock!.

TODO: Setting up nginx for reverse proxying TODO: Automatic start

Good proxy configuration, if you have a proxy in front:

```
server {
    listen 80
    server_name [DOMAIN]
    root /var/www/default
    location / {
            proxy_pass http://localhost:5000/;
    }
}
```

CHAPTER 2

Writing Behavior

The Bot Creator defines the behavior for all commands that users can run in `usercommands.py`.

2.1 Introduction

Creating new behavior consists of two easy steps:

1. Think about what word you want the user to call to invoke the behavior
2. Define a function with that word as its name. From that function, return whatever you want to reply to the user with. bc3cb will handle the rest.

Easy. Just like you've always wanted.

2.2 Writing your First Command

Consider the default *ping* command:

```
# bc3cb/usercommands.py

def ping(commandline, commandinfo):
    """
    Returns 'pong'.
    """

    return 'pong'
```

In Basecamp, when someone says '!bot ping' or directly messages the bot saying 'ping', the bot replies with 'pong'.

We're going to create a simple command that performs a dice roll.

Every command starts with your function header:

```
# bc3cb/usercommands.py

def diceroll(commandline, commandinfo):
    # statements go here!
```

Just like that, you've created a new command. If a user calls your bot by saying '!botname diceroll', it will be executed.

Note that all of your functions must take exactly two arguments. It is recommended to call them commandline and commandinfo, just like the example above. commandline and commandinfo will be discussed later on this page.

Now, let's add some code that generates a random number between 1 and 6.:

```
# bc3cb/usercommands.py

def diceroll(commandline, commandinfo):
    import random

    return random.randint(1, 6)
```

This will reply to the user with a random number between 1 and 6.

Notice that the `import` statement is inside of the function rather than at the top of the file. This is called lazy importing and is used because every invocation of the bot is started in a new process. Importing *everything* that *any* of your commands may require every time your bot is invoked is wasteful, whereas lazy importing takes less time.

2.3 What is 'commandline'?

`commandline` is always sent as the first argument of your functions. It is the message that the user sent to your bot, split into a token list using shlex.

For example, if the user sends "this is a command", commandline will be the following list:

```
['this', 'is', 'a', 'command']
```

This gets more exciting, of course. Shlex also allows the use of quoted strings. The user may send "this is 'a command'" and you will see:

```
['this', 'is', 'a command']
```

This allows you to create command line-like syntaxes for your bot.

2.4 What is 'commandinfo'?

`commandinfo` is always sent as the second argument of your functions. It is a dictionary that represents the JSON payload sent by Basecamp. This includes important information such as the user's name and title. You can read more about this JSON payload on the Basecamp 3 API Chatbots page.

2.5 Raising Exceptions

bc3cb tries its hardest to always reply to the user that invokes it. This includes a statement that catches any exception and replies with an error message.

You can throw any type of exception you would like, including the base Exception:

```
# bc3cb/usercommands.py

def neverworks(commandline, commandinfo):
    raise Exception('ShortDescriptiveName', 'Long error message that will show as the␣
↪summary of the error')
```

This will be caught by bc3cb. It will then send a message to the user saying "Long error message that will show as the summary of the error". The ShortDescriptiveName will be found by clicking to expand the message. It will also be logged to stdout as the following:

```
bc3cbCore - ERROR - "Basecamp User Name" caused: ShortDescriptiveName with the␣
↪command: [user message to bot]
```

Consider making the ShortDescriptiveName field different every time you write an exception. This will aid you in debugging.

2.6 Replying Before Returning

bc3cb passes Basecamp's "Callback URL" to you as part of commandinfo. This is the URL that you can use to create a new message with an HTTP POST. You can also import bc3cb.respond in your behavior, which handles the POST for you. Together, this means that you can easily send a response to the user before your command has completed.

For example, you can tell them that you're processing their long-running request:

```
def reallylongcommand(commandline, commandinfo):
    from . import respond

    interimresponse = "I'm working on it!"
    respond.respond(interimresponse, commandinfo['callback_url'])

    # Do some more stuff that'll take a while

    return "All done!"
```

The important lines are these:

```
from . import respond
respond.respond(string, commandinfo['callback_url'])
```

2.7 Some Notes

- You can't call anything within bc3cb.core from your commands as *core.py* imports *usercommands.py*.

CHAPTER 3

Internals

3.1 Internal Errors

Check out *Writing Behavior* to learn about returning errors from your behavior.

Internal errors are thrown back and forth inside of bc3cb. While it's valid for the bot creator to return these exceptions, it is not recommended. bc3cb will return different errors depending on the exception.

More will be added when the need arises.

3.1.1 bc3cbCommandNotFound

Used when the bot user requests a command that hasn't been defined by the user. For example, they called the bot with *!bot weqyasdrvjklase*, but you don't have a *weqyasdrvjklase* function in *usercommands.py*.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search