Mattermost BabelFish Documentation

Release develop

confirm IT solutions

Jun 30, 2017

Contents

1 The Docs **1**
- 1.1 Requirements . . . 1
- 1.2 Building the HTML docs . . . 1

2 Introduction **3**
- 2.1 What is BabelFish? . . . 3
- 2.2 Why BabelFish? . . . 3

3 Installation **5**

4 Run **7**
- 4.1 Flask server . . . 7
- 4.2 WSGI . . . 7

5 Configuration **9**
- 5.1 Settings . . . 9
- 5.2 Local settings . . . 9
- 5.3 Environment variables . . . 9

6 Plugins **11**
- 6.1 Giphy . . . 11
- 6.2 Github . . . 11

7 Developing plugins **13**
- 7.1 Naming . . . 13
- 7.2 Base plugin classes . . . 13
- 7.3 A slash command plugin example . . . 13

8 Indices and tables **15**

CHAPTER 1

The Docs

The purpose of the docs is the documentation of the Mattermost BabelFish framework. We're building the docs based on Sphinx Doc, the Python documentation tool.

1.1 Requirements

Before you can work with the docs you've to make sure you've installed all required Python packages / libraries. To install all dependencies you can use the requirements file.

It's recommended to use a **Python virtualenv** and place it in `.venv` or symlink it to `.venv`:

```
# Install virtualenv.
pip install virtualenv virtualenvwrapper

# Create new virtualenv and activate it.
virtualenv .venv
source .venv/bin/activate

# Install Python dependencies.
pip install -r requirements.txt
```

If you **don't want to use a virtualenv** you can simply run the following command to install the dependencies in your system site-packages:

```
pip install -r requirements.txt
```

1.2 Building the HTML docs

The built documentation is not included in the git repository. However, you can easily **build the documentation** by running the following command:

```
make html
```

Note: In case you get an error that the `sphinx-build` command was not found, you've to make sure you've installed the *requirements* and loaded the virtualenv by executing `source .venv/bin/activate`.

CHAPTER 2

Introduction

2.1 What is BabelFish?

BabelFish is a **framework** for Mattermost **webhooks** and **slash commands**. See it as a central hub for all your integrations. BabelFish is also...

- written in Python
- using Flask
- modular
- plugin-based
- well documented

2.2 Why BabelFish?

Let me explain why we created BabelFish.

We switched from Slack to Mattermost a while ago. Slack supports a lot of integrations for popular services. Unfortunately, Mattermost isn't that mature (yet) and will only support properitary webhooks and slash commands.

This means, if your service doesn't talk Mattermost JSON-ish, the integration won't work out of the box. Thus, you need something in between Mattermost and your application. And this is exactly why BabelFish was created.

BabelFish is a central hub which translates between the Mattermost JSON API and 3rd party services like Github and Giphy. Of course there are other alternatives out there, but most of them aim at exactly one function and you'll end up with running multiple application servers, each configured differently.

Another reason why we've created BabelFish is the open-source community. We love open-source and we want to give something back. We really hope that you like the concept and we'd love to see people developing more plugins.

CHAPTER 3

Installation

BabelFish runs on Python 3 and can be installed on Linux servers. It might work on Windows too, but we've never tested it - Windows is evil!

To install BabelFish, you've to:

```
# Clone the repository.
git clone git@github.com:confirm/Mattermost-BabelFish.git
cd Mattermost-BabelFish/babelfish/

# Create Python virtualenv.
virtualenv -p python3 .venv
source .venv/bin/activate

# Install requirements.
pip install -r requirements.txt
```

CHAPTER 4

Run

4.1 Flask server

For testing or debugging you can run the Flask app directly:

```
FLASK_APP=babelfish.py flask run
```

Hint: Please note that you must be in the `babelfish/` directory and that you might need to activate your virtualenv first.

4.2 WSGI

To run BabelFish in production, it's recommended that you use a proper WSGI application server. We're using uWSGI for our installation.

Here are to most important bits of our config:

```
[uwsgi]
socket     = <path to socket>
plugins    = python3
chdir      = <path to babelfish directory>
virtualenv = <path to virtualenv directory>
module     = babelfish:app
# more config...
```

CHAPTER 5

Configuration

5.1 Settings

All the configuration settings can be found in the `babelfish/settings.py` file.

Tip: Please read the comments in `settings.py` for more informations about the configuration parameters.

Caution: Do not change or overwrite `settings.py`. Instead of it, use one of the methods described below.

5.2 Local settings

If you want to customize your settings in a file, we recommend you create a new `settings_local.py` file next to the `settings.py` file. This will automatically be loaded and it is ignored by git. You can overwrite all the settings variable within this file.

5.3 Environment variables

As you might see in `babelfish/settings.py`, most of the parameters can also be configured via environment variables.

CHAPTER 6

Plugins

6.1 Giphy

6.1.1 Usage

The giphy plugin can be used as slash command, for example by typing `/giphy <text>`.

The plugin will lookup a matching gif image via the Giphy API and display it to all users in the channel.

6.1.2 Configuration

The giphy plugin requires minimal configuration:

- Create a new slash command in Mattermost to the URL endpoint `/giphy`
- Configure the token for the giphy plugin in BabelFish (*optional but recommended*)
- Configure a Giphy API key and/or the image rating (*optional*)

6.2 Github

6.2.1 Usage

The github plugin implements a webhook to display Github notifications in a Mattermost channel.

The plugin makes use of Mattermost's `attachments` and Markdown features (e.g. commits will be displayed in a Markdown table).

6.2.2 Configuration

Configuration of the github plugin is required:

- Create a new incoming webhook in Mattermost
- Configure the webook URL for the github plugin in BabelFish
- Create a new webhook in Github and point it to the URL endpoint `/github`
- Configure a github secret in BabelFish and in the GitHub webhook (*optional but recommended*)

CHAPTER 7

Developing plugins

7.1 Naming

Plugins should be named properly and obviously according to their function. For example:

- Desired slash command: `/giphy` _(can still be customised in your Mattermost integration)_
- Plugin name: `giphy`
- URL endpoint: `/giphy`
- Python plugin module & class: `plugins.giphy.GiphyPlugin`
- Settings: `GIPHY_*`

As you can see, the name of the plugin matches the URL endpoint and the Python module. The class itself is properly written in `UpperCamelCase` with the suffix `...Plugin`.

7.2 Base plugin classes

Ensure your plugin class is inherting from one of the following classes:

- `plugins.base.BaseSlashCommandPlugin` for slash command plugins
- `plugins.base.BaseWebhookPlugin` for webhook plugins

These classes implement the required parsing of the requests & responses, as well as the checks of the mattermost tokens.

7.3 A slash command plugin example

Creating new plugins is easy and straight-forward as you can see in the following example.

7.3.1 Say hello

Let's say for some reason you want to create a plugin which simply writes `Hello <name>` to the channel. In the case, create a new file stored under `plugins/hello.py` with the following content:

```
from base import BaseSlashCommandPlugin


class HelloPlugin(BaseSlashCommandPlugin):

    def request(self, username, text):
        return self.response('Hello ' + text)
```

Now add the plugin to the `PLUGINS` list in the `settings.py` file and you're ready to go!

7.3.2 Custom response username

If you want to send the message as a different user, set the `username` argument:

```
return self.response('Hello ' + text, username="Awesome Plugin")
```

7.3.3 Add plugin settings

Now let's say you want be able to configure the `Hello` string outside of your plugin. Let's add a new parameter to the `settings.py` file.

Your plugin settings variables need to be...

- all uppercase
- prefixed with your plugin name and an underscore (_)
- not named `<PLUGIN>_TOKEN` or `<PLUGIN>_WEBHOOK`

So let's add the following variable to the `settings.py` file:

```
HELLO_WORD = "Hello"
```

Now let's update our class accordingly and access that string:

```
return self.response('{} {}'.format(self.word, text)
```

As you can see, the variable defined in the settings is now available as instance property. Nice, isn't it?

CHAPTER 8

Indices and tables

- genindex
- modindex
- search