Plumeria Documentation

Release 0.1

sk89q

Aug 20, 2017

Contents

1 Considerations **3**

2 Installation **5**
- 2.1 Windows . . . 5
- 2.2 Debian/Ubuntu . . . 5
- 2.3 Mac OS X . . . 6

3 Configuration **7**
- 3.1 Enabling Plugins . . . 7
- 3.2 Plugin Configuration . . . 8
- 3.3 Permissions . . . 8
  - 3.3.1 Bot Administrators . . . 8
  - 3.3.2 Server Administrators . . . 8

4 Plugins **9**
- 4.1 Alias . . . 9
  - 4.1.1 Handling Piping . . . 9
  - 4.1.2 Input . . . 9

5 Writing Plugins **11**
- 5.1 Tutorial . . . 11
  - 5.1.1 Your First Command . . . 11
  - 5.1.2 Fetching URLs . . . 12
  - 5.1.3 Running Your Plugin . . . 13
  - 5.1.4 Adding Configuration . . . 13
  - 5.1.5 Rate Limiting . . . 14

6 Indices and tables **15**

Contents:

CHAPTER 1

Considerations

Plumeria is best run from some sort of dedicated server, either at home or in a proper data center, that is on a fast Internet connection. However, Plumeria will also work on your home computer, though perhaps not as quickly.

Tip: As of writing, students can get a year of free VPS hosting by signing up for the GitHub education pack.

Regardless of your choice, there are some considerations to keep in mind when hosting Plumeria — or any kind of bot.

- The bot will utilize CPU when it is invoked. Most commands won't use any appreciable processing power, but, for example, the image commands could (for brief amounts of time).
- If the system running the bot does not have a good Internet connection, some commands (image downloading-related) could saturate the connection. It also means that the bot will respond slowly.
- The public of the IP of the system running the bot can be exposed. This is merely a consequence of features like image fetching.
- Any sort of bot, application, or website that lets people fetch URLs is susceptible to a problem called Server-Side Request Forgery (SSRF). For example, normally your router's website can't be accessed from outside the Internet, but a program running on your computer would be considered on the inside. Fortunately, Plumeria does have protection in the form of checking where names and addresses resolve to, but this protection doesn't extend to certain plugins that invoke outside programs (like the website capture plugin).

CHAPTER 2

Installation

Windows

1. Install Python 3.5.
2. Download Plumeria (or via Git if you know how to use Git).
3. Double click **install.bat**. If successful, it should say "SUCCESSFUL" at the very end. If not, please file an issue.

Try double clicking **run_default.bat** to run the bot. Since it hasn't been configured yet, nothing will happen, but it should still start.

Debian/Ubuntu

Make sure that you have Python 3.5+ installed. Try running `python3` in shell and see what version is printed.

1. Install system packages:

```
sudo apt-get install python3-pip
```

2. Install:

```
pip3 install virtualenv
git clone https://github.com/sk89q/Plumeria.git plumeria
cd plumeria
python3 -m virtualenv .venv
. .venv/bin/activate
pip install -r requirements.txt
cp config.ini.example config.ini
```

Try running the bot:

```
.venv/bin/python plumeria-bot.py
```

Since it hasn't been configured yet, nothing will happen, but it should still start.

Mac OS X

1. Install Python 3.5.
2. If you haven't installed Git yet, run `git` in Terminal and say yes to the prompt.
3. Open Terminal in the directory where you want to download Plumeria and run these commands:

```
pip3 install virtualenv
git clone https://github.com/sk89q/Plumeria.git plumeria
cd plumeria
python3 -m virtualenv .venv
. .venv/bin/activate
pip install -r requirements.txt
cp config.ini.example config.ini
```

Try running the bot:

```
.venv/bin/python plumeria-bot.py
```

Since it hasn't been configured yet, nothing will happen, but it should still start.

CHAPTER 3

Configuration

Without any extra parameters, Plumeria will use `config.ini` to store configuration. If the file doesn't exist, then Plumeria will create it. Plumeria will update a configuration file with new values on start.

You can edit this configuration file with any text editor. Windows users may want to use an editor like Notepad++ for more editing features like longer undo history.

Rather than `config.ini`, a different configuration file can be used by passing it as an argument:

```
./plumeria-bot.py --config something_else.ini
```

Enabling Plugins

If no plugins are enabled (which is the case if the configuration file has just been created), the only content of the file will be a section to control which plugins are to be loaded:

```
[plugins]
plumeria.plugins.message_ops = True
plumeria.plugins.figlet = True
plumeria.plugins.string = False
plumeria.plugins.webserver = True
plumeria.plugins.memetext = False
plumeria.plugins.imdb = True
plumeria.plugins.gravatar = True
# and so on
```

You can change entries to `True` to turn on the plugin and `False` to turn off the plugin.

If you have no plugins enabled, Plumeria will start but it will sit and do nothing.

Plugin Configuration

Most plugins will have some extra configuration for you to change. However, configuration for a plugin will only be added to your file when the plugin is loaded, so you have to enable the plugin and then (re)start Plumeria to see those settings.

For example, until you actually enable the Discord transport plugin and then run Plumeria, you wouldn't see the following section:

```
[discord]
# The Discord token to login with (overrides password login if set)
token =
# The Discord password to login with
password =
# The Discord username to login to
username =
```

Permissions

Bot Administrators

A few commands are available only to users deemed "bot administrators."

To add yourself as a bot administrator, you will first need to find you Discord user ID. One way to find your user ID is to go to your Discord account settings, "Appearance," and check "Developer Mode" to allow you to right click yourself and choose "Copy ID." Once you have your ID, add it to your configuration file in the following section:

```
[perms]
admin_users = 0000000000000000
```

Server Administrators

Some functions, like creating server aliases, is limited to users deemed "server bot administrators."

Users are identified by having a role named `bot-admin` on a particular server.

CHAPTER 4

Plugins

Documentation for individual plugins is available below.

Alias

The alias plugin allows you to create new commands in a server that run existing commands. Only server bot administrators can create or delete aliases, but anyone can use them.

Aliases can be created using the **`alias`** command:

```
alias hello say hello
```

The first parameter is the name of the command, and the subsequent arguments refer to the command that will be run. The command that will be run must be a valid command! In this example, `say hello` uses the **`say`** command to return a message.

Aliases can be deleted with the **`alias delete`** command:

```
alias delete hello
```

Handling Piping

If you want to pipe commands in the alias command (rather than pipe the output of the alias command), you need to escape the vertical bars with a caret symbol (^):

```
alias rock_song tagtop rock ^| yt
```

Input

If you want your alias to accept input, such as:

```
hello bob
```

You will have to grab that input using the `get input` command (there is a list of "variables" when a command is run, and **`get`** reads the "input" variable that the alias plugin sets). For example, a command to find a cover version of a song on YouTube could be written as:

```
alias findcover get input ^| yt (cover)
```

CHAPTER 5

Writing Plugins

Plugins for Plumeria are written in Python 3.

Plugins can either be standalone Python packages, or they can also be placed into the **plugins** folder. Plugins can be single module files or directories.

Tutorial

Let's start our first plugin! Create a new file in the **plugins** folder and name it `my_first_plugin.py`.

Your First Command

We're going to create a new `fetch` command that downloads the content of a webpage.

```
import re
from plumeria.command import commands, CommandError

@commands.register("fetch", "download", "get page", category="Utility")
async def fetch(message):
    """
    Fetches a webpage.

    Example::

        /fetch http://www.google.com

    """
    q = message.content.strip()
    if not re.search("^https://", re.I): # naive URL checking
        raise CommandError("That's not a valid URL")
    # more to come
```

A command is created by decorating a function with `commands.register()`, which takes a list of aliases. Spaces are acceptable characters in aliases and can be used to create sub-commands. A category is required for the help page so related commands are grouped together to make them easier to find. The actual name of the function doesn't matter, but there can only be one parameter, which is the message object that contains information about what was sent and who sent it.

Docstrings are shown on the help page for commands and they should be formatted in reStructuredText, and example of a docstring can be seen above. Docstrings in Python are surrounded by three quotation marks (“”“) and appear first in a function or object.

Fetching URLs

Because Plumeria is written to be asynchronous, we'll use the `aiohttp` library to make HTTP requests. To improve security, we'll use the `DefaultClientSession` object that comes with Plumeria.

```
import re
from plumeria.command import commands, CommandError
from plumeria.util.http import DefaultClientSession

@commands.register("fetch", "download", "get page", category="Utility")
async def fetch(message):
    """
    Fetches a webpage.

    Example::

        /fetch http://www.google.com

    """
    url = message.content.strip()
    if not re.search("^https://", re.I): # naive URL checking
        raise CommandError("That's not a valid URL")

    with DefaultClientSession() as session:
        async with session.get(url) as resp:
            if require_success and resp.status != 200:
                raise CommandError("HTTP code is not 200; got {}".format(resp.status))
            return await resp.text()
```

We return text directly from the `fetch()` method, which is assumed to be Markdown. If we want to return a message with attachments or other bells and whistles, we would need to return a `plumeria.message.Response` object rather than a string, but that will be explained later.

To see how the HTTP client is used, see the documentation for aiohttp.

Tip: The function above is prefixed with *async*, which means that it is willing to give up control of the currently running “thread” so that something else can run. Python will manage what else will run for you, but you inform Python that you want to give up control by *awaiting* another function. In the example above, the function awaits the `session.request()` function (in the form of the `async with`) because requesting a webpage requires waiting for a remote server to respond, and then further on, the code awaits `resp.text()` because the other's server response must be fully received.

Running Your Plugin

If you have plugins in the **plugins** folder, Plumeria will be able to pick them up, but you still have to tell Plumeria to load your plugin. Open up your configuration file and add the following to the `[plugins]` section:

```
my_first_plugin = True
```

Restart Plumeria and see if your new plugin is loaded in the log, and then try the `.fetch https://github.com/sk89q/Plumeriae` command.

Adding Configuration

Configuration can be declared at the top of a file using `config.create()`, which returns a `plumeria.config.Setting` object that can be used to read the value from the configuration at a later point.

```
from plumeria import config

timeout = config.create("my_first_plugin", "fetch_timeout9", type=int, fallback=4,
                        comment="The maximum amount of time to wait for a webpage to␣
↪load")
```

When the value of `timeout` is required, simply call the object:

```
timeout()
```

> **Warning:** Configuration data can change while Plumeria is running.

We'll integrate this timeout into our command:

```
import re
from plumeria import config
from plumeria.command import commands, CommandError
from plumeria.util.http import DefaultClientSession

timeout = config.create("my_first_plugin", "fetch_timeout9", type=int, fallback=4,
                        comment="The maximum amount of time to wait for a webpage to␣
↪load")

@commands.register("fetch", "download", "get page", category="Utility")
async def fetch(message):
    """
    Fetches a webpage.

    Example::

        /fetch http://www.google.com

    """
    url = message.content.strip()
    if not re.search("^https://", re.I): # naive URL checking
        raise CommandError("That's not a valid URL")

    with DefaultClientSession() as session:
        async with session.get(url, timeout=timeout()) as resp:
```

```
            if require_success and resp.status != 200:
                raise CommandError("HTTP code is not 200; got {}".format(resp.status))
            return await resp.text()
```

Rate Limiting

To reduce abuse, we will want to limit how often the command can be used. There are two types of rate limits:

- A command cost, which is used to determine how many commands can be chained together
- A rate limit, which simply controls the rate of calls

By default, all commands have a cost of 1.0. Commands that have minimal CPU and network impact should have lower costs. Costs can be adjusted when registering the command:

```
@commands.register("fetch", "download", "get page", category="Utility", cost=1.0)
```

For our fetch command, we won't adjust the cost.

However, we do want to reduce how frequently the command can be used, so we'll apply a rate limit. Rate limits are per-user, per-channel, and per-server. Rate limits are simply added by applying a `@rate_limit()` decorator.

```
from plumeria.util.ratelimit import rate_limit

@commands.register("fetch", "download", "get page", category="Utility")
@rate_limit()
async def fetch(message):
    # etc.
```

Rate limits can be adjusted by changing burst size and fill rate:

```
@commands.register("fetch", "download", "get page", category="Utility")
@rate_limit(burst_size=10, fill_rate=0.5)
async def fetch(message):
    # etc.
```

Warning: `@rate_limit()` must appear **after** the command registration.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search