Assistant Documentation

Release 0.1.0

Jewel Mahanta

May 20, 2017

Contents

Assistant is a discord assistant bot. It is a self bot, which means only you can use its commands. Assistant runs on top of the Assistant framework.You can extend it by writing your own modules/add-ons or download from a selection of awesome community written plugins.

The default plugin comes with these commands, (`modules/default.py`):

- `evaljs`: Evaluate Javascript commands
- `evalpy`: Evalualte Python commands
- `sh`: Run shell commands

CHAPTER 1

Contents

Installation

It is recommended to run Assistant using docker. It saves you the hassle of manually installing dependencies.

Docker

1. Install docker
2. Install git
3. Clone the Assistant github repository:

```
git clone https://github.com/lap00zza/Assistant.git
cd Assistant
```

4. Create an environment variable named `DOCKER_TOKEN` and set it to you own token. To find your token, open discord and press `ctrl+shift+i`. Then go to the **Applications** tab and find the **Key** named `token`. This is your token.
5. Now build the image and run.

```
docker-compose build
docker-compose up
```

Normal

1. Install git
2. Install Python 3.6+
3. Install NodeJs 6.7.0+ (*for evaljs*)
4. Clone the Assistant github repository:

```
git clone https://github.com/lap00zza/Assistant.git
cd Assistant/bot
```

5. Install the dependencies

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

6. Create an environment variable named `DOCKER_TOKEN` and set it to you own token. To find your token, open discord and press `ctrl+shift+i`. Then go to the **Applications** tab and find the **Key** named `token`. This is your token.

7. Now run.

```
py run.py
```

API Reference

The complete API reference for Assistant. If you want to make your own modules, you should definitely read this.

Assistant

class `assistant.`**`Assistant`**(***kwargs*)

`add_event_listener`(*callback*, *event=None*)

Add a new event listener.

Parameters

- **`callback`** (*`coroutine`*) – The coroutine to call when this event is triggered. The callback **must** be a coroutine.
- **`event`** (*`[Optional] str`*) – The name of the event for which the callback is being registered. If a name is not given, `callback.__name__` will be used.

Raises `TypeError` – The callback is not a coroutine.

`add_module`(*module*)

Add a new module to Assistant.

Parameters **`module`** – The module to add.

Notes

This function is called from the `load` function of the module.

`event_listener`(*event=None*)

This function is a decorator. It is a convenience wrapper for *`add_event_listener()`*.

Parameters **`event`** (*`[Optional] str`*) – The name of the event to listen to. This can be a custom event or a standard discord event .

Examples

```
@my_assistant.event_listener()
async def on_message(message):
    print(message.content)

@my_assistant.event_listener(event="on_ready")
async def my_awesome_function():
    print("Awesome! We are ready to roll.")

@my_assistant.event_listener(event="my_custom_event")
async def my_awesome_function():
    print("Awesome! We are ready to roll once again.")
```

load_module(*name*)

Load a module. Modules are collection of commands and custom event listeners. They are stateful. Sample modules can be found in `/bot/modules` directory. **All modules must have a load function.**

Parameters **name** (*str*) – The name of the module to load. See Notes for clarification.

Raises

- `AttributeError` – Module does not have a load function.
- `TypeError` – You are trying to access a module using relative path. See Notes for correct name convention.

Notes

```
+---run.py (or any file with run)
|
+---subdirectory---+---hello.py
                   |
                   +---hello_again.py
```

Modules should be placed in a sub-directory from where `run()` is used. For example, (*using the above diagram as reference*) if the name of your module file is *hello.py* and it is placed inside subdirectory then run.py will look something like this:

```
from assistant import Assistant
my_assistant = Assistant()
# Remember, no need to append .py
my_assistant.load_module("subdirectory.hello")
my_assistant.run()
```

remove_event_listener(*callback*, *event=None*)

Remove an event listener.

Parameters

- **callback** (*coroutine*) – The coroutine to remove.
- **event** (*[Optional] str*) – The name of the event listener to remove. If a name is not given, `callback.__name__` will be used.

Common

class assistant.**Common**(***kwargs*)

add_command(*cmd*)
Add a command to the commands list.

Parameters **cmd** (Command) – The command to add.

command(**args*, ***kwargs*)
This is a decorator. It invokes *command()* and adds the command to the commands list using *add_command()*

Command

class assistant.**Command**(*name*, *callback*, ***kwargs*)
Represents a command.

name
str – The name of the command.

callback
coroutine – The coroutine to invoke when the command is used.

description
str – A short description of the command.

assistant.**command**(*name=None*, ***kwargs*)
This function is a decorator. It is used to generate a *Command* object.

Parameters **name** (*[Optional] str*) – The name of the command. If a name is not provided then the functions name func.__name__ is used instead.

Example

```
# you don't always have to specify a name.
# Although specifying a name can be helpful.
@command()
async def my_command(ctx):
    await ctx.send_message(message.channel, "Hello")

@command(name="ping")
async def _ping(ctx):
    await ctx.send_message(message.channel, "pong")
```

Raises TypeError – The callback function is not a coroutine.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index