The ShockValueBot 2 Specification

Michael Gohde

Initial Draft 2019-07-10

Table of Contents

1. Introduction

1.1 Purpose

This document exists to describe the interface by which ShockValueBot 2 plugins are implemented. In it, there exists a detailed explanation of how plugins are loaded, instantiated, and called throughout the execution of a ShockValueBot instance.

2. The Plugin API

2.1 Definition of a ShockValueBot 2 plugin

A ShockValueBot 2 plugin consists of a Python 3 module implementing the following:
1. A class conforming to the specification outlined in the following sections

2. A global variable named INST pointing to the class defined in part 1.

2.2 Definition of a ShockValueBot 2 plugin class

ShockValueBot plugins are instantiated and treated as Python objects. As such, they must feature a set of common methods and data structures meeting the following specification:

Common Methods	
__init__(self, **logger**, **client**, **pluginSet**)	A plugin class constructor must accept the following arguments: 1. **logger** – a function reference to be used when logging events locally. The logger function has the following signature: logger(**name**, **message**) Where **name** is the name of the plugin, and **message** is the data to log. 2. **client** – a discord.py client object corresponding to the currently running client in use by ShockValueBot. 3. **pluginSet** – an internally defined instance of the Plugins class. This represents the set of correctly loaded and instantiated plugins currently available to ShockValueBot.
canHandle(self, **cmd**, **fullmessage)**	This method is used to determine whether a given plugin is either: 1. Capable of handling a given command passed

	to ShockValueBot or 2. Able to provide a response to some pertinent information within a given message. It is expected that the method will return True or False corresponding to whether or not the plugin in question can meet the above criteria given the following arguments: 1. **cmd –** a lowercase, whitespace-stripped token representing the first “word” in a given message. Standalone commands are generally of the form *!command* 2. **fullmessage** – A discord.py message object representing the message received.
async handle(self, **message**, **channel**, **args**, **cmd**)	This method is used to directly respond to a given message assuming that the conditions provided to cahHandle() are met and it returns True. The following arguments have been provided to enable a reasonable degree of flexibility in terms of plugin implementation: 1. **message** – a discord.py message object corresponding to the message containing the command. 2. **channel** – a discord.py channel object corresponding to the channel in which the message was sent. 3. **args** – The contents of the received message after having the command passed to canHandle() left stripped. Ie. the remainder of the message after the command token. 4. **cmd** – The command token, a lowercase whitespace-stripped token representing the first “word” in a given message.
getHelp(self)	It is expected that this method will return a string containing help information regarding the correct usage of the plugin. For example, if the plugin

	implements a given command, the string returned should follow the form: *!command help related to the command or its arguments*
async update(self)	This optional method allows a plugin to perform regularly scheduled computation or updates. As presently implemented, ShockValueBot will call this method in every plugin in which it is implemented approximately once every two seconds.

Common Data	
NAME	Each ShockValueBot plugin class must contain a field representing its name after initialization.

3. Example Plugin

The following is a listing shows a plugin capable of greeting a user when they send a message starting with “!hello”. It does not implement the optional update() method as it is not necessary to do so for this specific use case.

```
import discord
import asyncio

class Hello:
    def __init__(self, logger, client, pluginSet):
        self.NAME="hello"
        logger(self.NAME, "Initializing hello plugin")

    def canHandle(self, cmdStr, fullMessage):
        return cmdStr.lower()=="!hello"

    async def handle(self, message, channel, args, cmd):
        await channel.send('Hello {0.author.mention}'.format(message))
        return True

    def getHelp(self):
        return "!hello greets a user."

INST=Hello
```