MERK Plugin Development Guide

Version .022

Summary

MERK plugins are Python 3 classes that inherit from a base class, named “Plugin”, that is imported from the MERK application. They are loaded from a subdirectory in the main directory where MERK stores settings and scripts, **`.merk/plugins`**, located in the user’s home directory.

A basic plugin looks something like this:

```
from merk import *

PACKAGE = "Dan’s Plugins"

class DumpPlugin(Plugin):
     NAME = "Dump Plugin"
     VERSION = "1.0"
     DESCRIPTION = "Displays all IRC network traffic"

     def line_in(self,data):
          print("<- "+data)

     def line_out(self,data):
          print("-> "+data)
```

Writing MERK plugins

Terminology

A single Python 3 module file intended to be loaded by MERK is called a *package*. A **`Plugin`**-derived class (or an instance of that class) contained in a package is called a *plugin*. MERK packages can contain multiple plugins.

The difference between a package and a plugin, however, is academic and will only be used in this document, and even then, sparingly. End users will probably call both packages and plugins "plugins" and that is fine.

Plugin requirements and optional features

MERK plugins are Python 3 classes that must meet four basic requirements:

- The class must inherit from the base class **`Plugin`**
- The class must have a **`NAME`** attribute string
- The class must have a **`VERSION`** attribute string
- The class must have at least one event method (see page 7)

There are three optional features that you can add to a plugin:

- A class attribute string named **`DESCRIPTION`**
- A string in the “root” of the plugin’s module named **`PACKAGE`**
- A 48x48px PNG[1] image file with the same file name as the module
- A 25x25px PNG image with the same name as the plugin class name with the file extension “.png”

To get access to the first requirement, simply import **`Plugin`** from **`merk`**. You can either explicitly import Plugin:

```
from merk import Plugin
```

or use a “splat” and import it implicitly:

```
from merk import *
```

Either way will work; however, even if the "splat" method is used, only **`Plugin`** will be imported into the current namespace[2].

The second two requirements, the **`NAME`** and **`VERSION`** attributes, are used by MERK to display the plugin in the client. These should be class attributes, rather than instance attributes[3]. **`NAME`** should be set to the name of the plugin, a short descriptive string; it must contain at least one character that is not whitespace. **`VERSION`** should be the version number of the

1 https://en.wikipedia.org/wiki/Portable_Network_Graphics
2 https://docs.python.org/3/tutorial/classes.html#python-scopes-and-namespaces
3 https://www.geeksforgeeks.org/class-instance-attributes-python/

plugin; if no version number is required, this attribute can be set to a blank string, but it *must* exist. A third attribute, **`DESCRIPTION`**, is optional; this should be a short string that describes what the plugin does.

The last requirement is for the plugin to have at least one event method. An event method is a plugin class method that MERK will execute when a specific event occurs.

If you add a string named **`PACKAGE`** to the plugin's package (the module containing the class), this string value will be used to display the plugin in MERK's "Plugins" menu.

If a 48x48 pixel PNG image is in the same directory as the module, and has the same name as the module (with the exception of the file extension), this image will be used as the module's "icon" in MERK's "Plugins" menu.

If a 25x25 pixel PNG image is in the same directory as the module, and has the same name as the plugin *class* (with the file extension ".png"), this image will be used as the plugin's icon in MERK's "Plugins" menu.

> For example, say we have a plugin. It's in a module named "**`bleep.py`**", and the **`Plugin`** class in the module is named **`MyPlugin`**. If you want to display a module icon, you would name your 48x48px PNG image "**`bleep.png`**". If you want to display a plugin icon, you would name your 25x25px PNG image "**`MyPlugin.png`**". Remember, file names are case-sensitive. Place both of these images in the same directory as **`bleep.py`**, and you're done!

For convenience, you can put your plugin in its own directory in **`.merk/plugins`**, named whatever you wish. Just like with normal Python modules, sub directories need a **`__init__.py`** file in order to be detected by MERK.

For best practice, each plugin should exist either as a single Python file, or in a single directory, with no subdirectories. This is not enforced in any way, however.

`Plugin` class and instance features

The **`Plugin`** class contains some built-in methods and the **`irc`** attribute to make interacting with the MERK client, any connected IRC servers, and even other **`Plugins`** easy.

The `irc` attribute

The **`irc`** attribute is part of the **`Plugin`** class, and is the way plugins interact with IRC servers. It is integrated as an instance attribute[4] of the class. This attribute/object is the instance of the Twisted IRC client[5] that MERK is using for communication with the IRC server. Anything you can normally do with the Twisted IRC client, you can use **`irc`** for. Whenever an event method is triggered, the **`irc`** attribute is set to the Twisted instance that is connected to the server event that triggered the method. So, for example, if you wanted to write a method that forwards all private messages to another user with the nickname “OtherNick”, you could write:

```
def private(self,user,message):
     self.irc.msg("OtherNick",user+": "+message)
```

If the **`irc`** attribute is unavailable (due to a disconnection, error, or other reason), the attribute’s value is set to **`None`**. For help on how to use the **`irc`** attribute, take a look at the documentation for the Twisted IRC Client[6].

One additional feature MERK built into the **`irc`** attribute is a new attribute, **`client_id`**. The **`client_id`** attribute is a string that is unique for each connection. This way, plugins can tell the difference between **`irc`** attributes that may be connected to the same IRC server.

Built-in methods

The **`Plugin`** class contains a few built-in methods for interacting with MERK. These can all be overloaded, if you wish. Many of these methods will return **`True`** if the call succeeded (that is, its purpose was completed successfully) or **`False`** if the call failed (the method's purpose was *not* completed, for whatever reason). For example, **`.writeWindow()`** will return **`False`** if the window named in the arguments does not exist, or if MERK is not connected to an IRC server.

Most methods can only interact with windows that are associated with the IRC connection that created the event they are called from. So, if MERK is connected to two servers, Server A and Server B, and is in the "#merk" channel in both servers, if the connection to Server A triggers the **`join()`** event method, most built-in methods called in that event method will only be able to interact with the console for Server A and the "#merk" channel window on Server A, and not with any of the windows associated with Server B.

4 https://docs.python.org/3/tutorial/classes.html#class-and-instance-variables
5 twisted.words.protocols.irc.IRCClient
6 https://twistedmatrix.com/documents/current/api/twisted.words.protocols.irc.IRCClient.html

writeConsole

Arguments	**text** (string)
Returns	**True** if the call succeeded, **False** if the call failed
Description	Prints **text** to the window associated with the current server connection, called the “console”. The text to be printed may be stylized with HTML.

writeWindow

Arguments	**name** (string), **text** (string)
Returns	**True** if the call succeeded, **False** if the call failed
Description	Prints **text** to the window named **name**. For example, to write to the window for the chat channel "#example", you would set name to equal **#example**. This method can only write to windows associated with the current server connection. The text to be printed may be stylized with HTML.

command

Arguments	**text** (string)
Returns	**True** if the call succeeded, **False** if the call failed
Description	Executes **text** as if it were a command typed into the MERK server console window associated with the current server connection.

Event methods

MERK executes the event method of every plugin that contains that event method when a specific event occurs (like the reception of a public message, a private message, or a notice message). In short, MERK plugins are event-driven[7].

load

Arguments None

Description Triggered every time MERK loads the plugin.

unload

Arguments None

Description Triggered every time MERK unloads the plugin (that is, when MERK closes).

line_in

Arguments **`data`** (string)

Description Triggered every time MERK receives data from an IRC server. **`data`** contains the data sent to the client.

line_out

Arguments **`data`** (string)

Description Triggered every time MERK sends data to an IRC server. **`data`** contains the data sent to the server.

public

Arguments **`channel`** (string), **`user`** (string), **`message`** (string)

Description Triggered every time MERK receives a public (channel) message . **`channel`** contains the name of the channel the message was sent to, **`user`** contains the nickname and hostmask (in the format **`NICKNAME!USERNAME@HOST`**) of the user that sent the message, and **`message`** contains the message sent.

private

Arguments **`user`** (string), **`message`** (string)

Description Triggered every time MERK receives a private message . **`user`** contains the nickname and hostmask (in the format **`NICKNAME!USERNAME@HOST`**) of the user that sent the message, and **`message`** contains the message sent.

7 https://en.wikipedia.org/wiki/Event-driven_programming

action

Arguments **target** (string), **user** (string), **message** (string)

Description Triggered every time MERK receives a CTCP[8] action message . **target** contains the name of the target (channel or user) the message was sent to, **user** contains the nickname and hostmask (in the format **NICKNAME!USERNAME@HOST**) of the user that sent the message, and **message** contains the message sent.

notice

Arguments **target** (string), **user** (string), **message** (string)

Description Triggered every time MERK receives a notice message . **target** contains the name of the target (channel or user) the message was sent to, **user** contains the nickname and hostmask (in the format **NICKNAME!USERNAME@HOST**) of the user that sent the message, and **message** contains the message sent.

join

Arguments **channel** (string), **user** (string)

Description Triggered every time MERK "sees" someone join a channel . **channel** contains the name of the channel, **user** contains the nickname of the user that joined.

part

Arguments **channel** (string), **user** (string)

Description Triggered every time MERK "sees" someone leave a channel . **channel** contains the name of the channel, **user** contains the nickname of the user that left.

connect

Arguments None

Description Triggered every time MERK completes registration on an IRC server.

tick

Arguments **uptime** (integer)

Description Triggered once a second while MERK is connected to an IRC server. **uptime** contains the number of seconds the current connection has been connected to the IRC server.

8 https://en.wikipedia.org/wiki/Client-to-client_protocol