Ərk Plugin Development Guide

Version 0.510

Ərk Plugins

An Ərk plugin consists of a Python module containing a class, which contains one or more callable methods; the class inherits from a class that comes with Ərk, `Plugin`.

Installing plugins

To install an Ərk plugin, copy it into the `plugins` directory in Ərk's installation directory. Ərk plugins follow the same rules that Python modules do, and can be contained in directories, span multiple files, etc.

Plugin implementation

All plugins must import the `Plugin` class, which can be imported from the `erk` module. Other objects which make writing plugins easier can also be imported:

- `ERK` – An object for displaying and retrieving information from the Ərk IRC client.
- `IRC` – An object for interacting with IRC servers.
- `HEAP` – A shared dict that all plugins have access to.
- `Message` – An enum defining different message types.
- `Event` – An enum defining different event types.

All of these, along with the Plugin class, can be imported with `from erk import *`.

The `Message` enum

This is used by Ørk to categorize the four types of messages:

1	`Message.PUBLIC`	Public messages sent to an IRC channel.
2	`Message.PRIVATE`	Private messages sent to the Ørk IRC client.
3	`Message.ACTION`	CTCP action messages sent either to a channel or directly to the client.
4	`Message.NOTICE`	Notice messages sent either to a channel or directly to the client.

Table 1: `Message` types

The `Event` enum

This is used by Ørk to categorize the fifteen types of events that will be passed to plugins:

1	`Event.LINE_IN`	Triggered whenever a message is sent to the IRC server.
2	`Event.LINE_OUT`	Triggered whenever a message is received from the IRC server.
3	`Event.CONNECT`	Triggered when Ørk connects to an IRC server.
4	`Event.DISCONNECT`	Triggered when Ørk disconnects from an IRC server.
5	`Event.REGISTERED`	Triggered when Ørk has finished registering with an IRC server.
6	`Event.MOTD`	Triggered when Ørk receives an IRC server's message of the day.
7	`Event.NICK`	Triggered when a user changes their nickname.
8	`Event.PART`	Triggered when a user leaves a channel.
9	`Event.JOIN`	Triggered when a user joins a channel.
10	`Event.KICK`	Triggered when a user is kicked from a channel.
11	`Event.TOPIC`	Triggered when a channel's topic changes.
12	`Event.MODE`	Triggered when a user or channel mode is set.
13	`Event.INVITE`	Triggered when Ørk is invited to an IRC channel.
14	`Event.QUIT`	Triggered when a user quits IRC.
15	`Event.ERROR`	Triggered when an error is received.
16	`Event.TICK`	Triggered once a second.

Table 2: `Event` types

The `Plugin` class

This is the base class of all Ørk plugins.

Attributes

The **`Plugin`** class features three class attributes: **`name`**, **`version`**, and **`author`**. These are set, by default, to **`"No name"`**, **`"1.0"`**, and **`"Unknown"`**, respectively. The values for these attributes are used to display plugin information in the Ørk client. They should be set to appropriate values in the plugin's **`__init__`** constructor.

There are two more class attributes that can optionally be set: **`website`** and **`source`**. Both of these attributes are set to **`None`** by default. Set **`website`** to a URL string to be displayed in the Ørk client as the plugin's website. Set **`source`** to a URL string to be displayed in the Ørk client as a link to the plugin's source code. These attributes, if desired, should also be set in the plugin's **`__init__`** constructor.

Plugins can be displayed in the Ørk client with a custom icon; if no icon accompanies the plugin, a default one will be used. Icons should be in the portable network graphic format (PNG), and be placed in the same location the plugin file is. The icon's file name should be the same as the file name of the plugin, with the icon's file extension changed to ".png". So, the icon for a plugin with the filename of **`MyPlugin.py`** should be named **`MyPlugin.png`**.

To be displayed properly, the plugin's icon should have a width and height of 25 pixels.

Methods

Plugins can contain (but don't have to contain) one or more callable methods.

`load()`

This method is called when Ørk first loads the plugin. If the plugin has been disabled via the Ørk GUI, this method will be called when the plugin is re-enabled.

`unload()`

This method is called when Ørk is shutting down. If the plugin is disabled via the Ørk GUI, this method will be called when the plugin is disabled.

`handle_input(text)`

This method is called whenever text is entered into the Ørk client and return is pressed. **`text`** is an **`INPUT()`** object. If any plugin's **`handle_input()`** method returns **`True`** (or a value that Python would consider **`True`**), Ørk will stop processing the input text as soon as the plugins finish execution; this allows plugins to implement their own commands. If used improperly, this can prevent Ørk from functioning.

For example, if a plugin with the following source code is loaded:

```
from erk import *

class BreakTheClient(Plugin):
  def handle_input(self,text):
    return True
```

This will break user input. Any time a user enters text into the client, the plugin will stop all text processing as soon as the text is entered; Ərk will not send any entered text as messages.

The **`handle_input()`** method should only return **`True`** if no further text processing is desired.

The `INPUT()` object

`INPUT(text=String,window=String,console=Boolean)`

`text` is a string containing the the text that was entered into the client. **`window`** is the name of the window the user entered the text into (this will be either a channel or user name, or a server hostname). **`console`** will be set to **`True`** if the text was entered into a server console window, or **`False`** if the text was entered into a channel or user window.

```
# Print if the input came from a console window
if text.console:
  print("The input came from " + text.window + "'s console")

# Print the input
print(text.text)
```

*Table 3: Example usage of an **`INPUT()`** object*

If used as a **`str()`**, **`INPUT()`** will return **`.text`**.

```
input_text = INPUT("hello world","#channel",False)

# Prints "hello world" to the console
print(input_text)
```

*Table 4: Using an **`INPUT()`** object as a **`str()`***

`handle_message(msg)`

This method is called whenever the Ərk IRC client receives a PRIVMSG or NOTICE from an IRC server. **`msg`** is a **`PRIVMSG()`** object.

The `PRIVMSG()` object

`PRIVMSG(type=Message,target=String,sender=String,message=String)`

`type` is a **`Message`** object containing the type of message (see *The Message enum* on page 4). **`target`** is the entity the message was sent to (either a channel name, for public messages, or the nickname the Ørk client is using for private messages). **`sender`** is information about the user that sent the message; this is a string that takes the form ***`nickname!username@host`***. **`server`** contains the IP or hostname of the server the message came from, and **`port`** contains the server port the client is connected to.

A **`PRIVMSG()`** object can be accessed like a normal Python object with four attributes (**`.type`**, **`.target`**, **`.sender`**, and **`.message`**):

```
# Print the message sender and contents
print(PRIVMSG.sender + ": "+ PRIVMSG.message)

# Print where the message was sent to
print(PRIVMSG.target)

# Print "hello world!" if the message is a public messages
if(PRIVMSG.type == Message.PUBLIC):
     print("hello world!")
```

*Table 5: Example usage of a **`PRIVMSG()`** object*

If used as a **`str()`**, a **`PRIVMSG()`** object will return **`.message`**:

```
msg = PRIVMSG(Message.PUBLIC,"#channel","bob!bob@host.com","Hello!")

# This will print "Hello!" to the console
print(str(msg))
```

*Table 6: Using a **`PRIVMSG()`** object as a **`str()`***

If used in a comparison operation, **`PRIVMSG()`** will use **`.type`** as its value:

```
# Display whether a given PRIVMSG() represents a public messages
if msg == Message.PUBLIC:
     print("It's a public message!")
else:
     print("It's not a public message!")
```

*Table 7: Using **`PRIVMSG()`** in comparison operations*

`handle_event(event)`

This method is called whenever the Ørk IRC client triggers an event; events occur when certain messages are received from the IRC server. **`event`** is an **`EVENT()`** object.

The `EVENT()` object

```
EVENT(type=Event,data=Dict)
```

`type` is an **`Event`** object containing the type of event (see *The Event enum* on page 4). **`data`** is a dict that contains different information depending on the type of event.

An **`EVENT()`** object can be accessed as a normal Python object with two attributes (**`.type`** and **`.data`**):

```
# Display if the event is a connection event
if EVENT.type==Event.CONNECT: print("It's a connection event!")

# Display the contents of the .data attribute
for key in EVENT.data:
  print(EVENT.data[key])
```

*Table 8: Example usage of an **`EVENT()`** object*

If used like a **`dict()`**, an **`EVENT()`** object will behave like a **`dict()`**:

```
event = EVENT(Event.LINE_IN,{ data: "PING :888CC663" })

# Prints "PING :888CC663"
if "data" in event:
  print(event["data"])
```

*Table 9: Using an **`EVENT()`** object as a **`dict()`***

If used in a comparison operation, **`EVENT()`** will use **`.type`** as its value:

```
# Display whether a given EVENT() represents a connection event
if event == Event.CONNECT:
     print("It's a connection event!")
else:
     print("It's not a connection event!")
```

*Table 10: Using **`EVENT()`** in comparison operations*

`EVENT()` object contents

Each different `Event` type will have different pass different data to the emitted `EVENT()` object. This data is stored in the `EVENT()`'s `.data` dict. All values, unless otherwise noted, are `String`s.

`Event`	Trigger	`.data` keys	
`LINE_IN`	A message is received from the IRC server	`line`	The line of data sent to the client from the server.
`LINE_OUT`	A message is sent to the IRC server	`line`	The line of data sent to the server from the client.
`CONNECT`	Connection to an IRC server	`server`	The IP or hostname of the server Ørk connected to.
		`port`	**Integer.** The port number Ørk is connected to on the server.
`DISCONNECT`	Disconnection from an IRC server	`reason`	A string describing what caused the disconnection.
`REGISTERED`	Server registration completed	`nickname`	The Ørk client nickname.
`MOTD`	Receipt of server's message of the day	`motd`	**List.** All lines in the server's MOTD.
`NICK`	User changes their nickname	`old`	The nickname the client was using prior to the change.
		`new`	The user's new nickname.
`PART`	User leaves a channel	`user`	The user who left the channel.
		`channel`	The channel the user left.
`JOIN`	User joins a channel	`user`	The user who joined the channel.
		`channel`	The channel the user joined.
`KICK`	User is kicked from a channel	`kicker`	The user who initiated the channel kick.
		`target`	The user being kicked.
		`channel`	The channel the kick occurred in.
		`message`	The reason for the kick, if any.

Event	Trigger	.data keys	
TOPIC	User sets a channel topic	user	The user who set the channel topic.
		channel	The channel whose topic was topic was set.
		topic	The new channel topic.
MODE	User sets a channel or user mode	user	The user who set or unset the mode.
		target	The user (or channel) the mode is being applied to.
		set	**Boolean.** Set to **True** if the user is setting a mode, or **False** if the user is un-setting a mode.
		modes	The mode(s) being set or unset by the user.
		arguments	**List.** Any arguments passed along with the mode.
INVITE	Channel invitation is received from a user	user	The user that sent the channel invitation.
		target	Who the channel invitation was sent to.
		channel	The name of the channel the client is being invited to.
QUIT	User quits IRC	user	The user quitting IRC.
		message	The user's parting message, if any.
ERROR	Server sends an error message	message	The error message sent by the server.
TICK	Triggered once a second	uptime	**Integer.** How long the Ørk client has been running, in seconds.

The `HEAP` object

The **`HEAP`** object is a shared dict that all plugins have access to. This allows for limited cross-plugin communication and the potential for data persistence. When Ørk starts up, **`HEAP`** is created as an empty dict (**`HEAP = {}`**). Other than creating **`HEAP`**, the Ørk application has no interactions with **`HEAP`**; it is designed to be used by plugins *only*.

The `ERK` object

The **`ERK`** object is an interface for the Ørk GUI. It can be used to display text in the client. It's a singleton object that every plugin can access, and features XX methods.

`active(message=String,log=False)`

```
# Prints "Hello, world!" to the current active window
ERK.active("Hello, world!")

# Prints "Hello, world!" and makes sure it is written to the log
ERK.active("Hello, world!",True)
```

The **`active()`** method prints text to the current active window in the Ørk client; it prints nothing if there is no active window. **`message`** is the text to display; **`log`** sets whether the printed text is to be written to Ørk's log. **`log`** is set to **`False`** by default. Text that is not logged will not be saved when Ørk writes channel/private message logs.

`console(message=String,log=False)`

```
# Prints "Hello, world!" to the console
ERK.console("Hello, world!")

# Prints "Hello, world!" and makes sure it is written to the log
ERK.console("Hello, world!",True)
```

The **`console()`** method prints text to the current server console. **`message`** is the text to display; **`log`** sets whether the printed text is to be written to Ørk's log. **`log`** is set to **`False`** by default. Text that is not logged will not be saved when Ørk writes console logs.

`channel(name=String,message=String,log=False)`

```
# Prints "Hello, world!" to the chat window for #erk
ERK.channel("#erk","Hello, world!")

# Prints "Hello, world!" and makes sure it is written to the log
ERK.channel("#erk","Hello, world!",True)
```

The **`channel()`** method prints text to a specific channel or user chat window. **`name`** is the name of the channel or user window to print to, **`message`** is the text to display, and **`log`** sets whether the printed text is to be saved to Ørk's log. **`log`** is set to **`False`** by default. Text that is not logged will not be saved when Ørk writes channel logs.

textWindow(title=String,contents=String)

```
# Displays "Hello world!" in a window
ERK.textWindow("Greeting","Hello world!")

# Create a text window
window = ERK.textWindow("Text Window")

# Write text to the new text window
window.write("Hello, world!")

# Get text window contents
print(window.contents())

# Clear text window
window.clear()
```

The **`textWindow()`** method creates a window in Ərk and displays text on it. The method will return a window object that can be saved for later use; the window is a **`QMainWindow`** (a type of Qt widget) with three methods:

- **`write(String)`** – Writes text to the window
- **`contents()`** – Returns the text contents of the window
- **`clear()`** – Clears all text from the window

fetch(id=String)

```
# Get the connection's ID
id = IRC.id()

# Fetch another IRC connection with the id
other_IRC = ERK.fetch(id)

# Print the other connection's nickname
print(other_IRC.nickname())
```

The **`fetch()`** method takes a connection ID (as returned by the **`IRC`** object's **`id()`** method) as an argument and returns an instance of the **`IRCConnectionObject`** for the server connection with that ID. If the ID is invalid or not found, **`fetch()`** returns **`None`**.

fetchAll()

```
# Get all connections
connections = ERK.fetchAll()

# Print each server the client is connected to
for server in connections:
  print(server.server()+":"+str(server.port()))
```

The **fetchAll()** method returns a list of **IRCConnection** objects, one for each IRC server that Ərk is currently connected to.

The `IRC` object

The **`IRC`** object is an interface for the Ərk IRC client. If Ərk is connected to multiple IRC servers, the **`IRC`** object works as an interface for the connection that triggered the plugin execution. For example, let's assume that Ərk is connected to two IRC servers, **`irc.efnet.org`** and **`irc.freenode.net`**. Below is a plugin that echoes back any private messages sent to it. If a private message is sent to the client from **`irc.freenode.net`**, the call to **`IRC.privmsg()`** will send the echo message *only* to the **`irc.freenode.net`** connection.

```
from erk import *

class Echo(Plugin):
  def handle_message(self,incoming):
    if incoming==Message.PRIVATE:
      nick = incoming.sender.split('!')[0]
      IRC.privmsg(nick,incoming.message)
```

The **`IRC`** object is an instance of the **`IRCConnectionObject`**, and internal object used to interact with an IRC server connection maintained by Ərk. The object will always be set to interact with the server connection that triggered the plugin's execution. Access to other server connections is possible with the **`ERK`** object's **`fetch()`** and **`fetchAll()`** methods.

`send(data=String)`

```
# Joins a channel named #example
IRC.send("JOIN #example")

# Sends a message to #example
IRC.send("PRIVMSG #example Hello, world!")
```

The **`send()`** method sends a raw, unaltered message to the IRC server. This allows to use functionality not handled by other **`IRC`** methods.

`privmsg(target=String,message=String,display=True)`

```
# Sends a public message to #example
IRC.privmsg("#example","Hello, world!")

# Sends a message to #example without displaying it in the client
IRC.privmsg("#example","Hello, world!",False)
```

The **`privmsg()`** method sends a chat message to the IRC server. **`target`** is the recipient of the message (a channel or user name), and **`message`** is the contents of the message. This message will displayed in Ərk the same as it would be displayed if sent via the GUI. To prevent the message from being displayed in Ərk, pass **`False`** as a third argument to the method.

notice(target=String,message=String,display=True)

```
# Sends a public notice to #example
IRC.notice("#example","Hello, world!")

# Sends a notice to #example without displaying it in the client
IRC.notice("#example","Hello, world!",False)
```

The **notice()** method sends a notice message to the IRC server. **target** is the recipient of the message (a channel or user name), and **message** is the contents of the message. This message will displayed in Ørk the same as it would be displayed if sent via the GUI. To prevent the message from being displayed in Ørk, pass **False** as a third argument to the method.

action(target=String,message=String,display=True)

```
# Sends a public CTCP action to #example
IRC.action("#example","says hello!")

# Sends a CTCP action to #example without displaying it
# in the client
IRC.action("#example","says hello!",False)
```

The **action()** method sends a CTCP action message to the IRC server. **target** is the recipient of the message (a channel or user name), and **message** is the contents of the message. This message will displayed in Ørk the same as it would be displayed if sent via the GUI. To prevent the message from being displayed in Ørk, pass **False** as a third argument to the method.

join(channel=String,key=None)

```
# Joins #example
IRC.join("#example")

# Joins #example with a key
IRC.join("#example","my_channel_key")
```

The **join()** method causes the IRC client to join a channel. If the channel requires a key to join, pass the channel key as a second argument.

part(channel=String,message=None)

```
# Parts #example
IRC.part("#example")

# Parts #example with a message
IRC.part("#example","See you later!")
```

The **part()** method causes the IRC client to leave a channel. An optional message can be passed as a second argument to part with a message.

server()

```
# Get the IP/hostname of the IRC server
server = IRC.server()
```

The **server()** method returns the IP or hostname of the IRC server as a string.

port()

```
# Get the port on the IRC server
port = IRC.port()
```

The **port()** method returns the port on the IRC server as an integer.

nickname()

```
# Get the current nickname
nickname = IRC.nickname()
```

The **nickname()** method returns the nickname currently in use as a string.

id()

```
# Get the connection's ID
id = IRC.id()
```

The **id()** method returns the IRC connection's identification token as a string. Each IRC connection has a unique ID which is generated randomly when the connection is created. The ID returned from this method can be used with the **ERK** object's **fetch()** method, allowing access to any connection the Ərk client is using from any other connection.

Example plugins

Echo

This plugin will cause Ərk to echo any private message sent to it back to the sender.

```
 1 from erk import *
 2
 3 class Echo(Plugin):
 4   def __init__(self):
 5     self.name = "Echo"
 6     self.version = "1.0"
 7     self.author = "Dan Hetrick"
 8
 9   def handle_message(self,msg):
10
11     # Execute if we're received a private message
12     if msg==Message.PRIVATE:
13
14       # Extract the sender's nickname from msg.sender
15       nickname = msg.sender.split('!')[0]
16
17       # Send the senders message back to them
18       IRC.privmsg(nickname,msg.message)
```

Using `HEAP`, `textWindow()`, and `handle_input`

This plugin will collect any private messages the Ərk client receives (from any connection) and store them in **`HEAP`**. To view all these messages, a command named "/pm" is created; when entered into the text entry of any window, the collected private messages are displayed in a **`textWindow`**.

```
1  from erk import *
2
3  class PMLogger(Plugin):
4    def __init__(self):
5      self.name = "PM Logger"
6      self.version = "1.0"
7      self.author = "Nutjob Labs"
8      self.display = None
9
10   def load(self):
11     # Create a place to store private messages
12     HEAP["private messages"] = []
13
14   def unload(self):
15     # Delete stored private messages
16     del HEAP["private messages"]
17
18   def handle_message(self,msg):
19
20     # Store private messages
21     if msg == Message.PRIVATE:
22       HEAP["private messages"].append(msg.sender+": "+msg.message)
23
24   def handle_input(self,text):
25
26     # If the user enters the command...
27     if text.text == "/pm":
28
29       # Create the textWindow if it hasn't been created
30       if self.display:
31         self.display.clear()
32       else:
33         self.display = ERK.textWindow("Private Messages")
34
35       # Display stored private messages
36       self.display.write( "\n".join(HEAP["private messages"]) )
37
38       # Return True so /pm isn't sent as a chat message
39       return True
40
```