Ərk IRC Client, Version 0.880

Ərk Plugin Guide

A guide for using and writing plugins for the Ərk IRC Client

https://github.com/nutjob-laboratories/erk
https://github.com/nutjob-laboratories/erk-plugins

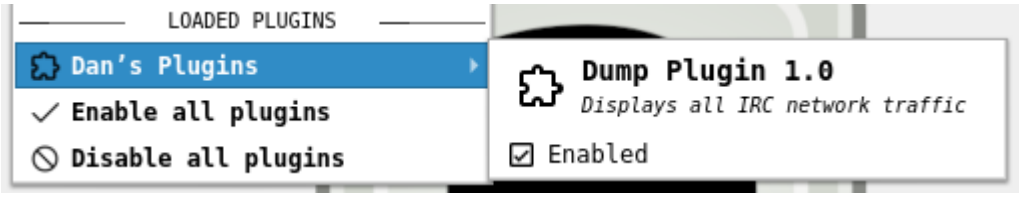
Summary

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

A basic plugin looks something like this:

```
from erk import *

_ERK_PLUGIN_ = "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)
```

If this was saved to a file and placed in **`.erk/plugins`**, the next time Ərk is started, it would load the plugin and appear in Ərk’s “Plugins” menu like so:



Once loaded, the plugin would print all network traffic, both input and output, from all servers that Ərk is connected to, to the console.

Using Ərk plugins

To install an Ərk plugin, simply place it in the **`.erk/plugins`** directory in your home directory. The next time Ərk starts up, the plugin will be loaded automatically. If there are major errors in the plugin (for example, improper Python code), Ərk will not start, and any errors will be printed to the console. If there are minor errors (for example, if plugins are missing needed attributes), Ərk will start normally, and any errors will be displayed to the user in a dialog window. Plugins with minor errors will *not* be loaded into the client, and, thus, will be ignored by Ərk.

Plugins can be loaded from other directories as well, by using the **`-P`**/**`--plugins`** command-line option. This option can be used multiple times; each call adds another directory to load plugins from.

Uninstalling Ərk plugins is as easy as deleting the plugin from **`.erk/plugins`**. No other steps are required.

Writing Ərk plugins

Plugin requirements and optional features

Ərk 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

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 **`_ERK_PLUGIN_`**
- A 48x48px PNG 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 **`erk`**. You can either explicitly import Plugin:

```
from erk import Plugin
```

or use a "splat" and import it implicitly:

```
from erk import *
```

The second two requirements, the **`NAME`** and **`VERSION`** attributes, are used by Ərk to display the plugin in the client (see page 3). These should be class attributes, rather than instance attributes[1]. **`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 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 Ərk will execute when a specific event occurs (see *Event Methods* on page 7).

If you add a string named **`_ERK_PLUGIN_`** to the plugin module, this string value will be used to display the plugin in Ərk'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 Ərk's "Plugins" menu.

1 https://www.geeksforgeeks.org/class-instance-attributes-python/

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 Ərk'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 image "**`bleep.png`**". If you want to display a plugin icon, you would name your 25x25px 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 **`.erk/plugins,`** named whatever you wish. Just like with normal Python modules, sub directories need a **`__init__.py`** file in order to be detected by Ərk.

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.

The easiest way to get started writing a plugin is to have Ərk create a "blank" plugin for you. Use the **`--generate`** command-line option, passing it the filename you want, and Ərk will generate a basic plugin skeleton for you. If no argument is passed to **`--generate`**, Ərk will print the plugin skeleton to STDOUT.

Event methods

Ərk 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, Ərk plugins are event-driven[2].

`action`

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

Description Triggered every time Ərk receives a CTCP action message. **`target`** contains the target of the message (if it's a public message, **`target`** will contain the name of the channel the message was sent to, and if it's a private message, **`target`** will contain the nickname the Ərk client is using), **`user`** contains the nickname, username, and hostname of the user that sent it (in the format **`nickname!username@hostname`**), and **`message`** contains the contents of the message.

`connect`

Arguments None

Description Triggered when Ərk connects to an IRC server. Please not that no IRC commands can be issued until Ərk finished registration with the IRC server; plugins will be notified when registration is complete by triggering the **`registered()`** event method (see page 12).

`connecting`

Arguments None

Description Triggered when Ərk begins the connection process to an IRC server. Please not that no IRC commands can be issued until Ərk finished registration with the IRC server; plugins will be notified when registration is complete by triggering the **`registered()`** event method (see page 12).

`ctcp`

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

Description Triggered whenever Ərk receives a CTCP message that is not otherwise recognized. **`user`** contains the nickname, username, and hostname of the user that sent it (in the format **`nickname!username@hostname`**), **`channel`** contains the channel (or nickname) the message was sent from, **`tag`** contains the CTCP message tag sent with the message, and **`message`** contains the contents of the sent message.

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

deop

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

Description Triggered whenever the Ərk client loses operator status in a channel. **user** is the nickname of the user that removed the operator status, and **channel** is the name of the channel the status was lost in.

devoice

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

Description Triggered whenever the Ərk client loses voiced status in a channel. **user** is the nickname of the user that removed the voiced status, and **channel** is the name of the channel the status was lost in.

exit

Arguments None

Description Triggered when the Ərk application is closing. This event is the last thing that Ərk does before exiting. *The **irc** object is **not available for use***, as all IRC connections have either been disconnected or are in the process of disconnecting.

input

Arguments **name** (string), **text** (string)

Description Triggered every time a user types something into Ərk's text input widget. **name** is the name of the window where the text was entered (a channel name for a public chat window, or a nickname for a private chat window), and **text** is the text that was entered. If the text was entered into a server console window, name will be set to **None**. If the user has implemented any macros, they will be interpolated into the text *before* being handed to this method. To stop Ərk from processing the user input, return **True** from this method; to allow Ərk to continue to work with the input text, return **False** or nothing. As it would be very easy to write a malicious plugin that would disable *any* input, Ərk will try to detect malicious **input()** methods and prevent the plugin that contains them from being loaded; due to this, the **input** event can be disabled in the "Preferences" dialog (see page 19).

invite

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

Description Triggered every time Ərk receives a channel invitation. **user** contains the nickname, username, and hostname of the user that sent the invitation (in the format **nickname!username@hostname**), and channel contains the name of the channel Ərk is being invited to.

join

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

Description Triggered every time another user joins a channel that the Ərk client is "present" in. **channel** contains the channel the user joined, and **user** contains the nickname, username, and hostname of the user that joined (in the format **nickname!username@hostname**).

joined

Arguments **channel** (string)

Description Triggered whenever Ərk joins a channel. **channel** contains the channel that the client joined.

kick

Arguments **channel** (string), **kickee** (string), **kicker** (string), **message** (string)

Description Triggered whenever a user is kicked from a channel Ərk is "present" in. **channel** contains the channel the user was kicked from, **kickee** contains the nickname of the user that was kicked out of the channel, **kicker** contains the nick of the user that kicked the user out, and **message** contains the message attached to the kick notification.

kicked

Arguments **channel** (string), **kicker** (string), **message** (string)

Description Triggered whenever Ərk is kicked from a channel. **channel** contains the channel the client was kicked from, **kicker** contains the nick of the user that kicked the client out, and **message** contains the message attached to the kick notification.

line_in

Arguments **data** (string)

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

line_out

Arguments **data** (string)

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

mode

Arguments **channel** (string), **user** (string), **mset** (bool), **modes** (string), **args** (list)

Description Triggered every time Ərk receives a mode message. **channel** contains the name of the channel the message was sent to, **user** contains the nickname, username, and hostname of the user that sent it (in the format **nickname!username@hostname**), **mset** is True if a mode is being set and False if a mode is being removed/unset, **modes** contains the mode(s) being set, and **args** contains a list of the arguments to the mode being set/unset.

motd

Arguments **message** (list)

Description Triggered when Ərk receives the message of the day (MOTD) from an IRC server. **message** contains the MOTD, with each entry consisting of one line.

nick

Arguments **oldnick** (string), **newnick** (string)

Description Triggered every time another user changes their nickname in Ərk's "presence". **oldnick** contains the old nickname, and **newnick** contains the new nickname.

notice

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

Description Triggered every time Ərk receives a notice message. **target** contains the name of the target the message was sent to (either a channel or your nickname), **user** contains the nickname, username, and hostname of the user that sent it (in the format **nickname!username@hostname**), and **message** contains the contents of the message.

op

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

Description Triggered whenever the Ərk client gains operator status in a channel. **user** is the nickname of the user that granted the operator status, and **channel** is the name of the channel the status was granted in.

oper

Arguments None

Description Triggered when the user successfully logs into an IRCop[3] account with Ərk.

3 https://tools.ietf.org/html/rfc1459#section-1.2.1

part

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

Description Triggered every time another user leaves a channel that the Ərk client is "present" in. **channel** contains the channel the user left, and **user** contains the nickname, username, and hostname of the user that left (in the format **nickname!username@hostname**).

parted

Arguments **channel** (string)

Description Triggered whenever Ərk leaves a channel. **channel** contains the channel that the client left.

private

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

Description Triggered every time Ərk receives a private message. **user** contains the nickname, username, and hostname of the user that sent it (in the format **nickname!username@hostname**), and **message** contains the contents of the message.

public

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

Description Triggered every time Ərk receives a public message. **channel** contains the name of the channel the message was sent to, **user** contains the nickname, username, and hostname of the user that sent it (in the format **nickname!username@hostname**), and **message** contains the contents of the message.

quit

Arguments **nickname** (string), **message** (string)

Description Triggered whenever Ərk is receives a quit notification. **nickname** is the nickname of the user that quit IRC, and **message** is the (optional) message attached to the quit notification.

registered

Arguments None

Description Triggered when Ərk completes connecting to an IRC server. This will trigger shortly after the **`connect()`** event (see page 7). Once Ərk is registered with the IRC server, commands can be sent to it as normal.

tick

Arguments **`uptime`** (integer)

Description Triggered once every second that Ərk is connected to a server. **`uptime`** is the length of the connection to the server in seconds. Timers are specific to each connection; each server connection's tick event is independent of all the other server connections.

topic

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

Description Triggered when the channel topic is set for a channel that Ərk is "present" in. **`channel`** is the name of the channel the topic was set in, **`user`** contains the nickname, username, and hostname of the user that set the topic (in the format **`nickname!username@hostname`**), and **`topic`** is the new topic. If **`user`** is set to an empty string, the server set the channel topic.

voice

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

Description Triggered whenever the Ərk client gained voiced status in a channel. **`user`** is the nickname of the user that granted the voiced status, and **`channel`** is the name of the channel the status was gained in.

`Plugin` class and instance features

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

The `irc` Object

The **`irc`** object is part of the **`Plugin`** class, and is the way plugins interact with IRC servers. It is integrated as an instance attribute of the class. This object is the instance of the Twisted IRC client[4] that Ərk 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`** object 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 private event method that forwards all private messages to another user with the nickname "OtherNick", you could write:

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

If the **`irc`** object is unavailable (due to a disconnection, error, or other reason), the object's value is set to **`None`**. For help on how to use the **`irc`** object, take a look at the documentation for the Twisted IRC Client[5].

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

The `stack` Object

The **`stack`** object is a List that's part of the **`Plugin`** class, and is shared by *all* plugins. It is integrated as an instance attribute of the class. The **`stack`** object provides a way to for separate plugins to share data. It's accessed just like the **`irc`** object (i.e. with **`self.stack`**). Other than it being a part of every **`Plugin`**, it's just a normal Python List.

---

4 **`twisted.words.protocols.irc.IRCClient`**

5 https://twistedmatrix.com/documents/current/api/twisted.words.protocols.irc.IRCClient.html

Built-in methods

The Plugin class also comes with several methods built into it. These methods are used to interact with the Ərk graphical user interface (GUI).

`ask`

Arguments	**`text`** (string)
Returns	String or **`None`**
Description	Displays a message box with **`text`**, a text entry widget, and “OK” and “Cancel” buttons. Users can enter text into the entry widget, and if they press “OK” or press return after entering text, whatever is in the text box will be returned as a string. If “Cancel” is clicked, or the message box is closed, **`ask()`** will return **`None`**.

`autocomplete`

Arguments	**`search`** (string), **`replacement`** (string)
Returns	Nothing
Description	Adds an entry to the autocompleter. **`search`** is the thing the autocompleter is looking for; for example, if you have implemented a new command, **`search`** would be the new command. **`replacement`** is what you want the autocompleter to replace the entry with if a match is made; if your new command, for example, takes an argument, **`replacement`** would be the new command followed by a space, so all the user has to do is type the argument.

`console`

Arguments	**`text`** (string)
Returns	Nothing
Description	Prints **`text`** to the window associated with the current server connection, called the “console”.

`current`

Arguments	None
Returns	**`String`** or **`None`**
Description	Retrieves the name of the current window in use (for use with **`write()`**, for example). If the current window is a server console, or there is no current window, **`current()`** returns **`None`**.

`directory`

Arguments	None
Returns	**`String`** or **`None`**
Description	Returns the name of the directory the plugin is located in. If the location is unknown, **`directory()`** will return **`None`**.

`exec`

Arguments	**`text`** (string)
Returns	**`True`** if the command was able to be executed, **`False`** if not
Description	Executes the contents of text as if it were entered into the text input widget in the Ərk client. This allows plugins to execute miscellaneous scripting commands (see *Erk Scripting and Commands* for more information). If **`text`** does not contain a command, it will be sent to the currently open window as chat.

`help`

Arguments	**`usage`** (string), **`description`** (string)
Returns	Nothing
Description	Adds an entry to the **`/help`** command (see *Erk Scripting and Commands* for more information). **`usage`** is the usage text (usually the command followed by example arguments), and **`description`** is the description of what the command does.

`port`

Arguments	None
Returns	**`Integer`** or **`None`**
Description	Returns the port number used to connect to the IRC server that triggered the event is called from; if unknown, then **`port()`** returns **`None`**.

`print`

Arguments	**`text`** (string), ...
Returns	Nothing
Description	Prints **`text`** to the currently open window; if the currently open window can't be found, then the text will print to the console. To print multiple items in the same command, separate the strings to print with commas (much like Python's **`print`** function).

script

Arguments	**filename** (string), **arguments** (list)
Returns	**True** if the call succeeded, and **False** if the call failed
Description	Executes the contents of **filename** as an Ərk script. This functions exactly like the Ərk command **/script** (see *Erk Scripting and Commands*). Much like the /script command, the script is executed in a separate thread, so **script()** will return *immediately* after being called. **script()** will return **True** if the script was able to be executed (regardless if there are errors or not in the script), and **False** if the script was *not* able to be executed. If scripting is disabled, **script() will *always fail*.**

server

Arguments	None
Returns	**String** or **None**
Description	Returns the hostname or IP address used to connect to the IRC server that triggered the event is called from; if unknown, then **server()** returns **None**.

switch

Arguments	**window** (string)
Returns	**True** if successful, **False** if failed
Description	Switches Ərk's view to the chat window named **window**. This can be a channel name, or a nickname of a user in private chat. Ərk *must* have that window open (so, the client must be present in the channel, or have a window opened for the private chat). If the window was "found", and switched do, **switch()** will return **True**; if not, for any reason, **switch()** will return **False**.

uptime

Arguments	None
Returns	**Integer** or **None**
Description	Returns the uptime of the current connection, in seconds. If the uptime is unknown, **uptime()** will return **None**.

windows

Arguments	None
Returns	**`List`** or **`None`**
Description	Returns a list of currently open windows. Only windows for the current IRC connection are shown; for example, if this command is called in a **`public()`** event method (see page 11), only windows associated with the connection that triggered the **`public()`** event will be returned.

write

Arguments	**`target`** (string), **`text`** (string)
Returns	**`Boolean`**
Description	Prints **`text`** to the window with **`target`** as its name. Windows are named after the chat they display; so, for example, the window displaying channel chat for #erk would be named “#erk”. The window displaying private chat with a user named “Bob” would be named “Bob”. If the window was found and the text was written to it, **`write`** will return **`True`**; if the window was *not* found, **`write`** will return **`False`**.

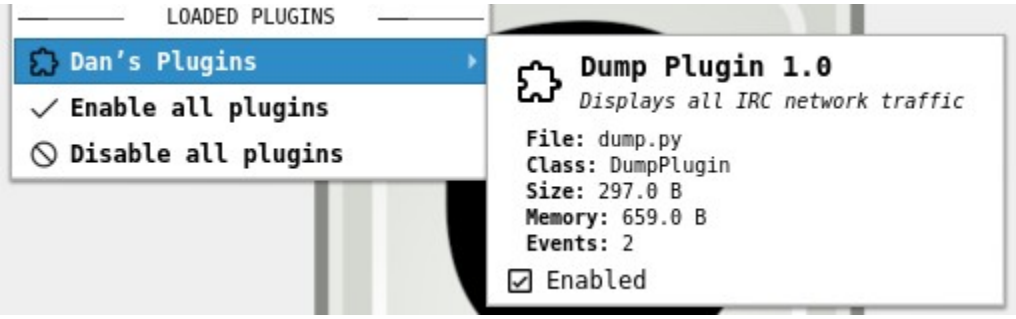
Miscellany

“Catching” ignored messages

Users that have been ignored with the **`/ignore`** command or through the GUI (see *Erk Scripting and Commands* for more information) will not trigger the **`private`** (page 11), **`public`** (page 11) , **`notice`** (page 10), and **`action`** (page 7) events. Any messages sent from ignored users to the client are completely ignored. This behavior can be changed by opening the “Preferences” dialog (either via the “Settings” menu or by using the **`--settings`** command-line flag), navigating to the “Plugins” page, and enabling “Plugins catch ignored messages”.

Plugin details

While developing plugins, it may be helpful to see more information on plugins when they are loaded into Ərk. To see more details in the “Plugins” menu, open the “Preferences” dialog (either via the “Settings” menu or by using the **`--settings`** command-line flag), navigate to the “Plugins” page, click the “Menu” tab, and enable “Show plugins details in menu”. The next time you open the “Plugins” menu (and every time after that, until you disable the option), plugin entries will look something like:



- **File.** The filename of the plugin module.
- **Class.** The name of the plugin class.
- **Size.** The size of the plugin module file.
- **Memory.** Roughly how much system memory the plugin is using.
- **Events.** The number of event methods (see page 7) the plugin has.

If the plugin was loaded from an external source (that is, not loaded from the **`.erk/plugins`** directory, by using the **`--plugins`** command-line option), this will be noted in the details.

Loading plugins from other directories

Ərk can load plugins from multiple directories, not just from **`.erk/plugins`**. To load plugins from another directory, or from multiple directories, use the **`--plugins`** command-line option, passing it

a directory path as an argument. To load multiple directories, use the **`--plugins`** option multiple times, passing a single directory as an argument to each call. All plugins in every directory passed in this manner to Ərk will be loaded and used. This can be helpful while developing a plugin; simply load the directory you're writing the plugin in while you're writing the plugin, and when you're done, copy the finished plugin into **`.erk/plugins`**.

Another way to load a directory of plugins is to click on the "Settings" menu, and clicking on "Load plugins". Once a directory is selected, Ərk will load all the plugins it finds in that directory.

If you want Ərk to load plugins from a certain directory *every* time Ərk loads, you can add a directory to load plugins from on the "Plugins" page of the "Preferences" dialog, on the "Sources" tab. Click the "Add directory" to select and add a directory to load plugins from; select a listing and click "Remove directory" to remove a directory from the list.

Malicious `input` events

It is possible to craft a malicious **`input`** event (see page 8) for a plugin. Since the event can "short circuit" user input, simply having an **`input`** event that always returns **`True`** will prevent a user from inputting *anything* into Ərk. Ərk tries to detect malicious input, but detection is nowhere near perfect, and malicious **`input`** events can be loaded. If a user loads a plugin and finds that they can no longer input anything into Ərk's text input widget, they can disable all plugin **`input`** events from the "Preferences" dialog. Click on the "Settings" menu, and click "Preferences" to load the "Preferences" dialog; navigate to the "Plugins" tab, and un-check "Enable plugin input events" to disable the **`input`** events for all plugins.