Version 0.410

Plugin Development Guide

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

The Erk Plugin System

The Erk plugin system allows user to extend and add new functionality to the Erk IRC client. Plugins are written in Python3; Erk will load any plugin files placed in the "plugins" directory automatically. There are two words that are used frequently in this document: **package**, and **plugin**:

- **Package:** Any file that contains one or more *plugins*.
- **Plugin**: A Python3 class that inherits from the **Plugin** class in Erk.

Plugins are event driven; each event has a corresponding plugin method that is executed when the event occurs. Plugins only need to contain the methods necessary for the events they need to function.

A package can contain multiple plugins. There is no limit to how many plugins Erk can load at one time other than memory, hard drive, and CPU limitations.

The Plugin Class

Erk contains a class (named, helpfully, **Plugin**) that all plugins must inherit from. To import the class into your plugin's source code, import it from the main Erk class, **erk**:

```
from erk import Plugin
```

Erk plugins have shared access to a dictionary named **Shared**. To take advantage of this, import **Shared** from **erk**:

```
from erk import Plugin,Shared
```

Inherited Class Requirements

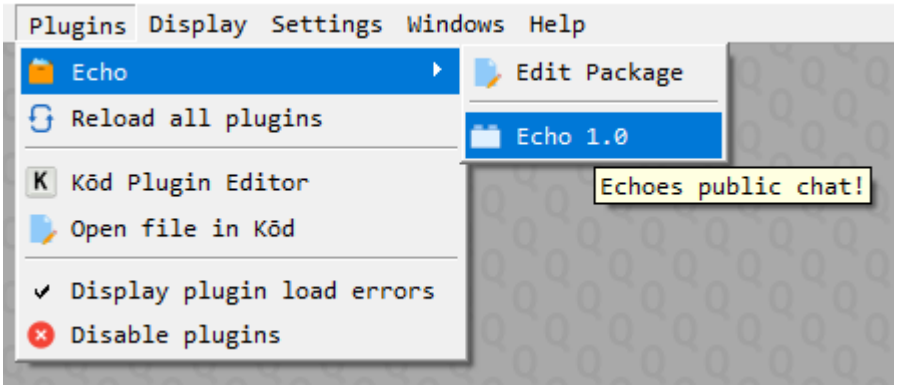
A valid plugin requires that the inherited class contains three attributes:

- **name** – Sets the name of the plugin
- **version** – Sets the plugin's version
- **description** – Sets a short description of the plugin.

So, for example, if you wanted to write a plugin named "Echo" that echoes everything said in public chat, setting the required attributes could look like this:

```
class EchoPlugin(Plugin):
  def __init__(self):
    self.name = "Echo"
    self.version = "1.0"
    self.description = "Echoes public chat!"
```

These values are used in Erk to display the plugin's information in the "Plugins" menu:



Events

The Erk plugin system is event driven: every time a given event occurs, any plugin that has a class method assigned to that event is executed. These events can be IRC-based (the client receives a public or private message, for example) or client-based (the user has entered text and pressed the return key). The class method is passed a number of arguments containing information about the event being handled.

For example, let's write a small plugin that echoes any channel chat back to the channel. To do that, we'll use the **`message_public`** event:

```
class EchoPlugin(Plugin):
  def __init__(self):
    self.name = "Echo"
    self.version = "1.0"
    self.description = "Echoes public chat!"

  def message_public(self,server,user,channel,message):
    # A channel message was received! We'll use the
    # built-in "msg" function to repeat the message
    # to the channel it came from.
    self.msg(channel,message)
```

When this plugin is loaded, Erk will automatically parrot anything someone says in a channel that the user is in:

```
23:30:11      wraithnix  hello there, quirc!
23:30:11          quirc  hello there, quirc!
23:30:16      wraithnix  huh?
23:30:16          quirc  huh?
23:30:21      wraithnix  stop repeating me!
23:30:21          quirc  stop repeating me!
```

The echo plugin in action.

There are 20 methods associated with events in the **Plugin** class, listed below. Many of the event methods are passed a string argument called a **serverID**; this is a unique 16 character string that is randomly generated, and serves as an identifier for each client-server connection.

Table 1: List of plugin events

Method	Arguments	Triggered By	Description
load	*None*	Plugin load.	Executes when Erk first loads the plugin.
unload	*None*	Erk exit.	Executes when the plugin is unloaded (upon Erk's exit).
server_connected	**serverID** (string)	Connection to an IRC server.	Executes when Erk connects to an IRC server.
server_disconnected	**serverID** (string) **reason** (string)	Disconnection from an IRC server.	Executes when Erk disconnects from an IRC server.
server_registered	**serverID** (string)	Registering with an IRC server.	Executes when Erk is registered with an IRC server.
server_motd	**serverID** (string) **motd** (list)	Receiving an IRC server's message of the day.	Executes when Erk receives an IRC server's message of the day (MOTD).
message_public	**serverID** (string) **channel** (string) **user** (string) **message** (string)	Receiving a public message.	Executes when Erk receives or sends a public message.
message_private	**serverID** (string) **user** (string) **message** (string)	Receiving a private message.	Executes when Erk receives or sends a private message.
message_notice	**serverID** (string) **channel** (string) **user** (string) **message** (string)	Receiving a notice.	Executes when Erk receives a notice.
message_action	**serverID** (string) **channel** (string) **user** (string) **message** (string)	Receiving a CTCP action message.	Executes when Erk receives a CTCP action message.
channel_join	**serverID** (string) **channel** (string) **user** (string)	User (including the Erk client) joining a channel.	Executes when a user joins a channel the Erk client is in.
channel_part	**serverID** (string) **channel** (string) **user** (string)	User (including the Erk client) leaving a channel.	Executes when a user leaves a channel the Erk client is in.

Method	Arguments	Triggered By	Description
channel_topic	**serverID** (string) **channel** (string) **user** (string) **topic** (string)	User changing an IRC channel's topic.	Executes when the channel topic of a channel Erk is in changes.
channel_invite	**serverID** (string) **channel** (string) **user** (string)	Receiving a channel invite from a user.	Executes when Erk receives a channel invite.
server_mode	**serverID** (string) **set_or_unset** (bool) **user** (string) **target** (string) **modes** (string) **arguments** (list)	Receiving a mode change notification.	Executes when Erk receives a mode change notification from an IRC server.
server_quit	**serverID** (string) **user** (string) **message** (string)	User quits (disconnects) from an IRC server.	Executes when a user in any of the channels Erk is in quits IRC.
server_raw	**serverID** (string) **data** (string)	Whenever the Erk client receives *anything* from an IRC server.	Executes whenever Erk receives a message from any server it is connected to.
tick	**serverID** (string) **uptime** (integer)	Triggered (roughly) once a second while connected to an IRC server.	Executes once a second while Erk is connected.
input	**serverID** (string) **source** (string) **text** (string)	User has entered text into any one of Erk's chat display windows.	Executes whenever text is entered into any of Erk's display windows.
menu	*None*	User has clicked on the plugin's entry in the "Plugins" menu.	Executes whenever the plugin's "Plugin" menu entry is clicked.

Class Methods

The Plugin class also possesses 27 built-in methods for interacting with the Erk client and IRC servers.

By default, all IRC methods send data to the server connection that triggered the event. Plugins, however, can execute commands on any server that Erk is connected to. To execute a command on server other than the one that triggered the plugin event, pass the server connection's **serverID** as a final argument.

Method arguments in italics are optional.

Table 2: List of class methods

Method	Arguments	Returns	Type	Description
msg	**target** (string) **message** (string) *serverID* (string)	Nothing	IRC	Sends **message** to **target**, which can be a channel or a nickname.
notice	**target** (string) **message** (string) *serverID* (string)	Nothing	IRC	Sends a notice containing **message** to **target**, which can be a channel or nickname.
action	**target** (string) **message** (string) *serverID* (string)	Nothing	IRC	Sends a CTCP action containing **message** to **target**, which can be a channel or nickname.
join	**channel** (string) *key* (string) *serverID* (string)	Nothing	IRC	Attempts to join an IRC **channel**.
part	**channel** (string) *message* (string) *serverID* (string)	Nothing	IRC	Leaves an IRC **channel**, sending a parting **message** to the channel (optional).
send	**line** (string) *serverID* (string)	Nothing	IRC	Sends a message directly to the IRC server; this allows plugins to send messages and commands that aren't built-in or supported by Erk.
kick	**channel** (string) **user** (string) *reason* (string) *serverID* (string)	Nothing	IRC	Attempts to kick **user** from **channel**.
invite	**user** (string) **channel** (string) *serverID* (string)	Nothing	IRC	Sends an invitation to **channel** to **user**.

topic	**channel** (string) **topic** (string) *serverID* (string)	Nothing	IRC	Attempts to change **channel**'s topic to **topic**.
mode	**target** (string) **operation** (boolean) **modes** (string) *limit* (integer) *user* (string) *mask* (string) *serverID* (string)	Nothing	IRC	Attempts to set or unset **modes** on **target** (which can be a channel or user). To set a mode, set **operation** to True; to unset a mode, set **operation** to False. ***limit*** is used in conjunction with the "l" mode, which limits the number of users in a channel. ***user*** is used with the "o", "v", or any mode that takes a user as an argument. ***mask*** is used with the "b" option (for banning users from a channel).
nick	**nick** (string) *serverID* (string)	Nothing	IRC	Attempts to set a new nickname.
quit	*message* (string) *serverID* (string)	Nothing	IRC	Disconnects from an IRC server, sending an optional **message**.
away	*message* (string)	Nothing	IRC	Sets the client to "away" on the current IRC server, using an optional **message**.
back	None	Nothing	IRC	Sets the client to "back" on the current IRC server
getNickname	*serverID* (string)	string	IRC	Returns the client's current nickname.
getHostname	*serverID* (string)	string	IRC	Returns the current server's hostname.
getServer	*serverID* (string)	string	IRC	Returns the IP the client used to connect to the current IRC server.
getPort	*serverID* (string)	integer	IRC	Returns the port the client used to connect to the current IRC server.
getConnections	None	list	IRC	Returns a list of the serverIDs of all the servers Erk is currently connected to.

getUsers	**channel** (string) *serverID* (string)	list	IRC	Returns a list of users in a channel, in the format of *nick*! *username@host*, if the username and host is known; if not, only the user's nickname will be available. Channel status symbols will be prepended to the nickname ('@' for channel operators, '+' for voiced users, etc.). If the client is not present in the given channel, or the user list cannot be found, an empty list is returned.
client	**serverID** (string)	Twisted IRCClient object for the desired server	IRC	Returns the Twisted IRCClient object for the connection associated with **serverID**.
color	**text** (string) **foreground** (integer) *background* (integer)	string	IRC	Colors a string with mIRC color codes. f**oreground** (and *background*, if used) must be a number between 0 and 15. If a number less than zero or greater than 15 is passed as an argument, the unchanged text is returned. **0** = white **1** = black **2** = blue **3** = green **4** = red **5** = brown **6** = purple **7** = orange **8** = yellow **9** = light green **10** = teal **11** = cyan **12** = light blue **13** = pink **14** = grey **15** = light grey

print	**text** (string) *window* (string)	Nothing	GUI	Prints a line of text in Erk; where that text is printed depends on the **window** argument: • None (prints to the active window) • all (prints to all windows) • log (prints to the connection log) • a username or channel (prints to the window displaying the channel or user chat)
suppress	**text** (string)	Nothing	GUI	Adds text to the suppression list. If any incoming private message matches an entry in the suppression list, it will not be displayed in Erk. Wildcards that can be used in suppression list entries: • * (matches anything) • ? (matches any single character)
unsuppress	**text** (string)	Nothing	GUI	Removes text from the suppression list.
getWindows	*serverID* (string)	dictionary	GUI	Returns a list of Erk window. The dictionary uses the serverID each window is associated with as the key; each dictionary entry is a list of all window names associated with that serverID. If **serverID** is passed as an argument, only windows associated with that serverID are returned.
serverIDtoHost	**serverID** (string)	string	GUI	Converts a **serverID** into a string containing the IP/hostname and port associated with that **serverID**, in this format: *host*:*port*

Special Attributes

There are three special attributes that can be set in a Erk plugin. These can be set in any plugin method, and they're used to turn off several plugin behaviors.

Attribute	Type	Default	Description
`silent`	boolean	`False`	Prevents any messages sent by the plugin from being displayed in Erk if set to **`True`**.
`nowindows`	boolean	`False`	Prevents window creation by the plugin if set to **`True`**.
`noirc`	boolean	`False`	Prevents the plugin from using any IRC commands if set to **`True`**.

Plugin Examples

The source code for all these plugins can be found at the Nutjob Laboratories plugin repository: https://github.com/nutjob-laboratories/erk-plugins

Public Chat Counter

As an example, let's write a plugin that tracks how many times users "speak" (send a public message) in a channel Erk is in. First, we have to import the **Plugin** class and the **Shared** dictionary:

```
from erk import Plugin,Shared
```

We're going to name our plugin class "CounterClass" and our plugin "Counter". It'll be version 1.0 of our plugin:

```
class CounterClass(Plugin):
  def __init__(self):
    self.name = "Counter"
    self.version = "1.0"
    self.description = "Counts every public message Erk sees"
```

Since we want to track public chat, we're going to use the **`message_public`** event method. We're also going to use the **`load`** event method to set our counter to zero, and we're going to store our counter in the **Shared** dictionary:

```
  def load(self):
    Shared["counter"] = 0

  def message_public(self,serverID,channel,user,message):
    Shared["counter"] = Shared["counter"] + 1
```

The reason why we're using the **Shared** dictionary rather than a class variable is simple: each plugin is ran once for *each connection Erk is using*. If we used a class variable, the plugin could only track the number of public messages it "sees" on each server. By using the **Shared** dictionary, we can track how many public messages Erk "sees" on every server it is connected to.

Okay, now we're counting all the public messages. How can we see what the current count is? We're going to use the **`message_private`** event to create a special command: "!count". Anyone who sends a private message consisting of "!count" to the Erk client will be sent back the current count via a private message:

```
  def message_private(self,serverID,user,message):
    # Since it's likely that the "user" argument contains
    # the user's nickname, username, and host, we'll parse
    # out the user's nick if that's the case
    tokens = user.split("!")
```

```
    if len(tokens)==2:
        user = tokens[0]

    if message == "!count":
      self.msg(user,"Total public messages: " + str(Shared["counter"]))
```

If we used this plugin now, it would count the number of public messages and send the count to anyone who asked for it. However, every time the count is sent to a user, it shows up in Erk's chat. It'll even open new windows for users we're not chatting with! To make our plugin silent, we're going to use two special plugin attributes that can be used to make Erk work how we want it to work. **`silent`** is an attribute that will prevent the display of any outgoing messages from the plugin. **`nowindows`** is an attribute that will prevent our plugin from opening any new windows. However, you'll still see the "!count" message that is sent to you by querying users. We'll use the **`suppress()`** method to ignore users sending us the count command. We'll modify our **`load`** event method to set the attributes and suppress display of the "!count" command; we'll set Erk to ignore any private messages that start with "!count!:

```
  def load(self):
    Shared["counter"] = 0
    self.silent = True
    self.nowindows = True
    self.suppress("!count*")
```

Let's put it all together!

```
from erk import Plugin,Shared

class CounterClass(Plugin):
  def __init__(self):
    self.name = "Counter"
    self.version = "1.0"
    self.description = "Counts every public message Erk sees"

  def load(self):
    Shared["counter"] = 0
    self.silent = True
    self.nowindows = True
    self.suppress("!count*")

  def message_public(self,serverID,channel,user,message):
    Shared["counter"] = Shared["counter"] + 1

  def message_private(self,serverID,user,message):
    tokens = user.split("!")
    if len(tokens)==2:
        user = tokens[0]

    if message == "!count":
      self.msg(user,"Total public messages: " + str(Shared["counter"]))
```

Persistent Chat Counter

In this example, we'll modify the **Public Chat Counter** to keep track of how many public chats it "sees" even when Erk is exited and restarted. To achieve this, we'll modify the **`load`** event method, and use another event method: **`unload`**.

The **`load`** event method is executed as soon as the plugin is loaded; that is, when Erk starts up. We'll need to store our chat count in a file, so we should look for that file and load it into memory if it exists. Our chat count file will be named "chatcount.txt". First, we'll load in the **Plugin** class and **Shared** dictionary, and import the **os** module to help figure out if the chat count file exists:

```
from erk import Plugin,Shared
import os
```

Now, we'll create a new **Plugin** class, *PersistentCounter*, and use some of the code already written in the **Public Chat Counter** example. We'll use the code we wrote for the **`message_public`** and **`message_private`** event methods:

```
class PersistentCounter(Plugin):
  def __init__(self):
    self.name = "Persistent Counter"
    self.version = "1.0"
    self.description = "Persistently counts every public message Erk sees"
    self.silent = True
    self.nowindows = True
    self.suppress("!count*")

  def message_public(self,serverID,channel,user,message):
    Shared["counter"] = Shared["counter"] + 1

  def message_private(self,serverID,user,message):
    tokens = user.split("!")
    if len(tokens)==2:
        user = tokens[0]

    if message == "!count":
      self.msg(user,"Total public messages: " + str(Shared["counter"]))
```

To make our plugin persistent, we'll write a new **`load`** event method. This will look for the "chatcount.txt" file, and load its contents into the **Shared** dictionary if it exists, or set the counter stored in **Shared** to zero if it doesn't:

```
  def load(self):
    if os.path.isfile("chatcount.txt"):
      f = open("chatcount.txt", "r")
      Shared["counter"] = int(f.read())
    else:
      Shared["counter"] = 0
```

However, the plugin isn't persistent yet! We need to store our chat count. We could save the count every time the plugin "sees" a public message, but this could take up a lot of time and processing power, especially if there's a lot of chatting going on. We're going to take a different path: we'll use the **unload** event method, and save the in-memory chat counter to disk when Erk exits:

```
  def unload(self):
    f = open("chatcount.txt","w+")
    f.write(str(Shared["counter"]))
```

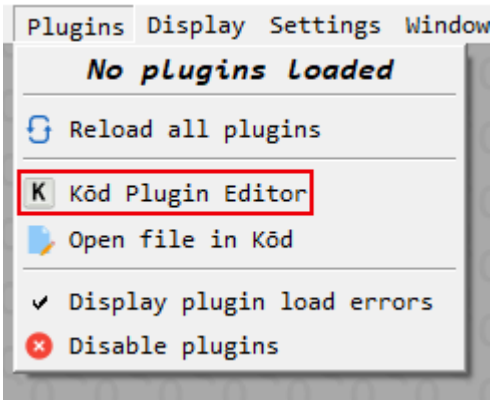
Our plugin now counts all public chat, reports it to users who send a private message consisting of "!count" to the client, and saves the chat count across reboots:

```
from erk import Plugin,Shared
import os

class PersistentCounter(Plugin):
  def __init__(self):
    self.name = "Persistent Counter"
    self.version = "1.0"
    self.description = "Persistently counts every public message Erk sees"
    self.silent = True
    self.nowindows = True
    self.suppress("!count*")

  def message_public(self,serverID,channel,user,message):
    Shared["counter"] = Shared["counter"] + 1

  def message_private(self,serverID,user,message):
    tokens = user.split("!")
    if len(tokens)==2:
        user = tokens[0]

    if message == "!count":
      self.msg(user,"Total public messages: " + str(Shared["counter"]))

  def load(self):
    if os.path.isfile("chatcount.txt"):
      f = open("chatcount.txt", "r")
      Shared["counter"] = int(f.read())
    else:
      Shared["counter"] = 0

  def unload(self):
    f = open("chatcount.txt","w+")
    f.write(str(Shared["counter"]))
```

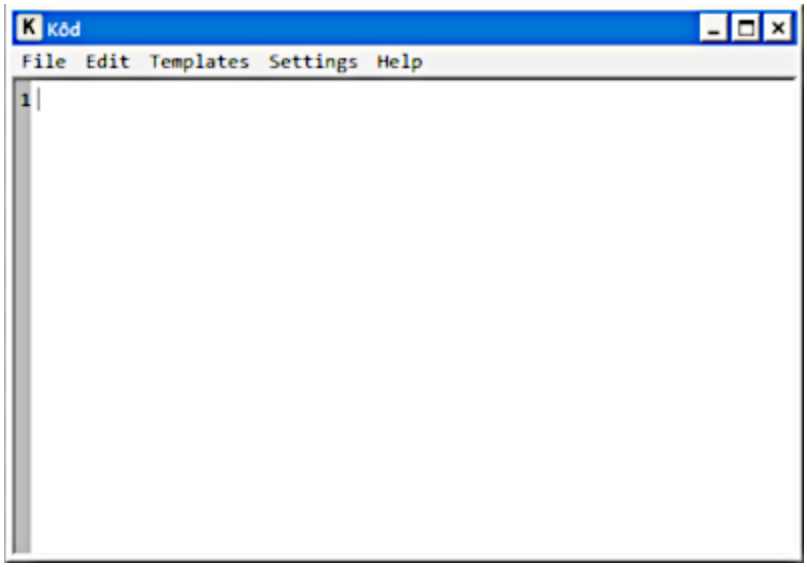
The Kōd Plugin Editor

Erk has a plugin editor built into it: **Kōd**. It's a basic text editor that does syntax highlighting for Python, with code generators and template for the most commonly written plugin code.

Kōd can be started using the command-line flag `--editor`, or by clicking on its entry in the "Plugins" menu:



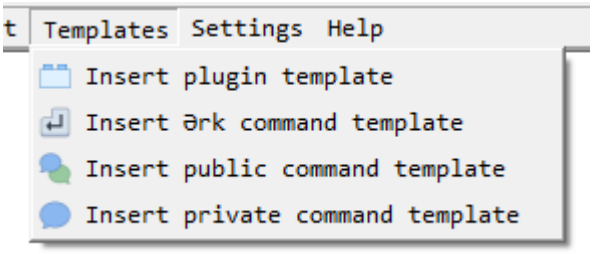
When started, a blank document is displayed, ready for editing:



Templates

Kōd can generate plugin code from templates, removing the need to write a lot of "boilerplate"[1] code. The generated code contains everything you need for a plugin; it will even load without any editing (even though the plugin would do nothing. To use the templates, open the "Templates" menu.

1 https://en.wikipedia.org/wiki/Boilerplate_text

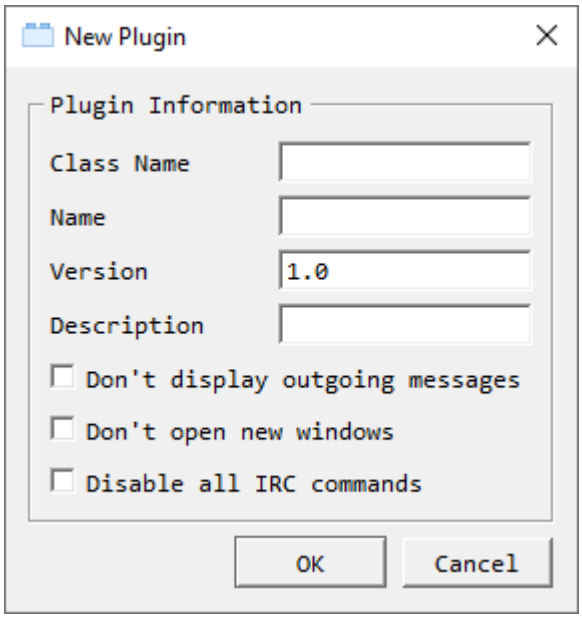


Kōd can generate four kinds of plugin code from these templates:

- **Plugin template.** Generates all the code necessary for an **Erk** plugin.
- **Erk command template.** Generates the code necessary to create a new command for the text entry widget.
- **Public command template.** Generates the code necessary to create a new command that can be triggered by public chat messages.
- **Private command template.** Generates the code necessary to create a new command that can be triggered by private chat messages.

Plugin

This generates a plugin class with all **Erk** event methods defined. Clicking on this template's menu item brings up a dialog where you can enter all the custom values for the new plugin:



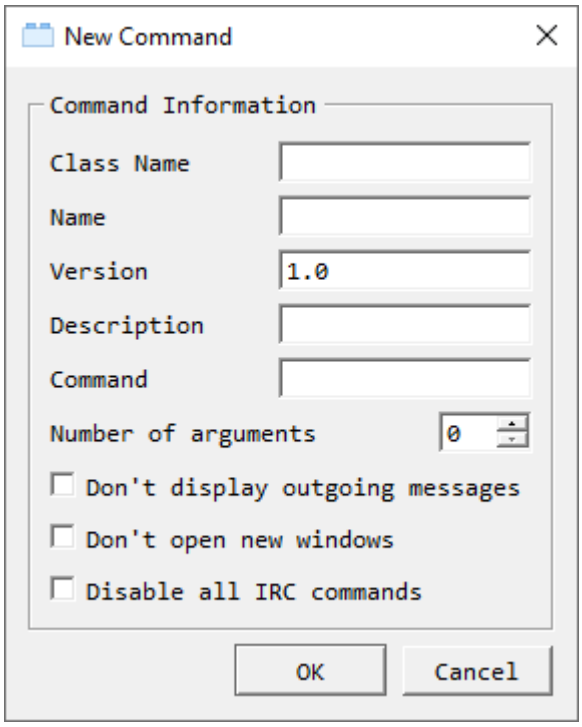
- **Class Name.** The class name the plugin will use; it cannot contain any spaces or punctuation. Any valid[2] **Python** class name will do.
- **Name.** The name of the plugin. This value is what will appear in the "Plugins" menu.
- **Version.** The version of the plugin; default is "1.0"

2 http://docs.python.org/reference/lexical_analysis.html#identifiers

- **Description.** A short description of the plugin. This value sets the tooltip that will display when hovering the mouse over the plugin's menu entry.
- **Don't display outgoing messages.** This sets the plugin's **`silent`** attribute to "True".
- **Don't open new windows.** This sets the plugin's **`nowindows`** attribute to "True".
- **Disable all IRC commands.** This sets the plugin's **`noirc`** attribute to "True".

Erk Command

This generates the code necessary to create a new command for **Erk**. Commands work exactly the same as the **`/me`**, **`/join`**, or **`/quit`** commands in the text entry widget. Clicking on this template's menu item brings up a dialog where you can enter all the custom values for the new plugin:



- **Class Name.** The class name the plugin will use; it cannot contain any spaces or punctuation. Any valid **Python** class name will do.
- **Name.** The name of the plugin. This value is what will appear in the "Plugins" menu.
- **Version.** The version of the plugin; default is "1.0"
- **Description.** A short description of the plugin. This value sets the tooltip that will display when hovering the mouse over the plugin's menu entry.
- **Command.** The text that will trigger the command; for example, the "trigger" for the **Erk** command to join a channel is **`/join`**.
- **Number of arguments.** How many arguments the command accepts.
- **Don't display outgoing messages.** This sets the plugin's **`silent`** attribute to "True".
- **Don't open new windows.** This sets the plugin's **`nowindows`** attribute to "True".
- **Disable all IRC commands.** This sets the plugin's `noirc` attribute to "True".
-

Public and Private Command

Both of these templates generate the code necessary to create commands triggered by public or private chat. Clicking on either of these templates' menu items brings up a dialog where you can enter all the custom values for the new plugin; the dialog works (and looks) the same as the dialog for **Erk** commands.