IRC Library for Python 3

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

Version 0.028

Summary

Erkle is a low level, event-driven IRC library for Python 3, designed for both quick and dirty IRC bots or for full blown IRC clients.

Philosophy

Erkle was designed with the following in mind:

- **Little or no boilerplate code.** To write a IRC bot or client, the programmer should only have to write the functionality he or she wants to employ.
- **Modular code.** Most IRC bots or clients will require some common functionality. The programmer shouldn't have to reinvent the wheel every time they create a new bot or client. Code should be able to be isolated into modules for repeated use.
- **Little or no software requirements.** The library should use the Python standard library over third party libraries whenever possible.
- **Whenever possible, the library should mirror the protocol.** Having an understanding of how the IRC protocol works should give an understanding of how the library works.
- **General purpose.** The library should be able to be used for IRC bots, IRC clients, or any other IRC-based purpose, with no preference for any one specific function.
- **Multiple clients, multiple connections.** The library should make it easy to create and maintain multiple IRC clients at once, from a single program.

Low Level

To use *Erkle*, understanding IRC and the IRC protocol is a necessity. *Erkle* is designed to be low level, meaning its interface is influenced by the protocol itself. Since there's no syntactic sugar to hide the difficult or complex parts of the protocol, *Erkle* code should be easy to understand if you understand the underlying protocol.

The IRC protocol is defined in a series of RFC documents:

- RFC 1459
- RFC 2812

Why not use Twisted or irclib?

I wanted an IRC library with as few requirements as possible, and didn't require subclassing. I also wanted a client that was small enough to be bundled with an application, rather than requiring a user to install it through **pip** or some other package manager. Last, I wanted a library that wasn't focused on writing IRC bots *or* writing IRC clients; I wanted a library that would work for both. Since I couldn't find this library, I decided to write *Erkle*.

Requirements

Erkle uses, for the most part, only modules in the Python standard library. To use SSL/TLS to connect to IRC servers, however, the **pyOpenSSL** library must be installed. To install this library via the Python package installer, **pip**, execute this command:

```
pip install pyOpenSSL
```

Python libraries

Erkle uses the following modules from the standard library:

- **`sys`**
- **`socket`**
- **`collections`**
- **`string`**
- **`threading`**
- **`ssl`** (only if it is available)

Erkle Object

Erkle is an object that creates and manages an IRC connection. *Erkle()* takes a single argument (see below) on creation, a *dict* that configures the IRC client.

Argument	Type	Description
`config`	*dict*	Sets the user information the client will use. The dictionary has two mandatory keys, and seven optional ones: • **nickname** (string) – Sets the nickname the client will use. The key can be shortened to "nick". Mandatory. • **username** (string) – Sets the username the client will use. The key can be shortened to "user". Optional; if this key is missing, the client will use the client's nickname as the username. • **realname** (string) – Sets the real name the client will use. The key can be shortened to "real". Optional; if this key is missing, the client will use the name of the library, the library's version, and "IRC Client" as the realname. • **alternate** (string) – Sets the nickname the client will use if the initial nickname is already taken. The key can be shortened to "alt". Optional; if this key is missing, the client will use the client's nickname with an underscore appended. • **server** (string) – Sets the host name or IP address of the IRC server the client will connect to. Mandatory. • **port** (integer) – Sets the server port the client will connect to. Optional; if this key is missing, port 6667 is used. • **password** (string) – Sets the password to be used to connect to the IRC server. Optional. • **ssl** (boolean) – Sets whether SSL/TLS is used to connect to the server. Optional; if this key is missing or is set to *False*, the client will *not* use SSL/TLS to connect to the server. • **encoding** (string) – Sets what string encoding the server uses. Optional; if this key is missing, UTF-8 is used as a default.

Methods (pre-connection)

Once the *Erkle* object is created, use the **`connect()`** or **`spawn()`** methods to cause the object to connect to the IRC server.

Method	Arguments	Description
`connect`	*None.*	Connects to the IRC server.
`spawn`	*None.*	Spawns a new thread, and connects to IRC using that thread. A reference to the created thread is stored in the *Erkle* object, which can be retrieved with the **`thread()`** method.
`disable`	• **tag** (string) • **...**	Prevents any hooked functions tagged with **tag** from executing. Any number of tags can be enabled, as long as they are passed as individual arguments. See *Tags*, page 8. This method can be called while the client is connected.
`enable`	• **tag** (string) • **...**	Removes **tag** from the list of disabled tags; hooked functions with this tag will be executed as normal. Any number of tags can be enabled, as long as they are passed as individual arguments. See *Tags*, page 8. This method can be called while the client is connected.
`tag`	• **tag** (string) • ...	Adds a **tag** to the *Erkle* object. Only hooked functions with that tag will be called when an event is triggered. See *Tags*, page 8. This method can be called while the client is connected. This method can be called while the client is connected.
`untag`	• **tag** (string) • **...**	Removes a **tag** from the *Erkle* object. See *Tags*, page 8. This method can be called while the client is connected. This method can be called while the client is connected.

Methods (post-connection)

The following methods can only be used after the *Erkle* object's **`connect()`** or **`spawn()`** method has been called.

Method	Arguments	Description
`thread`	*None.*	If *Erkle*'s connection was started with the **`spawn()`** method, the object's *Thread* object (see the Python documentation for the *Threading* library) will be returned; otherwise, **`None`** is returned.
`kill`	*None.*	If *Erkle*'s connection was started with the **`spawn()`** method, this will terminate the object's thread.
`send`	• **data** (string)	Sends a "raw" message to the IRC server; the message will not be processed in any way before being sent.
`privmsg`	• **target** (string) • **message** (string)	Sends a chat message to a channel or user. This can also be called via an alias: **`msg()`**
`action`	• **target** (string) • **message** (string)	Sends a CTCP action message to a channel or user. This can also be called via an alias: **`me()`**
`notice`	• **target** (string) • **message** (string)	Sends a notice to a user or channel.
`join`	• **channel** (string) • ***key*** (string)	Joins a channel.
`part`	• **channel** (string) • ***reason*** (string)	Leaves a channel.

`kick`	• **target** (string) • **channel** (string) • ***reason*** (string)	Kicks a user from a channel (the client must be a channel operator in the channel).
`ban`	• **channel** (string) • **mask** (string)	Bans any user who's nick/host/username matches a mask from a channel (the client must be a channel operator in the channel). See RFC 1459 for more information on masks.
`unban`	• **channel** (string) • **mask** (string)	Removes a channel ban from a channel (the client must be a channel operator in the channel).
`lock`	• **channel** (string) • **key** (string)	Sets a channel key on a channel (the client must be a channel operator in the channel).
`unlock`	• **channel** (string) • **key** (string)	Removes a channel key from a channel (the client must be a channel operator in the channel).
`mode`	• **target** (string) • **mode** (string)	Sets a mode on a channel or user. See RFC 1459 for more information on modes.
`invite`	• **user** (string) • **channel** (string)	Sends a channel invitation to a user.
`away`	• ***message*** (string)	Sets the client to "away" on the IRC server.
`back`	*None.*	Sets the client to "back" on the IRC server.
`whois`	• **user** (string)	Requests WHOIS data on a user from the server. When the WHOIS data is received, the **`whois`** event will be triggered.
`list`	*None.*	Requests a list of channels from the server. When the channel list is received, the **`list`** event will be triggered.
`quit`	• ***reason*** (string)	Disconnects from the IRC server. If *Erkle*'s connection was started with the **`spawn()`** method, this will terminate the object's thread.

Attributes

An *Erkle* object also has a number of attributes that store information about the server, client, and the *Erkle* object. Not all of these values will be available immediately; the values are populated as the server sends the appropriate data to the client. Most of these values should be available by the time the **`welcome`** event is triggered.

Attribute	Type	Description
`nickname`	*string*	The client's nickname.
`username`	*string*	The client's username.
`realname`	*string*	The client's realname.
`server`	*string*	The server's address.
`port`	*integer*	The server's port.
`password`	*string*	The password used to connect to the server, if there is one.
`usessl`	*boolean*	Whether SSL is being used for this connection or not.
`hostname`	*string*	The server's hostname.
`software`	*string*	The server's software.
`options`	*list*	A list of the options the server supports.
`network`	*string*	The network the server belongs to.

commands	*list*	A list of commands supported by the server.
maxchannels	*integer*	The maximum number of channels a client can join on the server.
maxnicklen	*integer*	The maximum number of characters allowed for a nickname on the server.
channellen	*integer*	The maximum number of characters allowed for a channel name on the server.
topiclen	*integer*	The maximum number of characters allowed for a channel topic on the server.
kicklen	*integer*	The maximum number of characters allowed for a kick message on the server.
awaylen	*integer*	The maximum number of characters allowed for an away message on the server.
maxtargets	*integer*	The maximum number of targets a message can be sent to on a server.
modes	*integer*	The maximum number of channel modes that can be set on the server.
chantypes	*list*	What channel types the server uses.
prefix	*list of lists*	What channel status prefixes the server uses; each entry contains a list with the first value being the status type, and the second value being the prefix used for that type.
chanmodes	*list*	What channel modes the server uses.
casemapping	*string*	The casemapping the server uses.
spoofed	*string*	If the client's host is spoofed by the server, then the spoofed host name will be stored here.
users	*dictionary of lists*	An in-memory database of channel users. The dictionary uses channel names for keys, and each dictionary entry is a list of the named channel's users.
topic	*dictionary*	An in-memory database of channel topics. The dictionary uses channel names for keys, and each dictionary entry is a string containing the named channel's topic (or *None* if the topic is blank or unknown).
channels	*list of lists*	An in-memory database of all the channels on a server. This attribute starts empty by default; it will only be populated if the *Erkle* **`list()`** method is called. Each entry in the list a list that contains, in this order: 0. channel name (string) 1. number of users in the channel (integer) 2. channel topic (string) (*None* if there's no topic)
uptime	*integer*	Reflects how many seconds have elapsed since the *Erkle* object's **`connect()`** or **`spawn()`** methods were called.
tags	*list*	A list of tags that has been applied to the *Erkle* object.

The "irc" decorator, tags, and events

Included with the *Erkle* object is the *irc* decorator. The *irc* decorator is used to decorate functions that should be executed when specific events occur; this is called "hooking" an event. *irc* exposes one method: **`event`**. To hook an event, pass the name of the event (as a string) as the only argument to the **`event`** method. For example, to hook an event named "connect", the decorator required would look like:

```
@irc.event("connect")
```

Events can be hooked to an unlimited number of functions. Function hooks will be executed in the order in which they were hooked.

Tags

Hooked functions can also have **tags**, which are any number of strings attached to the hooked function's event. To add tags to a hooked function, pass them as additional arguments (after the event) to the function's decorator. For example, to add the tags "myfunc" and "chat" to a function hooked to the "public" event, you would use:

```
@irc.event("public", "myfunc", "chat")
```

Tags are used with *Erkle*'s **`disable()`** and **`enable()`** methods (see *Methods on page 5).*

Hooked functions can have any number of tags, and tags do not have to be unique. Hooked functions that do *not* have any tags cannot be disabled with the **`disable()`** method.

The *Erkle* object can be tagged as well by using *Erkle*'s **`tag()`** and **`untag()`** methods. When a tag is added to the *Erkle* object, only hooked functions with that tag will be called when an event is triggered. *Erkle* objects can have multiple tags. By default, *Erkle* objects have no tags and will call every hooked function triggered by an event.

Events

There are 23 IRC events that can be hooked. The hooked function can take a number of different arguments, depending on the event. The first (and sometimes only) argument passed to every hooked function is **`connection`**, which is the *Erkle* object running the IRC connection.

Event	Arguments	Description
`connect`	• **Erkle object**	Triggered when the *Erkle* object connects to IRC.
`motd`	• **Erkle object** • **message** (string)	Triggered when the server's message of the day (MOTD) is received.
`welcome`	• **Erkle object**	Triggered when registration with the IRC server is complete.
`nick-taken`	• **Erkle object** • **nickname** (string)	Triggered when *Erkle*'s nickname is already taken during registration; **nickname** contains the new nickname.
`ping`	• **Erkle object**	Triggered when the IRC server sends *Erkle* a PING command.
`join`	• **Erkle object** • **nickname** (string) • **host** (string) • **channel** (string)	Triggered whenever a user joins a channel *Erkle* is in. **nickname** contains the user's nickname, **host** contains the user's host, and **channel** contains the name of the channel joined. This event will trigger when the *Erkle* object joins a channel as well.

part	• **Erkle object** • **nickname** (string) • **host** (string) • **channel** (string) • **reason** (string)	Triggered whenever a user leaves a channel *Erkle* is in. **nickname** contains the nickname of the user, **host** contains the user's host, **channel** contains the name of the channel, and **reason** contains the reason why the user quit. If no reason has been provided, **reason** will be set to **None**.
quit	• **Erkle object** • **nickname** (string) • **host** (string) • **reason** (string)	Triggered when a user quits the IRC server. **nickname** contains the user's nickname, **host** contains the user's host, and **reason** contains the reason why the user quit. If no reason has been provided, **reason** will be set to **None**.
nick	• **Erkle object** • **nickname** (string) • **host** (string) • **new_nickname** (string)	Triggered when a user changes their nickname. **nickname** contains the user's original nickname, **host** contains the user's host, and **new_nickname** contains the user's new nickname.
names	• **Erkle object** • **channel** (string) • **users** (list)	Triggered when *Erkle* generates a list of users in a specific channel. This list will be regenerated every time a user changes their nick, quits IRC, or leaves a channel. **channel** contains the name of the channel, and **users** contains a list of users in that channel. If the server is configured for it, each user entry will contain the user's nickname and host, in the form ***nickname*!*username*@*hostname***; otherwise, the entry will only contain the user's nickname. Channel status symbols ('@' for channel operators, '+' for voiced users, etc.) are prefixed to each user's nickname. Generated user lists are stored in the *Erkle* object, accessible in the **users** attribute.
public	• **Erkle object** • **nickname** (string) • **host** (string) • **channel** (string) • **message** (string)	Triggered when *Erkle* receives a public message. **nickname** contains the sender's nickname, **host** contains the sender's host, **channel** contains the name of the channel the message was sent to, and **message** contains the message contents.
private	• **Erkle object** • **nickname** (string) • **host** (string) • **message** (string)	Triggered when *Erkle* receives a private message. **nickname** contains the sender's nickname, **host** contains the sender's host, and **message** contains the message contents.
notice	• **Erkle object** • **sender** (string) • **message** (string)	Triggered when *Erkle* receives a notice message. **sender** contains the nickname of the sender, and **message** contains the message contents.
action	• **Erkle object** • **nickname** (string) • **host** (string) • **target** (string) • **message** (string)	Triggered when *Erkle* receives a CTCP action message. **nickname** contains the sender's nickname, **host** contains the sender's host, **target** contains the name of the channel or username the message was sent to, and **message** contains the message contents.
away	• **Erkle object** • **nickname** (string) • **reason** (string)	Triggered when *Erkle* receives an "away" notification.
back	• **Erkle object**	Triggered when *Erkle* unsets itself as "away".
topic	• **Erkle object** • **nickname** (string) • **host** (string) • **channel** (string) • **topic** (string)	Triggered when *Erkle* receives a channel topic update. **nickname** contains the topic setter's nickname, **host** contains the setter's host, **channel** contains channel name, and **topic** contains the channel's topic. If the topic is set to an empty string, **topic** is set to *None*.

`mode`	• **Erkle object** • **nickname** (string) • **host** (string) • **target** (string) • **mode** (string)	Triggered when *Erkle* receives a channel or user mode change notification. **nickname** contains the mode setter's nickname, **host** contains the setter's host, **target** contains the user or channel the mode applies to, and **mode** contains the modes (and mode parameters) being set. If the mode is being set by the server, **nickname** and **host** will be set to the server's hostname.
`kick`	• **Erkle object** • **nickname** (string) • **host** (string) • **channel** (string) • **target** (string) • **reason** (string)	Triggered whenever *Erkle* receives a kick notification. **nickname** contains the kicker's nickname, **host** contains the kicker's host, **channel** contains the channel being kicked from, **target** contains the nickname of the user being kicked, and **reason** contains the reason given for the kick. If no reason is provided, **reason** will be set to *None*.
`kicked`	• **Erkle object** • **nickname** (string) • **host** (string) • **channel** (string) • **reason** (string)	Triggered whenever *Erkle* is kicked from a channel. **nickname** contains the kicker's nickname, **host** contains the kicker's host, **channel** contains the channel being kicked from, and **reason** contains the reason given for the kick. If no reason is provided, **reason** will be set to *None*.
`invite`	• **Erkle object** • **nickname** (string) • **host** (string) • **channel** (string)	Triggered whenever *Erkle* receives a channel invitation. **nickname** contains the inviter's nickname, **host** contains the inviter's host, and **channel** contains the channel *Erkle* is being invited to.
`whois`	• **Erkle object** • **nickname** (string) • **username** (string) • **host** (string) • **realname** (string) • **server** (string) • **idle** (integer) • **signon** (string) • **channels** (list) • **privileges** (string)	Triggered whenever *Erkle* receives WHOIS data from the server. **nickname** contains the user's nickname, **username** contains the user's username, **host** contains the user's host, **realname** contains the user's realname, **server** contains the server the user is connected to, **idle** contains the number of seconds the user has been idle, **signon** contains the timestamp of when the user signed on to the server, **channels** contains a list of channels (with status) the user is in, and **privileges** contains any special privileges the user has (or *None* if the user has none).
`list`	• **Erkle object** • **channels** (list of lists)	Triggered whenever *Erkle* receives a channel list from the server. Each entry in **channels** is a list that contains, in this order: 0. channel name (string) 1. number of users in the channel (integer) 2. channel topic (string) (*None* if there's no topic)
`line`	• **Erkle object** • **line** (string)	Triggered whenever *Erkle* receives a line of data from the server.
`error`	• **Erkle object** • **code** (string) • **subject** (string) • **reason** (string)	Triggered whenever *Erkle* receives an error message from the server. **code** is the error's code (from the IRC RFC documents), **subject** is the "subject" of the error (if there is no "target", **subject** will be set to *None*), and **reason** contains a short description of the error.

Erkle's **`connect()`** is a blocking function, so hooked functions should be declared *before* **`connect()`** is called.

For programs with multiple IRC connections (and, thus, multiple *Erkle* objects), understand that hooked events apply to *every* connection. So, if you hook the "public" event to a function, that function will be called when *every connection triggers a "public" event*. To restrict some hooked functions to a specific *Erkle* object, add a tag to the object with the **`tag()`** method, and add that tag to any hooked functions you'd like the object to call.

Event sets

Erkle contains a few sets of pre-written event handlers; they reside in the **erkle.events** package. To use an event set, simply import it.

Package	**erkle.events.dump**
Hooks	**action**, **away**, **back**, **connect**, **join**, **kick**, **kicked**, **mode**, **motd**, **names**, **nick**, **nick-taken**, **notice**, **part**, **ping**, **private**, **public**, **quit**, **topic**, **welcome**
Description	Prints event-specific data from every **Erkle** event to the console.

Package	**erkle.events.messages**
Hooks	**action, notice**, **private**, **public**
Description	Prints incoming messages to the console.

Examples

Greeter Bot

Here's an example bot that connect to an IRC server, join a channel, and greets everyone who joins that channel by name:

```
from erkle import *

CHANNEL = "#erklebot"

config = {
     'nickname': 'greetbot',
     'server': 'irc.efnet.org'
}

@irc.event("welcome")
def welcomed(connection):
  connection.join(CHANNEL)

@irc.event("join")
def joined(connection,nickname,host,channel):
  connection.msg("Welcome to "+CHANNEL+", "+nickname+"!")

bot = Erkle(config)
bot.connect()
```

Auto-Op Bot

This bot will automatically grant channel operator status to any user in a list of nicknames contained in the script. The bot will have to be granted channel operator status by another channel operator, however.

```
from erkle import *

CHANNELS = [ "#erklebot", "#erklesupport", "#pythonfans" ]
OPERATORS = [ "alice", "bob", "carol", "dave" ]

config = {
     'nick': 'greetbot',
     'server': 'irc.efnet.org', 'port': 6667
}

@irc.event("welcome")
def welcomed(connection):
  for channel in CHANNELS:
    connection.join(channel)

@irc.event("join")
def joined(connection,nickname,host,channel):
  if nickname in OPERATORS:
    connection.mode(channel,"+o "+nickname)
    connection.privmsg(nickname,"Welcome back, "+nickname)

bot = Erkle(config)
bot.connect()
```

...

License

MIT License