An IRC Library for Python 3

Version 0.0622

SUMMARY

REQUIREMENTS

THE ERKLE OBJECT

THE "IRC" DECORATOR, TAGS, AND EVENTS

EVENT SETS

CUSTOM DATA CLASSES

EXAMPLES

LICENSE

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

Summary

Erkle is a low level, event-driven IRC library for Python 3, designed for IRC bots, IRC clients, or any other IRC-related purpose.

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 connections/clients at once, from a single program.
- **Not a "black box".** The library should not prevent the user from using the library to perform functions not envisioned by the library's author. Important internal objects and functionality should be easily available to the user.

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 can be hard to understand if you don't 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 any IRC-related activity. 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`**
- **`os`**
- **`socket`**
- **`collections`**
- **`string`**
- **`threading`**
- **`random`**
- **`ipaddress`**
- **`urllib.request`**
- **`ssl`** (only if it is available)

The Erkle Object

The *Erkle* object's constructor takes two mandatory arguments, and a number of optional key word arguments.

```
class erkle.Erkle(nickname, server, port=6667, username="nickname",
      realname="real name", alternate="other_nick", password=None, ssl=False,
      cert_authority=None, certificate=None, key=None, verify_host=False,
      verify_cert=False, encoding="utf-8", flood_protection=True, flood_rate=2,
      clock_resolution=0.25, tick_frequency=1.0, debug_input=False,
      debug_output=False, multithreaded=False, daemon=False, socket=None,
      fetch_external_ip=False, external_ip_source="http://myexternalip.com/raw",
      external_ip="127.0.0.1")
```

Mandatory arguments

- **nickname**
  - The nickname the *Erkle* object will use when it connects to an IRC server (string).
- **server**
  - The hostname or IP address of the IRC server to connect to (string).

Optional key word arguments

Any of these key words can be used with *Erkle*'s **`configure()`** method to change or set options after the *Erkle* object's creation.

Key word	Type	Default	Description
`port`	*int*	6667	Sets the server port the client will connect to.
`username`	*str*	*The client's nickname*	Sets the username the client will use. The key can be shortened to "user".
`realname`	*str*	"Erkle 0.0622 IRC Client"	Sets the real name the client will use. The key can be shortened to "real".
`alternate`	*str*	*Nickname followed by an underscore*	Sets the nickname the client will use if the initial nickname is already taken. The key can be shortened to "alt".
`password`	*str*	*None*	Sets the password to be used to connect to the IRC server.
`ssl`	*bool*	*False*	Sets whether SSL/TLS is used to connect to the server.
`cert_authority`	*str*	*None*	Loads a set of "certificate authority" certificates from a file.
`certificate`	*str*	*None*	If SSL/TLS is being used to connect, this sets the location of the client's certificate file (if needed).
`key`	*str*	*None*	If SSL/TLS is being used to connect, this sets the location of the client's key file (if needed).
`verify_host`	*bool*	*False*	If SSL/TLS is being used to connect, set this to **`True`** to verify the server's host name.
`verify_cert`	*bool*	*False*	If SSL/TLS is being used to connect, set this to **`True`** to verify the server's certificate.

Key word	Type	Default	Description
socket	*module*	*None*	To use a different module for networking, set this to the new module's object. The module must support *Python's* ***socket*** *library API*.
encoding	*str*	"utf-8"	Sets what string encoding the server uses.
flood_protection	*bool*	*True*	Sets whether *flood protection* is used or not.
flood_rate	number	2	Sets how often messages are sent to the server if flood protection is turned on, in seconds. This value can be either an integer or a float.
clock_resolution	number	0.25	How often the uptime clock is updated, in seconds. Lower numbers can make flood protection timing more accurate. This value can be either an integer or a float.
tick_frequency	number	1.0	Sets how often the **tick** event is triggered, in seconds. See *Events*. This value can be either an integer or a float.
multithreaded	*bool*	*False*	Executes *Erkle*'s **connect()** method in a separate thread.
daemon	*bool*	*False*	If *Erkle* is to run in a separate thread, the *thread will be daemonized*.
debug_input	*bool*	*False*	Set this to *True* to print all incoming IRC messages to the console.
debug_output	*bool*	*False*	Set this to *True* to print all outgoing IRC messages to the console.
fetch_external_ip	*bool*	*False*	Set this to *True* to attempt to retrieve *Erkle*'s external IP address; if *False*, "127.0.0.1" is used as *Erkle*'s IP address.
external_ip_source	*str*	"http://myexternalip.com/raw"	Set this to the URL *Erkle* will use to retrieve its external IP address, if using the **fetch_external_ip** option.
external_ip	*str*	"127.0.0.1"	Set this to the external IP address that *Erkle* will use.

Methods

Usable before connecting to an IRC server

Once the *Erkle* object is created, use its **`connect()`** method to cause the object to connect to the IRC server.

Method	Arguments	Description
`configure`	*Key words*	Any options set by the keywords available in *Erkle*'s constructor can be set with this method. See *Optional key word arguments*. This method *cannot* be called while the client is connected to IRC.
`connect`	*None.*	Connects to the IRC server. If the **`multithreaded`** configuration option is set to *True*, the IRC client/connection will be executed in a separate thread.
`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*. 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*. 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*. 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*. This method can be called while the client is connected. This method can be called while the client is connected.

Usable after connecting to an IRC server

The following methods can only be used after the *Erkle* object's **`connect()`** method has been called. If flood protection is turned on (the default), any outgoing messages to the server are added to a queue and sent at a rate of rate of one message every two seconds.

Method	Arguments	Returns	Description
`thread`	*None.*	*Nothing*	If *Erkle* has the **`multithreaded`** configuration option set to *True*, the object's *Thread* object (see *the Python documentation for the Threading library*) will be returned; otherwise, **`None`** is returned.
`kill`	*None.*	*Nothing*	If *Erkle* has the **`multithreaded`** configuration option set to *True*, this will terminate the object's thread.
`socket`	*None.*	*Nothing*	Returns the socket the *Erkle* object is using for the IRC connection.
`send`	• **data** (string)	*Nothing*	Sends a "raw" message to the IRC server; the message will not be processed in any way before being sent. This method can be used to send commands to the IRC server that don't have a corresponding *Erkle* method. For example, to send the IRC operator rehash command, you could call ***`Erkle`*`.send("REHASH")`**.

Method	Arguments	Returns	Description
`privmsg`	• **target** (string) OR **target** (list) • **message** (string)	*Nothing*	Sends a chat message to a channel or user. This can also be called via an alias: `msg()`. Pass a list of channels or users to send a PRIVMSG to multiple users or channels.
`cprivmsg`	• **nickname** (string) • **channel** (string) • **message** (string)	*Nothing*	Sends a private message to **nickname** in **channel** that bypasses flood protection limits. Both the target nickname and the client must be in the same channel, and the client must be a channel operator. Not all servers support the CPRIVMSG command; if the server currently connected to supports CPRIVMSG it should be in the client's `commands` attribute.
`action`	• **target** (string) OR **target** (list) • **message** (string)	*Nothing*	Sends a CTCP action message to a channel or user. This can also be called via an alias: `me()`. Pass a list of channels or users to send a CTCP action to multiple users or channels.
`notice`	• **target** (string) OR **target** (list) • **message** (string)	*Nothing*	Sends a notice to a user or channel.. Pass a list of channels or users to send a NOTICE to multiple users or channels.
`cnotice`	• **nickname** (string) • **channel** (string) • **message** (string)	*Nothing*	Sends a channel NOTICE to **nickname** in **channel** that bypasses flood protection limits. Both the target nickname and the client must be in the same channel, and the client must be a channel operator. Not all servers support the CNOTICE command; if the server currently connected to supports CNOTICE it should be in the client's `commands` attribute.
`ctcp`	• **target** (string) • **command** (string) • **message** (string)	*Nothing*	Sends a CTCP command/message to a channel or user. Pass a list of channels or users to send to multiple users or channels.
`oper`	• **username** (string) • **password** (string)	*Nothing*	Logs into an IRC operator account on the server. If the login is successful, the server will trigger an `oper` event.
`join`	• **channel** (string) • ***key*** (string)	*Nothing*	Joins a channel.
`part`	• **channel** (string) • ***reason*** (string)	*Nothing*	Leaves a channel.
`kick`	• **target** (string) • **channel** (string) • ***reason*** (string)	*Nothing*	Kicks a user from a channel (the client must be a channel operator in the channel).
`ban`	• **channel** (string) • **mask** (string)	*Nothing*	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)	*Nothing*	Removes a channel ban from a channel (the client must be a channel operator in the channel).
`lock`	• **channel** (string) • **key** (string)	*Nothing*	Sets a channel key on a channel (the client must be a channel operator in the channel).
`unlock`	• **channel** (string) • **key** (string)	*Nothing*	Removes a channel key from a channel (the client must be a channel operator in the channel).
`mode`	• **target** (string) • **mode** (string)	*Nothing*	Sets a mode on a channel or user. See *RFC 1459* for more information on modes.

Method	Arguments	Returns	Description
`invite`	• **user** (string) • **channel** (string)	*Nothing*	Sends a channel invitation to a user.
`away`	• ***message*** (string)	*Nothing*	Sets the client to "away" on the IRC server.
`back`	*None.*	*Nothing*	Sets the client to "back" on the IRC server.
`whois`	• **user** (string)	*Nothing*	Requests WHOIS data on a user from the server. When the WHOIS data is received, the **`whois`** event will be triggered.
`list`	*None.*	*Nothing*	Requests a list of channels from the server. When the channel list is received, the **`list`** event will be triggered.
`quit`	• ***reason*** (string)	*Nothing*	Disconnects from the IRC server. If *Erkle* is running in **`multithreaded`** mode, this will terminate the object's thread.
`dccchat`	• **nickname** (string) • **port** (integer) • ***host_ip*** (string)	integer	Sends a DCC chat request to a user. **nickname** is the nickname of the user to chat with, **port** is what port number to use, and **host_ip** should contain the IP address the other user should use to connect to chat; if **host_ip** is not passed as an argument, *Erkle* will use the external IP address of the computer running the *Erkle* object. This method returns a unique integer called the connection's **client ID**; this is used with the **`chat()`** method.
`chat`	• **clientid** (integer) • **message** (string)	*Nothing*	Sends a message to a connected user via DCC chat. **clientid** is the client ID of the DCC chat session (obtained from the **`dccchat()`** method or any of the DCC chat events.
`closechat`	• **clientid** (integer)	boolean	Closes a DCC chat session. **clientid** is the client ID of the DCC chat session. Returns *True* if the client ID was found and closed, and *False* if the client ID was *not* found.

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 **`registered`** 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.
`certificate`	*dictionary*	If SSL is being used for this connection, this will store a dictionary representation of the server's certificate (see *getpeercert()* from Python's **`ssl`** library).

Attribute	Type	Description
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. The list will only contain users in channels the *Erkle* object has joined and is still present in. Each list entry is a *UserData* class object (see *UserData class*).
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 is a *ChannelData* class object (see *ChannelData class*).
uptime	*integer*	Reflects how many seconds have elapsed since the *Erkle* object's **connect()**method was called.
tags	*list*	A list of tags that has been applied to the *Erkle* object.

Attribute	Type	Description
`encoding`	*string*	The string encoding scheme the *Erkle* object is using.
`multithreaded`	*boolean*	If *Erkle* is running in a separate thread, this is will be *True*; otherwise, this will be *False*.
`external_ip`	*string*	The external IP address of the computer running 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*).

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.

Hooked functions with an asterisk (*) tag will be executed by every *Erkle* object, regardless of how the object is tagged. Since this is a special tag, *Erkle* objects cannot have this tag, and an error will be thrown if an asterisk is added or removed from *Erkle* object's tags.

To see an example of *Erkle* tag usage, see *Using client and event tags*.

Events

There are 33 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
`connecting`	• **Erkle object**	Triggered when the *Erkle* object starts the connection process.
`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.
`registered`	• **Erkle object**	Triggered when registration with the IRC server is complete.

Event	Arguments	Description
`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.
`joined`	• **Erkle object** • **channel (string)**	Triggered whenever *Erkle* joins a channel. **channel** contains the name of the joined channel.
`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`.
`parted`	• **Erkle object** • **channel (string)**	Triggered whenever *Erkle* leaves a channel. **channel** contains the name of the parted channel.
`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_nick** (string)	Triggered when a user changes their nickname. **nickname** contains the user's original nickname, **host** contains the user's host, and **new_nick** 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. Each entry in the list is a *UserData* class object (see *UserData class*). 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.

Event	Arguments	Description
`ctcp`	• **Erkle object** • **nickname** (string) • **host** (string) • **target** (string) • **command** (string) • **message** (string)	Triggered when *Erkle* receives a CTCP 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, **command** contains the CTCP command sent, and **message** contains the data or message sent.
`ctcp-reply`	• **Erkle object** • **sender** (string) • **command** (string) • **message** (string)	Triggered when *Erkle* receives a CTCP reply. **sender** contains the sender, **command** contains the CTCP command, and **message** contains the data or message sent.
`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** • **whois** (string)	Triggered whenever *Erkle* receives WHOIS data from the server. **whois** contains a *WhoisData* class object (see ***WhoisData class***).
`list`	• **Erkle object** • **channels** (list of lists)	Triggered whenever *Erkle* receives a channel list from the server. Each entry in **channels** is a *ChannelData* class object (see ***ChannelData class***). Generated channel lists are stored in the *Erkle* object, accessible in the ***channels*** attribute.
`oper`	• **Erkle object**	Triggered when *Erkle* is granted IRC operator status (usually the result of the `oper()` method being called on the *Erkle* object).

Event	Arguments	Description
`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.
`tick`	• **Erkle object**	Triggered once per second by default. The interval can be changed with the **clock-frequency** option (see *The Erkle Object*).
`dcc-chat`	• **Erkle object** • **nickname** (string) • **clientid** (integer) • **message** (string)	Triggered whenever *Erkle* receives a DCC chat message. **nickname** contains the nickname of the sending user, **clientid** contains the connection's client ID, and **message** contains the message sent. **clientid** is used with the **`chat()`** and **`closechat()`** methods (see *Usable after connecting to an IRC server*).
`dcc-chat-accept`	• **Erkle object** • **nickname** (string) • **address** (string) • **port** (integer) • **clientid** (integer)	Triggered whenever *Erkle* begins a DCC chat session with a user. **nickname** contains the nickname of the connected user, **address** contains the IP address of the sender, **port** contains the port the user is using, and **clientid** contains the connection's client ID. **clientid** is used with the **`chat()`** and **`closechat()`** methods (see *Usable after connecting to an IRC server*).
`dcc-chat-end`	• **Erkle object** • **nickname** (string) • **address** (string) • **port** (integer) • **clientid** (integer)	Triggered whenever *Erkle* ends a DCC chat session with a user. **nickname** contains the nickname of the disconnected user, **address** contains the IP address of the sender, **port** contains the port the user is using, and **clientid** contains the connection's client ID. **clientid** is used with the **`chat()`** and **`closechat()`** methods (see *Usable after connecting to an IRC server*).

Erkle's **`connect()`** is a blocking function (if not ran in **`multithreaded`** mode), 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. To make sure a hooked function executes on *all* connections, apply the **`*`** tag to it.

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. This type of module is called *event modules* (see *Using event modules with Erkle*).

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.

Custom data classes

ChannelData class

The *ChannelData* class is used by the *Erkle* client to present results from the LIST command. *ChannelData* has no methods, only attributes.

Attribute	Type	Description
`name`	*string*	The channel's name.
`users`	*integer*	How many users are in the channel.
`topic`	*string*	The channel's topic; if there is no topic, this will be set to *None*.

UserData class

The *UserData* class is used by the *Erkle* client to store user information. *UserData* has no methods, only attributes.

Attribute	Type	Description
`nickname`	*string*	The user's nickname.
`username`	*string*	The user's username.
`host`	*string*	The user's host.
`op`	*boolean*	*True* if the user is a channel operator, *False* if not.
`voiced`	*boolean*	*True* if the user is voiced, *False* if not.
`owner`	*boolean*	*True* if the user is a channel owner, *False* if not.
`admin`	*boolean*	*True* if the user is an admin, *False* if not.
`halfop`	*boolean*	*True* if the user is a channel half-op, *False* if not.

WhoisData class

The *WhoisData* class is used by *Erkle* to present results from the WHOIS command. *WhoisData* has no methods, only attributes.

Attribute	Type	Description
`nickame`	*string*	The user's nickname.
`username`	*string*	The user's username.
`host`	*string*	The user's host.
`realname`	*string*	The user's real name.
`privs`	*string*	The user's privileges; if the user has none, this will be set to *None*.
`idle`	*integer*	How long the user has been idle, in seconds.
`signon`	*integer*	When the user connected to IRC as a UNIX epoch timestamp.
`channels`	*list*	What channels the user is in.
`server`	*string*	What server the user is connected to.

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"

# Join #erklebot as soon as we connect
@irc.event("registered")
def welcomed(connection):
     connection.join(CHANNEL)

# Greet everyone who joins #erklebot
@irc.event("join")
def joined(connection,nickname,host,channel):
     connection.msg("Welcome to "+CHANNEL+", "+nickname+"!")

# Create the IRC client
bot = Erkle('greetbot','irc.efnet.org')

# Connect to IRC
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" ]

# Join all channels in the CHANNEL list as soon
# as we connect
@irc.event("registered")
def welcomed(connection):
     for channel in CHANNELS:
          connection.join(channel)

# If anyone with a nickname in the OPERATORS list
# joins a channel the client is in, send a mode
# message to the server to grant channel operator status
@irc.event("join")
def joined(connection,nickname,host,channel):
     if nickname in OPERATORS:
          connection.mode(channel,"+o "+nickname)
          connection.privmsg(nickname,"Welcome back, "+nickname)

# Create the client
bot = Erkle('greetbot','irc.efnet.org',port=6667)

# Connect the client to the IRC server
bot.connect()
```

Using configure()

This bot will connect to an IRC server and join a channel. If passed no command line arguments, it will connect to the Freenode IRC network via TCP/IP; if passed "ssl" as a command line argument, it will use SSL/TLS to connect to Freenode.

```
import sys
from erkle import *

# Join all channels in the CHANNEL list as soon
# as we connect
@irc.event("registered")
def welcomed(connection):
     connection.join("#erklelib")

# Create the client
bot = Erkle('greetbot','chat.freenode.net')

connect_via_ssl = False
if len(sys.argv)>1:
     if sys.argv[1].lower()=="ssl":
          connect_via_ssl = True

if connect_via_ssl:
     bot.configure(ssl=True,port=6697)
else:
     bot.configure(port=6667)

# Connect the client to the IRC server
bot.connect()
```

Using client and event tags

This bot will connect to the same channel on two different IRC networks and display any public chat. Each client will use a tagged "public" event that only that client will execute.

```
from erkle import *

CHANNEL =  "#erklebot"

# This event will only be triggered when the client
# connected to EFnet gets a "public" event
@irc.event("public", "EFnet")
def public(connection,nickname,host,channel,message):
     print("EFnet->"+nickname+": "+message)

# This event will only be triggered when the client
# connected to Rizon gets a "public" event
@irc.event("public", "Rizon")
def public(connection,nickname,host,channel,message):
     print("Rizon->"+nickname+": "+message)

# This event will be triggered by both clients
@irc.event("registered", "*")
def welcomed(connection):
     connection.join(CHANNEL)

# Create the EFnet client
efnet_bot = Erkle('multibot','irc.efnet.org',multithreaded=True)

# Add the "EFnet" tag to the EFnet client
efnet_bot.tag("EFnet")

# Create the Rizon client
rizon_bot = Erkle('multibot','irc.rizon.org',multithreaded=True)

# Add the "Rizon" tag to the Rizon client
rizon_bot.tag("Rizon")

# Connect both clients to IRC, each on
# their own thread
efnet_bot.connect()
rizon_bot.connect()
```

Using event modules with *Erkle*

Hooked event functions can be written in Python modules and included in *Erkle* programs by only using **`import`**. This is not a function of *Erkle*, this is just how Python works when it loads modules. A module containing *Erkle* hooked event functions is called an *event module*.

As an example of this, we're going to write a module that automatically joins a specific channel when *Erkle* connects to an IRC server.

Open a file named ***joiner.py***, and write the following to it and save the file:

```
# Import Erkle, so we can use the "irc" function decorator
from erkle import *

# Join #erklebot as soon as we connect
@irc.event("registered")
def welcomed(connection):
     connection.join("#erklebot")
```

Now, in the same directory that you saved ***joiner.py***, open a file named ***modexample.py*** (or whatever else you'd like to name it), and write the following to it and save:

```
from erkle import *

# Import our "joiner.py" module
import joiner

# Create our Erkle object
joinbot = Erkle('erklemod','irc.efnet.org',alternate='erk1em0d')

# Connect to IRC
joinbot.connect()
```

Now, when we run ***modexample.py***, our *Erkle* bot will join "#erklebot" as soon as it connects to the IRC server.

DCC chat "partyline" bot

A "partyline" is a kind of private chat; think of it as a mini-IRC server built into an IRC bot. Users can send "partyline" as a private message to the bot, and the bot will connect to the user via DCC. Any DCC chat messages sent to the bot will be forwarded to all other connected users.

Partyline event module

We're going to implement the bot's functionality as an event module, making it easy to add a partyline to any *Erkle* bot:

```
# partyline.py

import random
from erkle import *

PARTYLINE_USERS,USED_PORTS = [],[]

# Offer a DCC chat connection to anyone who
# sends "partyline" as a private message
@irc.event("private")
def fevent(connection,nickname,host,message):
     if message.lower().strip()=="partyline":
          # Select a random port number from
          # 10,000 to 60,000
          rport = random.randint(10000,60000)
          while rport in USED_PORTS:
               rport = random.randint(10000,60000)
          USED_PORTS.append(rport)

          # Connect user to DCC chat
          connection.dccchat(nickname,rport,"127.0.0.1")

# A user sent a DCC chat message to the partyline
@irc.event("dcc-chat")
def fevent(connection,nickname,clientid,message):
     # Format chat for broadcast
     outmsg = "<"+nickname+"> "+message

     # Broadcast the chat to all clients except
     # the user who sent the chat
     broadcast(connection,clientid,outmsg)

# A user has connected to the partyline
@irc.event("dcc-chat-accept")
def fevent(connection,nickname,address,port,clientid):
     # Add the user to the list of partyline users
     if not clientid in PARTYLINE_USERS:
          PARTYLINE_USERS.append(clientid)

     # Welcome the new user to the partyline
     connection.chat(clientid,"Welcome to the erklebot partyline!")

     # Notify everyone that the user has joined the partyline
     broadcast(connection,clientid,nickname+" has joined the partyline.")
```

Code continues on the next page.

```
# A user has disconnected from the partyline
@irc.event("dcc-chat-end")
def fevent(connection,nickname,address,port,clientid):
     # Remove the user from the list of partyline users
     try:
          PARTYLINE_USERS.remove(clientid)
     except:
          pass

     # Notify everyone else that the user left the partyline
     broadcast(connection,0,nickname+" has left the partyline.")

# broadcast()
# Sends DCC chat messages to all users except
# for one (the user who sent out the DCC chat
# message. Set "clientid" to 0 (zero) to send
# to all users in the partyline.
def broadcast(connection,clientid,message):
     for user in PARTYLINE_USERS:
          if user != clientid:
               connection.chat(user,message)
```

Save this code to a file named "partyline.py". With our event module written, all we have to do is use it:

```
from erkle import *

import partyline

# Create our Erkle object
partybot = Erkle('erkleparty','irc.efnet.org')

# Connect to IRC
partybot.connect()
```

Example usage

```
     alice | partyline
         * | Received a DCC CHAT offer from erkleparty
         * | DCC CHAT connection established to erkleparty [127.0.0.1:24775]
erkleparty | Welcome to the erklebot partyline!
erkleparty | bob has joined the partyline.
     alice | hello, bob! welcome to the partyline!
erkleparty | <bob> thanks, alice. it's nice to be here!
```

An example partyline session, using HexChat

License

MIT License