Bones IRC Bot Documentation

Release 0.2.0-DEV

404'd

October 01, 2015

Contents

The Bones IRC Bot is a bare bones IRC Bot made with extensibility in mind. It provides an easy to use API for writing Bones modules which is the base of the bot itself. The bot is by default an empty shell which does work like managing connections, configurations, error handling and providing and implementing the API itself.

A basic Bones module may be made in as few lines as this:

```
import bones.bot
import bones.event

class Greeter(bones.bot.Module):

    @bones.event.handler(event=bones.event.UserJoinEvent)
    def greet_user(self, event):
        event.user.msg(
            "Welcome to %s, %s!" % (event.channel.name, event.user.nickname)
        )
```

The documentation on this page is just as much documentation of the API as it is of the bot itself. No matter whether you just want to make something new or if you want to help out, this documentation will cover most of it. Note that internal methods may appear here because of this even though they're not meant to be used and/or a part of the public API.

CHAPTER 1

Getting Started

1.1 Installation

1.1.1 Getting a copy of the source

There's basically two ways to install Bones, the version-controlled way and the uncontrolled way. The version-controlled way is recommended because it makes upgrading and pull request merging easier, in case you'd want to try out something that haven't arrived yet or want to try out a development-version.

> **Warning:** Development releases must **always** be installed from version-controlled source from the branch `develop`, or from an archived copy of the `develop` branch. This is partly because there will be no tags for releases which is currently under development. You should always use VCS for development versions to make it easier to update your local copy.

Using version-control

Using `git` we'll clone the repository where the Bones IRC Bot source code is contained, and then check out a copy of the source of the current Bones release, v0.2.0-DEV.

```
git clone https://github.com/404d/Bones-IRCBot
cd Bones-IRCBot
git checkout tags/v0.2.0-DEV
```

If the release you're trying to install is a development version, you should run this instead of the last step above:

```
git checkout develop
```

Using archived releases

We'll just download an archived and compressed copy of the source code from Github and uncompress that. If you're installing a released version, run these commands in your shell:

```
wget https://github.com/404d/Bones-IRCBot/archive/v0.2.0-DEV.tar.gz
tar -xzf v0.2.0-DEV.tar.gz
cd Bones-IRCBot-v0.2.0-DEV
```

If you're installing a development version, run these commands instead:

```
wget https://github.com/404d/Bones-IRCBot/archive/develop.tar.gz
tar -xzf develop.tar.gz
cd Bones-IRCBot-develop
```

1.1.2 Installing Requirements and Dependencies

From now on we'll assume you already have a working environment with pip installed. The dependency tree for Bones may look weird for some, but basically just skip the headers that doesn't look like they're for you.

Note: Module usually refers to Bones modules, which is Python classes that add functionality to the bot. However it may also refer to Python modules which may contain Python code and classes. All Bones modules are contained within a Python module, so be sure that you know which one of the two you're talking/reading about.

Installing base dependencies

Bones IRC Bot is based on Twisted, mainly because the aim of the Bones bot is to provide an easily usable and extensive API in order to make scripting easier for developers. However SSL is not supported by default and as such pyOpenSSL is a dependency for SSL connections to work.

```
pip install twisted pyopenssl
```

If you really don't care for SSL support you can just remove `pyOpenSSL` from the command above. Bones isn't stupid and will only try to do anything with pyOpenSSL if it is available on the system and just carry on if it's unneeded.

Installing module dependencies

Bones by itself does next to nothing; all functionality is provided by modules, and some of these modules have dependencies of themselves, maybe Python modules, Bones modules or something in between.

When it comes to all the default modules, there's about two dependencies you need to think of. All the modules who parses and fetches information from websites uses BeautifulSoup 4 for parsing the HTML trees, and a lot of the plugins that work with some sort of data storage uses a database for storage through the `bones.modules.storage.Database` Bones module and the SQLAlchemy Python module. To install the dependencies of the bundled modules, run this command in your shell:

```
pip install beautifulsoup4 sqlalchemy
```

Note: SQLAlchemy may require additional dependencies depending on what kind of database you're going to use. For more information about this, read about SQLAlchemy Dialects.

1.2 Getting started with modules

Bones modules are basically just normal Python classes with a familiar interface so that the bot may load them on initialization.

1.2.1 Creating a dummy module

We'll start off with something easy; a module that can be loaded by the bot but does absolutely nothing. To do this we'll import the Python module *bones.bot* and write a class that inherits from the class *Module*. The class itself will be empty. The class will be saved as `module.py` inside a directory named `tutorial`.

```
import bones.bot

class DummyModule(bones.bot.Module):
    pass
```

Note: Make sure there exists a file named `__init__.py` inside the `tutorial` folder, if not the module won't load because Python won't recognize `tutorial` as a package.

1.2.2 Loading the dummy module

Now that the module is ready to be used we'll need to tell the bot to load it. First up open up your configuration file and find the line that starts with `modules =`. This is where we'll add the module path in order to load it.

Now if you followed the previous section to every little detail you'll have the module inside the file `tutorial/module.py`. To find out what your module path is, we'll use dot-notation. Dot-notation is the way you refer to modules and classes in `import` statements, so the name of each folder, file and class in the path is separated by a punctuation mark (therefore dot-notation). As the module is named `DummyModule` inside the file `tutorial/module.py`, the path to it is `tutorial.module.DummyModule`

To load the module into Bones we'll append the path to the `modules` list in your configuration file. Let's take this example here:

```
[bot]
nickname = Bones
username = bones
realname = Bones IRC Bot
channel = #Gameshaft
    #Temporals

modules = bones.modules.utilities.Utilities
;     bones.modules.services.NickServ
;     bones.modules.services.HostServ
```

In this configuration file the `modules` list already contains the module `bones.modules.utilities.Utilities`, so we'll need to add a new line beneath this one and indent it with 4 spaces. After that you can just add the module path, and the result will be this:

```
modules = bones.modules.utilities.Utilities
    tutorial.module.DummyModule
```

Save the file, boot the bot and you should see something like this in your log:

```
2013-10-20 22:45:02,865 - bones.bot - INFO - Loaded module bones.modules.funserv.UselessResponses
2013-10-20 22:45:02,866 - bones.bot - INFO - Loaded module tutorial.module.DummyModule
2013-10-20 22:45:02,868 - bones.bot - INFO - Connecting to server irc.chatno.de:+6697
2013-10-20 22:45:04,045 - bones.bot - INFO - Signed on as Bones_.
```

If one of the lines read `Loaded module tutorial.module.DummyModule` you've successfully "written" a working module!

1.3 Using Events

Bones, and the Bones API is heavily based on events. Whenever something happens you'll be able to find out in one way or another. To make your method listen to and act upon events, you should use the *bones.event.handler()* decorator. This decorator takes one of two keyword arguments, `trigger` and `event`, but we'll explain this soon.

1.3.1 Creating an event handler

Event handlers in Bones is simply a method that is hooked up to an event. Event handlers should take 1 argument, `event` which will be an object containing stuff relevant to the event, like the current bot instance or the user that sent a message.

We will now create a module that will greet a user when he joins the channel. To start off, create a new module like you did in the previous example, but name it `WelcomeBack`. Now you'll have something that looks like this:

```
1 import bones.bot
2
3 class WelcomeBack(bones.bot.Module):
4     pass
```

So far so good, but we want to make this module actually do something. To start off, we should import the package *bones.event*. Add the following line just below the import you've already got:

```
import bones.event
```

Now we're getting close to something, but it's still not doing anything. Now, make a method on the class and that takes two arguments, `self` and `event`:

```
def greetUser(self, event):
    pass
```

Now we'll take off on a tangent in order to explain something. `event` will contain everything we need. In our module, or to be specific this method, `event` will be an instance of *bones.event.UserJoinEvent*. This this object will have two attributes that are of importance to us; first being `user` and the other being `channel`. *User* will give us access to the user's nickname, and *Channel* will let us send messages to the channel. I advice you to click on those two links to read about the attributes and methods available on those two objects.

Ok, so what we need to do is 1) get the user's nickname and 2) send a message to the channel. The first problem can be solved by making use of `event.user.nickname`, and the other can be solved by using `event.channel.msg()`. To start off we should build our string:

```
greeting = "Welcome back, %s!" % event.user.nickname
```

Now that our message is ready, all that's left is sending it to our channel:

```
event.channel.msg(greeting)
```

We're almost done! There's just one problem left though: If you were to load your module now, nothing would happen when the user joins despite you having written code that should do something when that happens. But why is this? The answer isn't far away: you haven't turned your method into an event handler yet.

Event handlers are regular methods that gets tied to an even by using a decorator, in this case *bones.event.handler()*. In other to register this method as an event handler we need to know one thing: What is the class of the event we want to use? In this case we want to do something with *bones.event.UserJoinEvent*, so what we'll do is we'll pass that class as the `event` argument to *bones.event.handler()* and place the decorator above our method, like this:

```
@bones.event.handler(event=bones.event.UserJoinEvent)
def greetUser(self, event):
    ...
```

All that's left now is to add the new module to your configuration:

```
modules = bones.modules.utilities.Utilities
    tutorial.module.DummyModule
    tutorial.module.WelcomeBack
```

And you should be all set. For reference, here's what your file should look like, if we look away from the dummy module we made in our previous tutorial:

```
1 import bones.bot
2 import bones.event
3
4 class WelcomeBack(bones.bot.Module):
5
6     @bones.event.handler(event=bones.event.UserJoinEvent)
7     def greetUser(self, event):
8         greeting = "Welcome back, %s!" % event.user.nickname
9         event.channel.msg(greeting)
```

See also:

Events API

CHAPTER 2

API Documentation

2.1 Bot Client and Base API

2.1.1 BonesBot

class bones.bot.**BonesBot**(**args*, ***kwargs*)
Bases: twisted.words.protocols.irc.IRCClient

create_user(*target*)
Prepares a User object for the given target.

get_channel(*name*)
Returns the Channel object for the given channel.

get_user(*target*)
Returns the User object for the given target if it exists, None if otherwise.

signedOn()
Event called when the bot receives a registration confirmation from the server

2.1.2 BonesBotFactory

class bones.bot.**BonesBotFactory**(*settings*)
Bases: twisted.internet.protocol.ClientFactory

The Twisted client factory that provides connection management and configuration for each individual bot configuration.

sourceURL
A hardcoded string URL to the Bones IRC Bot repository. Sent to clients in response to a CTCP SOURCEURL query.

versionEnv
Currently unused. Sent to clients as a part of a CTCP VERSION reply.

versionName
The name of the bot. Sent to clients as a part of a CTCP VERSION reply.

versionNum
The release name of the current bot version. Sent to clients as a part of a CTCP VERSION reply.

clientConnectionFailed(*connector*, *reason*)
Called when an error occured with the connection. This method will take care of reconnecting the bot to the server after a variable time period.

clientConnectionLost(*connector*, *reason*)
Called when the connection to the server was lost. This method will take care of reconnecting the bot to the server after a variable time period.

connect()
Connects this bot factory to the server it is configured for. Gets called automatically by the default manager at boot.

loadModule(*path*)
Loads the specified module and adds it to the bot if it is a valid Bones module.

Parameters **path** (*str.*) – The Python dot-notation path to the module that should be loaded.

Raises `BonesModuleAlreadyLoadedException`, `InvalidBonesModuleException`, `InvalidConfigurationException`, `NoSuchBonesException`

2.1.3 Exceptions

exception `bones.bot.`**BonesModuleAlreadyLoadedException**

exception `bones.bot.`**InvalidBonesModuleException**

exception `bones.bot.`**InvalidConfigurationException**

exception `bones.bot.`**NoSuchBonesModuleException**

2.1.4 Module

class `bones.bot.`**Module**(*settings*, *factory*)
Bones module base class

Parameters

- **settings** (`bones.config.ServerConfiguration`) – The settings for the current server factory.
- **factory** (*bones.bot.BonesBotFactory*) – The bot factory that instanciated this module.

settings
A `bones.config.ServerConfiguration` instance containing all the currently loaded settings for this server factory and all its bots and modules.

factory
A *bones.bot.BonesBotFactory* instance representing the factory which instanciates the clients whom this module is used with.

See also:

Getting started with modules

2.2 Configuration API

2.3 Events API

This section details how events and the event system in the Bones IRC Bot works. For more detailed information regarding the IRC protocol, please read **RFC 1459**.

See also:

Using Events

2.3.1 Methods

`bones.event.`**`fire`**(*server*, *event*, **args*, ***kwargs*)

Call all event handlers with the specified event identifier registered to the provided server with the provided arguments.

This may be called in your Bones module to fire custom events.

Note: You should create a class that inherits from *`Event`*, and put your arguments inside this.

Warning: Using this to alter standard bot behaviour is not supported! This means doing anything like (but not limited to) firing events that are supposed to be handled by the bot core, for example *`PrivmsgEvent`*.

Parameters

- **`server`** (*str*) – the server tag that identifies the server this occured on
- **`event`** (*object*) – an event instance or event identifier
- **`callback`** (*callable*) – a callable that should be called after all event handlers have been triggered

`bones.event.`**`register`**(*obj*, *server*)

Look through a Module for registered event handlers and add them to the event handler list.

This is an internal function automatically called by the server bot factory while creating the factory. **Do not** call this in your modules!

Parameters

- **`obj`** (*`bones.bot.Module`*) – the `Module` instance to look at.
- **`server`** (*`bones.bot.BonesBot`*) – the server tag that the supplied module runs under.

2.3.2 Decorators

`bones.event.`**`handler`**(*event=None*, *trigger=None*)

Marks the decorated callable as an event handler for the given type of event, or as a trigger handler for the given trigger.

Note: For all events that are tied to Bones core, the event identifier is the class definition of an event.

Warning: You are free to use one callable for multiple events or multiple triggers, but it is not supported to use the same callable for both event handling and trigger handling.

Parameters

- **event** (*object*) – the identifier for the event you are going to handle
- **trigger** (*str*) – the trigger command to react to

2.3.3 Base Events

Base events are used as building blocks to add attributes and functions to other events. The event *bones.event.ChannelMessageEvent* for example inherits from both `bones.event.EventWithSource` and `bones.event.EventWithTarget`.

class `bones.event.`**Event**

client
A *bones.bot.BonesBot* instance representing the server connection which received this event.

class `bones.event.`**IrcPrivmsgEvent**(*client*, *user*, *channel*, *message*)
Bases: *bones.event.Event*

Event fired when the bot receives a message from another user, either over a query or from a channel.

Parameters

- **client** (*bones.bot.BonesBot*) – A *BonesBot* instance representing the current server connection.
- **user** (*bones.event.User*) – The hostmask of the user who sent this message.
- **channel** (*bones.event.Target*) – A *Target* instance representing the communication target this message was sent to.
- **message** (*string*) – The message that was sent to the target.

client
A *BonesBot* instance representing the server connection which received this event.

message
The message string that was sent.

channel
A `Target` instance representing the communication channel this message was sent to. This may be an object something that inherits *~bones.bot.Target*, like *~bones.bot.Channel* and *~bones.bot.User*.

user
A `User` instance representing the user that sent the message.

2.3.4 Events

All events inherits the attributes of its parents in addition to its own attributes. The class *UserJoinEvent* for example inherits the attribute `client` from the class *Event* even though that attribute may not be mentioned in the *UserJoinEvent* documentation.

class `bones.event.`**BotInitializedEvent**(*factory*)
Bases: *bones.event.Event*

An event that is fired whenever the bot factory for a configuartion has been initialized and is ready to connect to the server.

factory
The *BonesBotFactory* instance which was initialized.

class bones.event.**BotJoinEvent**(*client*, *channel*)
Bases: *bones.event.Event*

An event that is fired whenever the bot joins a new channel. The bot has "joined" a channel when the server informs the bot of its presence in that channel, and not when the bot calls `client.join("#channel")`.

client
The client instance that this event applies to.

channel
A *Channel* instance representing the channel that were joined by the bot.

class bones.event.**BotKickedEvent**(*client*, *channel*, *kicker*, *message*)
Bases: *bones.event.Event*

An event that is fired whenever the bot gets kicked from a channel it is in.

client
The client instance that this event applies to.

channel
A *Channel* instance representing the channel that the bot was kicked form.

kicker
A *User* instance representing the user that kicked the bot.

message
The kick reason as specified by the kicker, as a string.

class bones.event.**BotModuleLoaded**(*module*)
Bases: *bones.event.Event*

Called by the *BonesBotFactory* when a module is loaded during initialization.

module
The module instance that were initialized and loaded.

class bones.event.**BotNickChangedEvent**(*client*, *nick*)
Bases: *bones.event.Event*

Called by the bot when its nickname has changed, either by the bot itself or something like services, nick collision or the like.

client
The client instance that this event applies to.

nick
The new nickname that the bot now goes by.

class bones.event.**BotNoticeReceivedEvent**(*client*, *user*, *channel*, *message*)
Bases: *bones.event.Event*

Fired whenever the bot receives a notice.

client
The client instance that this event applies to.

user
A string representing the nickname of the user that sent the notice.

channel
A string representing the name of the channel that the notice were sent to.

message
The message that was sendt as part of the notice.

class bones.event.**BotPreJoinEvent**(*client*, *channel*)
Bases: *bones.event.Event*

Called by the bot before the bot joins a channel. This may be used to prevent the bot from joining a channel.

client
The client instance that this event applies to.

channel
The channel that the bot is trying to join, in string format.

isCancelled
A boolean that tells the bot whether to stop this event chain and prevent the bot from joining a channel.

class bones.event.**PreNicknameInUseError**(*client*, *prefix*, *params*)
Bases: *bones.event.Event*

An event that is fired before the bot's username is changed because of collision. As the bot's default behaviour tells it to cycle through all of the nicks in the bot nickname list, this may be used to prevent floods when changing the nickname by hand. An example use of this event is available in the bones.modules.utilities.NickFix module that makes the bot try to recover its nick if it is in use.

client
The client instance that this event applies to.

isCancelled
A boolean that tells the bot whether to stop this event chain and prevent the bot from automatically changing its nick.

class bones.event.**BotSignedOnEvent**(*client*)
Bases: *bones.event.Event*

An event which is fired whenever the bot finishes connecting and registrating with the server.

Parameters **client** (*bones.bot.BonesBot*) – The bot instance that this event originates from.

client
The client instance that this event applies to.

class bones.event.**BounceEvent**(*client*, *info*)
Bases: *bones.event.Event*

class bones.event.**ChannelMessageEvent**(*client*, *user*, *channel*, *message*)
Bases: *bones.event.IrcPrivmsgEvent*

class bones.event.**ChannelTopicChangedEvent**(*client*, *user*, *channel*, *newTopic*)
Bases: *bones.event.Event*

class bones.event.**CTCPVersionEvent**(*client*, *user*)
Bases: *bones.event.Event*

Fired by the bot after a CTCP VERSION has been received. This event may be used to cancel the CTCP VERSION reply that is usually sent.

client
The client instance that this event applies to.

isCancelled
A boolean that tells the bot whether to stop this event chain and prevent the bot from joining a channel.

user
The user that sent the CTCP VERSION request, as a *bones.event.User* instance.

class bones.event.**CTCPPongEvent**(*client*, *user*, *secs*)
Bases: *bones.event.Event*

Fired by the bot after a CTCP PING reply has been received. This event may be used to get the result of a `client.ping` request.

client
The client instance that this event applies to.

secs
Time elapsed since the ping request started, in seconds as a float.

user
The user that sent the CTCP VERSION request, as a *bones.event.User* instance.

class bones.event.**IRCUnknownCommandEvent**(*client*, *prefix*, *command*, *params*)
Bases: *bones.event.Event*

Fired whenever the bot encouters an unknown numeric reply and/or command.

client
The client instance that this event applies to.

prefix
The sender of the unknown command.

command
The numeric or string representation of the unknown command.

params
All supplied parameters, as a list.

class bones.event.**ModeChangedEvent**(*client*, *user*, *channel*, *set*, *modes*, *args*)
Bases: *bones.event.Event*

class bones.event.**PrivmsgEvent**(*client*, *user*, *channel*, *msg*)
Bases: *bones.event.Event*

Event fired when the bot receives a message from another user, either over a query or from a channel.

Deprecated since version Use: *bones.event.IrcPrivmsgEvent* instead.

Parameters

- **client** (*bones.bot.BonesBot*) – A *BonesBot* instance representing the current server connection.
- **user** (*string*) – The hostmask of the user who sent this message.
- **channel** (*bones.event.Target*) – A *Target* instance representing the communication target this message was sent to.
- **msg** (*string*) – The message that was sent to the target.

client
A *BonesBot* instance representing the server connection which received this event.

msg
The message string that was sent.

channel
A `Target` instance representing the communication channel this message was sent to. This may be an object something that inherits *~bones.bot.Target*, like *~bones.bot.Channel* and *~bones.bot.User*.

user
A `User` instance representing the user that sent the message.

class `bones.event.`**ServerChannelCountEvent**(*client*, *channels*)
Bases: *bones.event.Event*

Fired when the bot receives the `RPL_LUSERCHANNELS` numeric.

client
A *BonesBot* instance that represents the client that received this numeric reply.

channels
The server channel count, as an integer.

class `bones.event.`**ServerCreatedEvent**(*client*, *when*)
Bases: *bones.event.Event*

Fired when the bot receives the `RPL_CREATED` numeric.

client
A *BonesBot* instance that represents the client that received this numeric reply.

when
The creation date, as a string.

class `bones.event.`**ServerClientInfoEvent**(*client*, *info*)
Bases: *bones.event.Event*

Fired when the bot receives the `RPL_LUSERCLIENT` numeric.

client
A *BonesBot* instance that represents the client that received this numeric reply.

info
The client info, as a string.

class `bones.event.`**ServerHostInfoEvent**(*client*, *info*)
Bases: *bones.event.Event*

Fired when the bot receives the `RPL_YOURHOST` numeric.

client
A *BonesBot* instance that represents the client that received this numeric reply.

info
The server info, as a string.

class `bones.event.`**ServerInfoEvent**(*client*, *servername*, *version*, *umodes*, *cmodes*)
Bases: *bones.event.Event*

Fired when the bot receives the `RPL_MYINFO` numeric.

client
A *BonesBot* instance that represents the client that received this numeric reply.

servername
The name of the current server, as a string.

version
The version info of the current server, as a string.

umodes
The supported user modes on the current server.

cmodes
The supported channel modes on the current server.

class bones.event.**ServerLocalInfoEvent**(*client*, *info*)
Bases: *bones.event.Event*

Fired when the bot receives the RPL_LUSERME numeric.

client
A *BonesBot* instance that represents the client that received this numeric reply.

info
The server info, as a string.

class bones.event.**ServerMOTDReceivedEvent**(*client*, *motd*)
Bases: *bones.event.Event*

class bones.event.**ServerOpCountEvent**(*client*, *ops*)
Bases: *bones.event.Event*

An event that is fired during server connection. This event details how many operators are connected to the server.

Parameters **client** (*bones.bot.BonesBot*) – The bot instance for the server this event originated from.

client
A *bones.bot.BonesBot* instance representing the server connection which received this event.

ops
The number of local operators connected to the server, as an integer.

class bones.event.**ServerSupportEvent**(*client*, *options*)
Bases: *bones.event.Event*

An event that is fired whenever the bot receives an ISUPPORT during connection. This should be used by your Bones module if you for example need WATCH for your module to work.

Parameters **client** (*bones.bot.BonesBot*) – The bot instance for the server this event originated from.

client
A *bones.bot.BonesBot* instance representing the server connection which received this event.

options
A list of all the different options the server supports, in strings with the format "KEY=VALUE".

class bones.event.**TriggerEvent**(*client*, *args=None*, *channel=None*, *user=None*, *msg=None*, *match=None*)
Bases: *bones.event.ChannelMessageEvent*

An event that is fired by the bot whenever it receives a PRIVMSG event that starts with a valid trigger prefix and is a valid command.

Parameters

- **client** (*bones.bot.BonesBot*) – The bot instance which the event originated from.
- **args** (*list*) – A list of all the arguments provided with the trigger command.
- **user** (bones.bot.User) – The user which initiated this event.
- **msg** (*str.*) – The original message that was parsed to reveal the trigger command.

- **match** (SRE_Match) – The Regular Expression match object containing additional information about the parsing of the trigger command.

See also:

Event, *ChannelMessageEvent*

args
A list of strings containing all the arguments passed to the trigger command.

match
The regex match object that was returned while parsing the original message, msg

class bones.event.**UserActionEvent**(*client*, *user*, *channel*, *data*)
Bases: *bones.event.Event*

An event that is fired whenever another user sends a CTCP ACTION to the channel.

Parameters

- **client** (*bones.bot.BonesBot*) – The bot instance where this event occured.
- **user** (*bones.event.User*) – The user that initiated this event.
- **channel** (*str.*) – The channel the CTCP ACTION was sent to.
- **data** (*str.*) – The text which is actioned.

channel
A string representation of the channel the *User* joined.

client
The *bones.bot.BonesBot* instance representing the connection to the server that this event originated from.

data
A string representing the text which was actioned. Using the following example,

```
* Nickname slaps Operator around a bit with a large trout
```

the actioned text is everything following the nickname and the space immediately following i, or in other words slaps Operator around a bit with a large trout.

user
A *User* instance representing the user who initiated this event.

class bones.event.**UserJoinEvent**(*client*, *channel*, *user*)
Bases: *bones.event.Event*

An event that is fired whenever another user joins one of the channels the bot is in.

Parameters

- **client** (*bones.bot.BonesBot*) – The bot instance where this event occured.
- **channel** (*str.*) – The channel where this event occured.
- **user** (*User*) – The user who fired this event.

channel
A string representation of the channel the *User* joined.

client
The *bones.bot.BonesBot* instance representing the connection to the server that this event originated from.

user
A *User* instance representing the user who joined.

class bones.event.**UserKickedEvent**(*client*, *kickee*, *channel*, *kicker*, *message*)
Bases: *bones.event.Event*

An event that is fired whenever a user have been kicked from a channel.

Parameters

- **client** (*bones.bot.BonesBot*) – The bot instance where this event occured.
- **channel** (*bones.event.Channel*) – The channel instance where this event occured.
- **kickee** (*User*) – The nickname of the user who was kicked.
- **kicker** (*User*) – The nickname of the user who kicked the kickee.
- **message** (*str.*) – The message provided with the kick, usually as a reason for the kick.

channel
A *bones.event.Channel* instance representing the channel where the kick occurred.

client
A *bones.bot.BonesBot* instance representing the server from which the event originated.

kickee
A *User* instance representing the nickname of the user who was kicked by the kicker.

kicker
A *User* instance representing the nickname of the user who kicked the kickee. This is the user who initiated this event.

message
A string representing the message the kicker sent with the kick. This is often used as a reason for explaining the kick.

class bones.event.**UserMessageEvent**(*client*, *user*, *channel*, *message*)
Bases: *bones.event.IrcPrivmsgEvent*

class bones.event.**UserNickChangedEvent**(*client*, *user*, *oldname*, *newname*)
Bases: *bones.event.Event*

An event which is fired whenever a user in one of the joined channels changes his/her nickname.

Parameters

- **client** – The bot instance that this event originated from.
- **user** – The user who changed his/her nickname.

type user: bones.event.user :param oldname: The previous nickname the user went by. :type oldname: str :param newname: The new nickname the user is now using. :type newname: str

client
The *bones.bot.BonesBot* instance representing the connection to the server that this event originated from.

user
A *User* instance representing the user who changed his/her nickname.

newname
A string representation of the new nickname the user is using on the server.

oldname
A string representation of the nickname the user went by on the server before the nickname change.

class bones.event.**UserPartEvent**(*client*, *user*, *channel*)
Bases: *bones.event.Event*

An event that is fired whenever a user leaves a channel. This should not be confused with a *bones.event.UserQuitEvent*, which is sent only when a user quits from the server.

A *UserPartEvent* is sent once to each channel the user parts from. The user may still be connected to IRC, but the user is not available in the channel designated by the *channel* attribute.

Parameters

- **client** (*bones.bot.BonesBot*) – The bot instance that this event originated from.
- **user** (*bones.event.User*) – The the user who parted from the channel.
- **channel** (*str.*) – The name of the channel the user parted from.

channel
A string representation of the name of the channel the user parted from.

client
The *bones.bot.BonesBot* instance representing the connection to the server that this event originated from.

user
A *User* instance representing the nickname of the user who parted from the channel.

class bones.event.**UserQuitEvent**(*client*, *user*, *quitMessage*)
Bases: *bones.event.Event*

An event that is fired whenever a user quits IRC, or in other words leaves the server. This may be either because of a ping timeout, server/IRC operator KILL or the user sending QUIT to the server. This event should not be confused with a *bones.event.UserPartEvent*, which is sent once only when a user leaves a channel the bot is a part of.

A *UserQuitEvent* is sent once to the bot when the user quits from IRC, and does not mention what channels the user left while doing so. As such, when a *UserQuitEvent* is sent all plugins should usually treat this as the user parted from all channels the users where in.

The bot will not receive a *UserQuitEvent* if the user has only been involved with the bot through a query, and not been in any of the channels the bot is in. The bot will also not receive a *UserQuitEvent* if the user left all the channels the bot was in, and then QUIT from IRC afterwards.

Parameters

- **client** (*bones.bot.BonesBot*) – The bot instance where this event occured.
- **user** (*bones.event.User*) – The user who quit IRC.
- **quitMessage** (*str.*) – The message sent with the quit command.

client
The *bones.bot.BonesBot* instance representing the connection to the server where this event occured.

user
A *User* instance representing the nickname of the user who quit IRC.

quitMessage
A string representing the message that was sent with the quit. This message is usually formated by the server so that user-specified quit messages doesn't look like i.e. a ping timeout or an operator KILL.

2.3.5 Utility Classes

class bones.event.**Channel**(*name*, *server*)
Bases: *bones.event.Target*

Utility class representing a channel on a server.

modes
A dictionary of mode-value pairs representing the modes in the channel. Modes such as +b will be added to and removed from this list when the bot sees them.

users
A list of user instances representing all the users in the channel.

topic
An Topic instance containing the current topic and the user that wrote it.

kick(*user*, *reason=None*)
Kick a user from the channel.

Parameters

- **user** (*User*) – The user that should be kicked.
- **reason** (*str*) – A message that will be shown to users in the channel when kicking user.

part(*reason=None*)
Makes the bot part the channel.

Parameters **reason** (*str*) – The part message that will be sent to the channel when parting the channel.

setTopic(*topic*)
Changes the channel's topic.

Parameters **topic** (*str*) – The topic that should be changed to.

class bones.event.**Target**(*name*, *server*)
Utility class providing easy access to methods commonly used against targets.

Parameters

- **name** (*string*) – a string identifying the message target, as used in protocol message MSG targetNameHere :Message to be sent
- **server** (*bones.bot.BonesBot*) – the *BonesBot* client instance that will be used to send messages to this target.

name
String with the target name. This could for example be a nick, a hostname or a channel name.

server
BonesBot instance that will be used to send the messages to the target.

msg(*msg*)
Sends the provided message to the represented target.

Parameters **msg** (*string*) – message to be sent.

notice(*msg*)
Sends the provided message as a notice to the represented target.

Parameters **msg** (*string*) – message to be sent as a notice

class bones.event.**User**(*mask*, *server*)

Bases: *bones.event.Target*

Utility class turning a hostmask into distinguishable nickname, user and hostname attributes.

Parameters

- **mask** (*str*) – The IRC hostmask to be parsed. Ex: Bones!bot@192.168.0.2
- **server** (*bones.bot.BonesBot*) – *BonesBot* instance representing the server connection where we can reach this user.

mask

A string of the hostmask that this object originated from.

nickname

A string of the nickname for the provided hostmask. Given the hostmask above, the nickname will be Bones.

hostname

A string of the hostname for the provided hostmask. Given the hostmask above, the hostname will be 192.168.0.2. If the provided hostmask is missing the hostname part, this will be None.

username

A string of the username for the provided hostmask. Given the hostmask above, the username will be bot. If the provided hostmask is missing the username part, this will be None.

kick(*channel*, *reason=None*)

Kicks the user from the specified channel.

reason

A string that will be supplied with the kick as a reason for the kick.

channel

The *Channel* instance that represents the channel the user is to be kicked from.

ping()

Sends the user a CTCP PING query.

CHAPTER 3

Indices and tables

- genindex
- glossary
- modindex
- search

b

A

B

C

T

U

V

W