Omnipresence Documentation

Release 2.4

Kevin Xiwei Zheng

February 01, 2015

Contents

Omnipresence is an IRC utility bot built on Twisted, originally developed for the #yackfest channel on EsperNet. Most of its functionality is provided through plugins for commands and channel-specific event handlers, with some other sugar thrown in for flavor.

CHAPTER 1

Contents

1.1 Installing and running Omnipresence

Omnipresence requires Python 2.7 or later. There is no Python 3 support at this time.

Installation can be performed with the provided setup script:

```
$ python setup.py install
```

The following dependencies should be automatically installed by the setup script:

- Twisted 14.0.0 or later
- pyOpenSSL and service_identity
- SQLObject 0.10 or later
- Beautiful Soup 4

You will need to install an additional package to provide support for the specific database engine you wish to use. For example, MySQL support is provided by the *mysql-python* package. Some plugins have their own dependencies; consult the sample configuration in *docs/omnipresence.sample.cfg* for more details.

Omnipresence is installed as a Twisted application plugin executable through twistd, which handles daemonizing and logging. The twistd command takes the location of the bot configuration file as its sole argument. For example, to run Omnipresence in the foreground and log to stdout, use:

```
$ twistd -no omnipresence omnipresence.cfg
```

The following command starts Omnipresence as a daemon, logging to the file *messages* and using the PID file *pid*:

```
$ twistd -l messages --pidfile pid omnipresence omnipresence.cfg
```

For full information on the available options, consult the twistd documentation.

1.2 Writing plugins

Omnipresence supports two different types of plugins:

- **Handler plugins** listen for and respond to general events.
- **Command plugins** are more specialized variants of handler plugins that only respond when a specific *keyword* is sent to the bot in a message. In IRC channels, a *command prefix* is also expected. Both of these are specified in the bot configuration.

Plugins are expected to be module-level variables in submodules of the package `omnipresence.plugins`. Twisted's plugin documentation has further details. In practice, this means that you will write a plugin class that implements the provided interfaces, and assign an instance of that class to a variable in your plugin module:

```
# omnipresence/plugins/example.py
from twisted.plugin import IPlugin
from omnipresence.iomnipresence import ICommand


class ExampleCommand(object):
    implements(IPlugin, ICommand)
    name = 'example'

    def execute(self, bot, prefix, reply_target, channel, args):
        # ... command performs its work ...
        bot.reply(reply_target, channel, text)


# Don't forget to do this at the end of your module file, or Omnipresence
# will not load your command plugin!
example = ExampleCommand()
```

1.2.1 Handler plugins

Handler plugins are expected to implement both `twisted.plugin.IPlugin` and `IHandler`.

interface `omnipresence.iomnipresence.`**`IHandler`**

A handler that responds to IRC events passed to it by the bot. There are no required methods for this interface, since handlers may implement only a subset of available events. Callbacks are generally the same as those defined in `omnipresence.IRCClient`, except that an instance of the bot protocol class is provided as the second argument (after *self*).

`registered``()`

An optional callback, fired when the plugin is initialized. At this point, an `omnipresence.IRCClientFactory` object has been assigned to the `self.factory` object attribute, which can be used to read configuration data (through `config`).

`name`

The name used to refer to this handler in the configuration file, among other places.

1.2.2 Command plugins

Handler plugins are expected to implement both `twisted.plugin.IPlugin` and `ICommand`.

interface `omnipresence.iomnipresence.`**`ICommand`**

A command that is invoked in response to specially-formatted IRC messages.

The docstring is used to provide documentation for the `help` command plugin, with `%s` standing in for the keyword assigned by the bot's configuration. Generally, command docstrings take the form of a brief usage note, with the following formatting:

- Text to be typed literally by the user, including the command keyword, is presented in boldface, through wrapping with the IRC format code `\x02`.
- Variable names are presented with an underline, through wrapping with the IRC format code `\x1F`.
- Optional arguments are surrounded by unformatted square brackets.

•Choices among mutually-exclusive arguments are separated with vertical pipes.

For example:

```
class SampleCommand(object):
    '''
    \x02%s\x02
    \x1Fa\x1F|\x1Fb\x1F|\x1Fc\x1F
    [\x1Foptional_argument\x1F] -
    Provides an example within documentation.
    '''
```

This would be presented to a typical IRC client as follows, assuming that the command keyword `sample` has been assigned:

sample *a* | *b* | *c* [*optional_argument*] - Provides an example within documentation.

registered()
See `IHandler.registered()`.

execute(*self*, *bot*, *prefix*, *reply_target*, *channel*, *args*)
Invoked when a command message is seen.

Parameters

- **bot** (`omnipresence.IRCClient`) – The current bot protocol instance.
- **prefix** (*str*) – The `nick@user!host` prefix of the user that invoked this command.
- **reply_target** (*str*) – The target of command output redirection suggested by the invoking user with `> target`, or *prefix* if no such redirection is specified. This is not necessarily a valid prefix or nickname, as it is given by the user!
- **channel** (*str*) – The channel on which the command was invoked. For private messages, this is generally the bot's own nickname.
- **args** (*str*) – The arguments passed with the command, including the keyword used to invoke the command (`"keyword arg1 arg2"`).

Generally, a command's `execute()` method should either call the bot's `reply()` method and (implicitly) return `None`; or create a Twisted `Deferred` object, add a callback that calls `reply()`, and return that `Deferred` (see *Using Deferreds*). An error handler will be automatically added that replies with the associated value of any exceptions that are not handled by the command itself.

Most command plugins shipped with Omnipresence send error or "no-result" replies to *prefix*, giving the invoking user a chance to correct any potential mistakes, while successful replies are sent to *reply_target*:

```
def execute(self, bot, prefix, reply_target, channel, args):
    results = do_something(args)

    if not results:
        bot.reply(prefix, channel, 'No results found.')
        return

    bot.reply(reply_target, channel, results[0])
```

name
The name used to refer to this command in the configuration file, among other places.

Web-based commands

Omnipresence provides a convenience class for commands that rely on making an HTTP request and parsing the response, `WebCommand`. See the class documentation for more details.

1.2.3 Using Deferreds

Command and handler plugins that need to perform actions that would otherwise block the main thread should use Twisted's deferred execution mechanism to make the blocking call asynchronously. Omnipresence automatically adds an errback to any `Deferred` objects returned by a command plugin's `execute()` method, so that any unhandled errors encountered during the command's execution can be reported to the user.

1.3 API reference

1.3.1 Core IRC client

The Omnipresence IRC utility bot.

class `omnipresence.`**`IRCClient`**(*factory*)

Omnipresence's core IRC client protocol class. Common parameters for callbacks and class methods include:

prefix A full `nick!user@host` mask.

channel* or *nick An IRC channel, for commands directed at an entire channel; or specific nickname, for commands directed at a single user. Despite their names, parameters with either name will usually take both channels and nicks as values. To distinguish between the two in a callback, use the Twisted constant `CHANNEL_PREFIXES`:

```
from twisted.words.protocols import irc

class Handler(object):
    # ...
    def callback(self, prefix, channel):
        if channel[0] in irc.CHANNEL_PREFIXES:
            # addressed to a channel
        else:
            # addressed to the bot specifically
```

`action`(*prefix*, *channel*, *data*)

Called when a `/me` action is performed in the given *channel*.

`channel_names` = **None**

A mapping of channels to the set of nicks present in each channel.

`connectionLost`(*reason*)

Called when the connection to the IRC server has been lost or disconnected.

`connectionMade`()

Called when a connection has been successfully made to the IRC server.

`factory` = **None**

The `IRCClientFactory` that created this client.

`heartbeatInterval` = **60**

The number of seconds to wait between sending successive PINGs to the server. This overrides a class variable in Twisted's implementation, hence the unusual capitalization.

irc_ERR_NICKNAMEINUSE(*prefix*, *params*)
Called when the bot attempts to use a nickname that is already taken by another user.

join(*channel*)
Join the given *channel*. If joins have been suspended with `suspend_joins()`, add the channel to the join queue and actually join it when `resume_joins()` is called.

joined(*prefix*, *channel*)
Called when the bot successfully joins the given *channel*. Use this to perform channel-specific initialization.

kick(*channel*, *nick*, *reason=None*)
Kick the the given *nick* from the given *channel*.

kickedFrom(*channel*, *kicker*, *message*)
Called when the bot is kicked from the given *channel*.

last_pong = None
The time of the last PONG seen from the server.

leave(*channel*, *reason=None*)
Leave the given *channel*.

left(*prefix*, *channel*)
Called when the bot leaves the given *channel*.

max_lag = 150
The maximum acceptable lag, in seconds. If this amount of time elapses following a PING from the client with no PONG response from the server, the connection has timed out. (The timeout check only occurs at every `heartbeatInterval`, so actual disconnection intervals may vary by up to one heartbeat.)

me(*channel*, *action*)
Perform an action in the given *channel*.

message_buffers = None
A mapping of channels to a mapping containing message buffers for each channel, keyed by nick.

mode(*chan*, *set*, *modes*, *limit=None*, *user=None*, *mask=None*)
Change the mode of the given *channel*. See the Twisted documentation for information on this method's parameters.

modeChanged(*prefix*, *channel*, *set*, *modes*, *args*)
Called when a channel's mode is changed. See the Twisted documentation for information on this method's parameters.

msg(*nick*, *message*)
Send a message to the nickname or channel specified by *nick*.

myInfo(*servername*, *version*, *umodes*, *cmodes*)
Called with information about the IRC server at logon.

names(**channels*)
Ask the IRC server for a list of nicknames in the given channels. Plugins generally should not need to call this method, as it is automatically invoked on join.

nickChanged(*nick*)
Called when the bot's nickname is changed.

notice(*nick*, *message*)
Send a notice to the nickname or channel specified by *nick*.

noticed(*prefix*, *channel*, *message*)
Called when we receive a notice from another user. Behaves largely the same as `privmsg()`.

privmsg(*prefix*, *channel*, *message*)
Called when we receive a message from another user.

quit(*message=''*)
Quit from the IRC server.

reactor = None
The reactor in use on this client. This may be overridden when a deterministic clock is needed, such as in unit tests.

reply(*prefix*, *channel*, *message*)
Send a reply to a user. The method used depends on the values of *prefix* and *channel*:

- If *prefix* is specified and *channel* starts with an IRC channel prefix (such as # or +), send the reply publicly to the given channel, addressed to the nickname specified by *prefix*.
- If *prefix* is specified and *channel* is the bot's nickname, send the reply as a private notice to the nickname specified by *prefix*.
- If *prefix* is not specified, send the reply publicly to the channel given by *channel*, with no nickname addressing.

Long replies are buffered in order to satisfy protocol message length limits; a maximum of 256 characters will be sent at any given time. Further content from a buffered reply can be retrieved by using the command provided with the *more* plugin.

When possible, Omnipresence attempts to truncate replies on whitespace, instead of in the middle of a word. Replies are _always_ broken on newlines, which can be useful for creating commands that progressively give more information.

reply_with_error(*failure*, *prefix*, *channel*, *keyword*)
Call `reply()` with information on an error that occurred during an invocation of the command with the given *keyword*. *failure* should be an instance of `twisted.python.failure.Failure`.

Note: This method is automatically called whenever an unhandled exception occurs in a command's `execute()` method, and usually does not need to be invoked manually.

---

resume_joins()
Resume immediate joining of channels after suspending it with `suspend_joins()`, and perform any channel joins that have been queued in the interim.

setNick(*nickname*)
Change the bot's nickname.

signedOn()
Called after successfully signing on to the server.

signon_timed_out()
Called when a timeout occurs after connecting to the server, but before receiving the `RPL_WELCOME` message that starts the normal PING heartbeat.

signon_timeout = None
An `IDelayedCall` used to detect timeouts that occur after connecting to the server, but before receiving the `RPL_WELCOME` message that starts the normal PING heartbeat.

suspend_joins()
Suspend all channel joins until `resume_joins()` is called.

suspended_joins = None
If joins are suspended, a set containing the channels to join when joins are resumed. Otherwise, `None`.

topic(*channel*, *topic=None*)
Change the topic of *channel* if a *topic* is provided; otherwise, ask the IRC server for the current channel topic, which will be provided through the `topicUpdated()` callback.

topicUpdated(*nick*, *channel*, *newTopic*)
Called when the topic of the given *channel* is changed.

userJoined(*prefix*, *channel*)
Called when another user joins the given *channel*.

userKicked(*kickee*, *channel*, *kicker*, *message*)
Called when another user kicks a third party from the given *channel*.

userLeft(*prefix*, *channel*)
Called when another user leaves the given *channel*.

userQuit(*prefix*, *quitMessage*)
Called when another user has quit the IRC server.

userRenamed(*oldname*, *newname*)
Called when another user changes nick.

1.3.2 Web utility methods

Utility methods for retrieving and manipulating data from Web resources.

class `omnipresence.web.`**WebCommand**
A utility class for writing command plugins that make a single HTTP GET request and do something with the response.

Subclasses should define a `url` property containing the string `%s`, and implement the `reply()` method. When the command is invoked, `%s` is substituted with the command's literal argument string, and a deferred request to the resulting URL is made with `reply()` as its success callback.

An optional property `arg_type` can be used to indicate the type of argument that your custom command expects. This is used to provide a usage message should no arguments be given; for example, setting `arg_type` to `'a search term'` sets the usage message to "Please specify a search term." The default value is `'an argument string'`.

reply(*response*, *bot*, *prefix*, *reply_target*, *channel*, *args*)
Implement this method in your command subclass. The *response* argument will contain a `(headers, content)` response tuple as returned by `request()`. The other arguments are as passed in to `ICommand.execute()`.

`omnipresence.web.`**decode_html_entities**(*s*)
Convert HTML entities in a string to their Unicode character equivalents. This method is equivalent to:

```
textify_html(s, format_output=False)
```

Deprecated since version 2.2: Use `textify_html()` instead.

`omnipresence.web.`**request**(**args*, ***kwargs*)
Make an HTTP request, and return a Deferred that will yield an httplib2-style `(headers, content)` tuple to its callback.

Arguments are as for a request to a typical Twisted Web agent, with the addition of one keyword argument, *max_bytes*, that specifies the maximum number of bytes to fetch from the desired resource. If no `User-Agent` header is specified, one is added before making the request.

Two custom headers are returned in the response, in addition to any set by the HTTP server: `X-Omni-Location` contains the final location of the request resource after following all redirects, and

`X-Omni-Length` contains the original value of the response's `Content-Length` header, which Twisted may overwrite if the actual response exceeds *max_bytes* in size.

`omnipresence.web.`**`textify_html`**(*html*, *format_output=True*)
Convert the contents of *html* to a Unicode string. *html* can be either a string containing HTML markup, or a Beautiful Soup tag object. If *format_output* is `True`, IRC formatting codes are added to simulate common element styles.

1.3.3 Other helper methods

General utility functions used within Omnipresence.

`omnipresence.util.`**`ago`**(*then*, *now=None*)
Given a datetime object, return a string giving an approximate relative time, such as "5 days ago".

`omnipresence.util.`**`andify`**(*seq*, *two_comma=False*)
Given a list, join its elements and return a string of the form "*x* and *y*" for a two-element list, or "*x*, *y*, and *z*" for three or more elements. If *two_comma* is True, insert a comma before "and" even if the list is only two elements long ("*x*, and *y*").

`omnipresence.util.`**`duration_to_timedelta`**(*duration*)
Convert a duration of the form "?w?d?h?m?s" into a `datetime.timedelta` object, where individual components are optional.

`omnipresence.util.`**`readable_duration`**(*duration*)
Convert a duration of the form "?w?d?h?m?s" to a readable string representation of the form "2 weeks, 5 days, and 20 hours".

`omnipresence.util.`**`truncate_unicode`**(*s*, *char_limit*, *byte_limit*, *encoding='utf-8'*)
Truncate a Unicode string so that it fits both within the specified character limit and, when encoded in the given encoding, the specified byte limit. Return the truncated string as a byte string.

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

O

S

T

U

W