Omnipresence Documentation

Release 3.0a2

Kevin Xiwei Zheng

December 11, 2016

1 Installation **3**

2 Configuration **5**
2.1 Essentials . . . 5
2.2 Connections . . . 6
2.3 Channels . . . 6
2.4 Plugins . . . 6
2.5 Variables . . . 7
2.6 Ignore rules . . . 7
2.7 Data blocks . . . 8
2.8 Reloading . . . 8

3 Built-in plugins **9**
3.1 `.help` . . . 9
3.2 `.more` . . . 9
3.3 `.anidb` . . . 10
3.4 `.autorejoin` . . . 10
3.5 `.autovoice` . . . 10
3.6 `.dice` . . . 10
3.7 `.geonames` . . . 11
3.8 `.google` . . . 12
3.9 `.mstranslate` . . . 12
3.10 `.url` . . . 12
3.11 `.vndb` . . . 13
3.12 `.wwwjdic` . . . 13

4 Writing event plugins **15**
4.1 Command replies . . . 16
4.2 Configuration options . . . 17
4.3 Command base classes . . . 18
4.4 Writing tests . . . 18

5 API reference **19**
5.1 Connections . . . 19
5.2 Messages . . . 20
5.3 Hostmasks . . . 22
5.4 Case mappings . . . 23
5.5 Web resource interactions . . . 24

5.6 Human-readable output helpers 24

Python Module Index **25**

Omnipresence is an IRC utility bot built on Twisted, originally developed for EsperNet's #yackfest channel. It comes with a bucketload of plugins, flexible configuration, and an easy-to-use API for writing your own extensions.

CHAPTER 1

Installation

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

We recommend installing Omnipresence through PyPI:

```
$ pip install omnipresence[html]
```

If you have a source distribution:

```
$ pip install -e .[html]
```

Some built-in plugins have additional dependencies, which are listed in their documentation and in requirements files inside the plugin source directory.

Omnipresence is installed as an application plugin for twistd, which handles daemonizing and logging. The twistd command takes the location of the bot settings file as its sole argument. To run Omnipresence in the foreground and log to stdout, run:

```
$ twistd -no omnipresence settings.yaml
```

To start Omnipresence as a daemon, writing its process ID to *pid* and logging to the file *messages*, run:

```
$ twistd -l messages --pidfile pid omnipresence settings.yaml
```

For full information on available options, consult the twistd man page.

CHAPTER 2

Configuration

Omnipresence reads its configuration from a YAML file passed in on the twistd command line:

```
$ twistd -no omnipresence settings.yaml
```

This page details the syntax and basic directives of configuration files. For information on the settings provided by Omnipresence's built-in event plugins, see Built-in plugins.

See also:

The Wikipedia article on YAML gives a rundown of the basic structure and syntax of a YAML file.

2.1 Essentials

A directive is made up of a command and associated value, which are represented in the YAML format as a mapping key and value, respectively. Each command is a string made up of an initial keyword and zero or more arguments, which are parsed in a shell-like fashion (see `shlex.split`), while the value's type depends on the directive.

The directives for *plugins*, *variables*, and *ignore rules* cascade down into individual channels unless explicitly overridden. For example, given the configuration file:

```
set foo: 12345
channel redstapler:
    set foo: 67890
```

the variable `foo` is assumed to have the value `12345` everywhere except for the #redstapler channel, where it is `67890`.

The particular order of directives at the same nesting level does not matter; in other words, there is no concept of "earlier" or "later" in a file, only "shallower" and "deeper." This means that the following configuration is equivalent to the last one:

```
channel redstapler:
    set foo: 67890
set foo: 12345
```

Avoid giving the same directive more than once inside a block. The configuration parser may choose to use the value of either one depending on the whims of the YAML parser.

2.2 Connections

The following directives are only valid at the root of a configuration file, and specify the details of the connection to the IRC server:

- `host` is the hostname of the server. This directive is mandatory.
- `port` is the port to connect to on the server. It defaults to 6667.
- `ssl` determines whether to use SSL. It defaults to `False`.
- `nickname` is the bot's initial nickname. It defaults to `"Omnipresence"`.
- `username` is the username to use in the bot's hostmask if one is not provided by identd. It defaults to the value of `nickname`.
- `realname` is the bot's "real name," visible in WHOIS. It has no default value.
- `password` is the server password to use. It has no default value.

2.3 Channels

The `channel` and `private` directives give settings specific to a channel or direct messages for the bot, respectively:

```
private:
    plugin .nickserv: on
channel foo:
    plugin foo.specific: [foo]
channel bar:
    enabled: off
```

Like the connection control directives, these directives are only valid at the root of a configuration file.

The `channel` directive takes the name of a channel as its sole argument. The `#` prefix is optional and is automatically added if no other known channel prefix is present. As `#` is also used to indicate comments in YAML, the directive must be quoted if it is given:

```
"channel #foo":
    plugin foo.specific: [foo]
```

Needless to say, leaving it off is generally easier.

Inside a `channel` block, the value of the `enabled` directive controls Omnipresence's automatic join and part behavior. If it is true, the default for all explicitly configured channels, the channel is automatically joined on bot start and configuration reload. If false, the channel is not joined on bot start, and is parted from on reload if the bot is present there. If set to the string `"soft"`, the default for all channels not explicitly mentioned in the configuration, the channel is not joined on bot start, but is not parted from on reload.

2.4 Plugins

The `plugin` directive enables or disables a plugin in the current block and all blocks below it, unless overridden:

```
plugin .rss: on
plugin .wikipedia: [w, wp]
plugin .wikipedia/Random: [wr]
plugin foo.custom: [foo]
```

It takes the plugin's configuration name as its sole argument. Names that begin with a period (`.`) refer to built-in plugins, while others are custom plugins provided by third-party packages. If a package provides multiple plugins, alternatives are available by adding a slash and a second name (`/Random`).

The value is either a list of command keywords to use for plugins that provide a command, or Boolean `True` or `False`. `False` disables the plugin.

2.5 Variables

The `set` directive sets the value of a configuration variable:

```
set nickname: Omnipresence
set google.key: 0123456789abcdef
```

It takes the name of the variable to set as its sole argument. By convention, names not containing a period (`.`) are used for Omnipresence core settings, while those with a period belong to plugins. The value depends on the specific variable being set. Note that Omnipresence does not parse directives inside variable blocks, so the following configuration syntax is valid:

```
set deliberately.unused.variable:
    channel example: hello world
```

(You should use *data blocks* instead of abusing variable blocks to store arbitrary data for later reuse, however.)

To unset a variable, set it to `None` using a tilde character (~):

```
set rss.feeds: ~
```

The following variables affect Omnipresence's behavior:

- `command_prefixes` is a list of prefixes Omnipresence searches for in public channels to indicate a command. It has no default value.
- `direct_addressing` allows the bot's configured or current nickname, followed by a colon or a comma, to be a command prefix. It defaults to `True`.
- `reply_format` is a format string used for replies to public channels. The strings `{target}` and `{message}` are replaced by the target nickname and content of the reply, respectively. The default is `"\x0314{target}:  {message}"`, which colors the response text gray.
- `encoding` is the name of a Python character encoding used to encode and decode messages. The default is `"utf-8"`.

2.6 Ignore rules

The `ignore` directive tells Omnipresence to not pass messages from certain user hostmasks to certain plugins:

```
ignore no_google_for_you:
    hostmasks: ["*!*@foo.example"]
    include: [google]
ignore otherbots:
    hostmasks: [foobot, barbot]
    exclude: [.chanlog]
```

It takes an arbitrary name as its sole argument. This name can be used in nested blocks to disable the ignore rule:

```
channel mercy:
    ignore no_google_for_you: off
```

The value is either Boolean `False`, or a mapping containing a `hostmasks` directive and at most one of `include` or `exclude`. The value of `hostmasks` is a list of hostmasks the ignore rule applies to. If `include` is given, its value is used as an exhaustive list of plugins that should not respond to events from the given hostmasks. Otherwise, all plugins except those given in `exclude`, if present, ignore those hostmasks. If more than one ignore rule applies to a particular user, any rules with `exclude` take precedence over those with `include`; in either case, all values for each are combined.

2.7 Data blocks

The `data` directive opens a block that can store arbitrary data. Its contents are not parsed at all:

```
data:
    ignore totalanarchy:
        channel thismakesnosense: hello world
    but_here_are_some_defaults: &defaults
        plugin .help: [h, help]
        plugin .more: [m, more]
```

This feature allows the use of YAML references to define repeated configuration templates where they will explicitly not be parsed. For example, the `defaults` value from the data block above can now be used for specific channel settings:

```
channel bar:
    <<: *defaults
channel baz:
    <<: *defaults
    plugin baz.plugin: [quux]
```

2.8 Reloading

To reload the bot configuration, send a SIGUSR1 to the running process. Omnipresence will join and part channels according to *the channel configuration*. Changes to *connection directives* are ignored; they require a full restart of the bot.

CHAPTER 3

Built-in plugins

The following plugins are included in the Omnipresence distribution.

Some plugins have additional dependencies beyond what the Omnipresence setup script automatically installs for you. They are listed in the documentation below, and each such plugin in the Omnipresence source distribution also has a `requirements.txt` file in its directory that can be used with pip:

```
$ pip install -r omnipresence/plugins/url/requirements.txt
```

3.1 `.help`

Event plugins for providing command help.

class `omnipresence.plugins.help.`**`Default`**

Show detailed help for other commands, or list all available commands if no argument is given.

Alice help

Bot Available commands: **alpha**, **beta**, **help**. For further help, use **help** *keyword*. To redirect a command reply to another user, use *command* **>** *nick*.

Alice help alpha

Bot **alpha** *argument* - Help for command **alpha**.

3.2 `.more`

Event plugins for reading command reply buffers.

class `omnipresence.plugins.more.`**`Default`**

Show additional text from a user's reply buffer.

Brian count

Bot 1

Brian more

Bot 2

Brian more

Bot 3

In public channels, an optional argument can be passed to view the contents of another user's reply buffer.

Alice count

Bot alice: 1

Brian more alice

Bot brian: 2

3.3 `.anidb`

Event plugins for searching AniDB.

class `omnipresence.plugins.anidb.`**`Default`**

Look up an anime title on AniDB.

Alice anidb bakemonogatari

Bot http://anidb.net/a6327 — **Bakemonogatari** — TV Series, 12 episodes from 2009-07-03 to 2009-09-25 — rated 8.43 (7443)

3.4 `.autorejoin`

Event plugins for automatically rejoining channels after kicks.

class `omnipresence.plugins.autorejoin.`**`Default`**

Automatically rejoin channels after being kicked from them.

3.5 `.autovoice`

An event plugin that automatically voices users as they join a channel, unless that channel is moderated.

class `omnipresence.plugins.autovoice.`**`Default`**

Automatically voice users as they enter a channel, unless moderation is set with the `+m` channel mode. Useful for managing a sudden influx of new users.

Note that Omnipresence almost certainly has to have channel operator privileges (`+o`) in order for this plugin to work.

3.6 `.dice`

Event plugins for storing banks of die rolls used in role-playing games.

class `omnipresence.plugins.dice.`**`Default`**

Manage dice pools.

The `new`, `add`, `use`, and `clear` subcommands affect a per-user bank of die rolls, while `roll` is used for one-off rolls that should not be added to the bank.

Alice dice new 4d6

Bot Rolled **1 4 5 6** = 16. Bank now has **1 4 5 6** = 16.

Brian dice new 4d6

Bot Rolled **2 3 3 4** = 12. Bank now has **2 3 3 4** = 12.

Alice dice

Bot Bank has **1 4 5 6** = 16.

Alice dice show brian

Bot Bank has **2 3 3 4** = 12.

Brian dice roll 2d20

Bot Rolled **7 15** = 22.

Brian dice use 3 3

Bot Used **3 3** = 6. Bank now has **2 4** = 12.

Alice dice clear

Bot Bank cleared.

`banks` **= None**
User die banks, keyed by a (channel, nick) tuple.

`random` **= None**
The instance of `random.Random` used for die rolls. This is overridden for deterministic testing.

`roll_dice`(*dice*)
Return random rolls for the *dice* given as an iterable containing some combination of individual die groups as strings, such as `"2d6"`. Integers are accepted as dice; they "roll" to themselves.

3.7 `.geonames`

Event plugins for GeoNames services.

The `geonames.username` *settings variable* must be set to a valid GeoNames API username for them to function.

class `omnipresence.plugins.geonames.`**`Time`**
Look up the current time in a world location.

Brian time beijing

Bot Beijing, Beijing, China (39.91, 116.40): 2015-08-14 11:10

If pytz is installed, case-sensitive tz database names are also supported.

Alice time UTC

Bot UTC (tz database): 2015-08-14 03:10

class `omnipresence.plugins.geonames.`**`Weather`**
Look up weather conditions in a world location.

Brian weather london

Bot London, England, United Kingdom (51.51, -0.13): 19.0°C/66.2°F, broken clouds, 93% humidity from London City Airport (EGLC) as of 26 minutes ago

3.8 `.google`

Event plugins for Google searches.

class `omnipresence.plugins.google.`**`Default`**

Perform a Google Web search.

The `google.key` and `google.cx` *settings variables* must be set to valid Google Custom Search API credentials. For more information on setting up a Custom Search account, see the Stack Overflow topic "What are the alternatives now that the Google web search API has been deprecated?"

Alice google far-out son of lung

Bot https://www.youtube.com/watch?v=7g0sNbHWf9k — FSOL - Far Out Son Of Lung - YouTube: Aug 19, 2006 ... Far Out Son Of Lung And The Ramblings Of A Madman. (+886999 more)

3.9 `.mstranslate`

Event plugins for Microsoft Translator.

class `omnipresence.plugins.mstranslate.`**`Default`**

Translate text between languages with Microsoft Translator.

The `mstranslate.subscription_key` *settings variable* must be set to a valid Microsoft Cognitive Services subscription key. For more information, see the Text Translation API documentation.

The `mstranslate.default_target` settings variable is an optional two-letter language code specifying the target language to use if the user does not specify one. It defaults to `en` for English.

Alice mstranslate hola

Bot Hello

Alice mstranslate hola de:

Bot Hallo

3.10 `.url`

Event plugins for previewing the content of mentioned URLs.

class `omnipresence.plugins.url.`**`Default`**

Fetch the titles of URLs mentioned in normal messages or actions.

Requires Little Brother.

Charlie http://www.example.com/ is an example site

Bot [www.example.com] Example Domain

Alice http://www.example.org/ and http://www.example.net/ too

Bot [www.example.org] Example Domain

Bot [www.example.net] Example Domain

3.11 `.vndb`

Event plugins for searching the Visual Novel Database.

class `omnipresence.plugins.vndb.`**`Default`**

Look up a visual novel title on the Visual Novel Database.

Brian vndb ever17

Bot https://vndb.org/v17 — **Ever17 -The Out of Infinity-**, first release 2002-08-29 — rated 8.71 (3763) (+1 more)

3.12 `.wwwjdic`

Event plugins for searching WWWJDIC.

class `omnipresence.plugins.wwwjdic.`**`Default`**

Define a Japanese word or phrase using Jim Breen's WWWJDIC.

If Waapuro is installed, Nihon-shiki romanizations are provided alongside the kana spellings.

Alice wwwjdic kotoba

Bot (P);; [(kotoba) (P); (ketoba) ()(ok)] (n) (1) (See) language; dialect; (2) (See) word; words; phrase; term; expression; remark; (3) speech; (manner of) speaking; (P) (+28 more)

CHAPTER 4

Writing event plugins

Event plugins are Omnipresence's primary extension mechanism. This page details how to add new functionality by creating your own plugins. For information on the built-in plugins shipped with the Omnipresence distribution, see Built-in plugins.

class `omnipresence.plugin.`**`EventPlugin`**`(`*bot*`)`

A container for callbacks that Omnipresence fires when IRC messages are received.

Omnipresence locates event plugin classes by their Python module and class names, as detailed in Configuration. For convenience, a plugin key containing only a module name implies the class name `Default`. The following code in the top-level module `foo` therefore creates event plugins named `foo` (or `foo/Default`) and `foo/Other`:

```
from omnipresence.plugin import EventPlugin

class Default(EventPlugin):
    pass

class Other(EventPlugin):
    pass
```

When a message is received, Omnipresence looks for a plugin method named `on_` followed by the name of the *`MessageType`*, such as `on_privmsg`. If one exists, it is called with a *`Message`* object as the sole parameter. For example, the following plugin sends a private message to greet incoming channel users:

```
class Default(EventPlugin):
    def on_join(self, message):
        greeting = 'Hello, {}!'.format(message.actor.nick)
        message.connection.msg(message.venue, greeting)
```

Callbacks that need to execute blocking code can return a Twisted `Deferred` object:

```
class Default(EventPlugin):
    def on_privmsg(self, message):
        d = some_deferred_task()
        d.addCallback(lambda s: message.connection.msg(message.venue, s))
        return d
```

By default, callbacks are not fired for outgoing events generated by bot messages, in order to reduce the probability of accidental response loops. To change this behavior, set the `outgoing` attribute of a callback method to `True`. Inside the callback, each message's *`outgoing`* attribute can be used to determine its direction of transit:

```
from twisted.python import log

class Default(EventPlugin):
    def on_privmsg(self, message):
        direction = 'Outgoing' if message.outgoing else 'Incoming'
        log.msg('%s message: %r' % (direction, message))
    on_privmsg.outgoing = True
```

Note: Since most servers echo joins, parts, and quits back to clients, callbacks registered for these actions will always fire once on bot actions, twice if enabled for outgoing messages. You may wish to compare the message's *actor* and the connection's `nickname` attributes to distinguish bot actions in these cases:

```
class Default(EventPlugin):
    def on_join(self, message):
        if message.actor.matches(message.connection.nickname):
            the_bot_joined()
        else:
            someone_else_joined()
```

4.1 Command replies

Any plugin with an `on_command` callback can be assigned one or more keywords in *its configuration*. Unlike most other callbacks, whose return values are ignored, any value returned from `on_command` becomes the command reply, and is sent as either a channel message addressed to the command target or a private notice depending on how the command was invoked. A command reply may take one of the following forms:

- `None`, in which case no reply is shown to the target user, not even a "no results" message. This is useful for commands that have other visible side effects, such as changing the channel topic or mode.
- A byte or Unicode string. Long strings are broken into chunks of up to `CHUNK_LENGTH` bytes and treated as a sequence.
- A sequence of strings. Any reply strings containing more than *MAX_REPLY_LENGTH* bytes are truncated on display.
- An iterator yielding either strings or `Deferred` objects that yield strings. Long reply strings are truncated as with sequence replies.
- A `Deferred` object yielding any of the above.

Unless the reply is `None`, the first reply is immediately shown to the target user, and any remaining replies are placed in a buffer for later retrieval using the *.more* command. Newlines inside replies are displayed as a slash surrounded by spaces.

The following example plugin implements an infinite counter:

```
from itertools import count
from omnipresence.plugin import EventPlugin

class Default(EventPlugin):
    def on_command(self, msg):
        return count()
```

4.1.1 Error reporting

By default, if an error occurs inside an `on_command` callback, Omnipresence replies with a generic error message and logs the full traceback to the twistd log file. This behavior can be changed with the `show_errors` configuration option. If you wish to always show detailed information for an error, raise a *`UserVisibleError`*:

exception `omnipresence.plugin.`**`UserVisibleError`**(**args*)

Raise this inside a command callback if you need to return an error message to the user, regardless of whether or not the `show_errors` configuration option is enabled. Errors are always given as replies to the invoking user, even if command redirection is requested.

4.1.2 Help strings

To provide a help string for the *`.help`* command, return it from the `on_cmdhelp` callback. The incoming *`Message`*'s *`content`* attribute contains any additional arguments to *`.help`*, allowing help subtopics:

```
def on_cmdhelp(self, msg):
    if msg.content == 'detailed':
        return '\x02detailed\x02 - Show more information.'
    if msg.content == 'terse':
        return '\x02terse\x02 - Show less information.'
    return ('[\x02detailed\x02|\x02terse\x02] - Do some stuff. '
            'For more details, see help for \x02{}\x02 \x1Faction\x1F.'
            .format(msg.subaction))
```

Note that the command keyword is automatically prepended to the help string on display.

Omnipresence's built-in plugins provide help strings of the form `usage - Help text.`, where the usage string is formatted as follows:

- Strings to be typed literally by the user are bolded using `\x02`.
- Strings representing command arguments are underlined using `\x1F`.
- Optional components are surrounded by brackets (`[optional]`).
- Alternatives are separated by vertical bars (`this|that|other`).

4.2 Configuration options

Use the `Message.settings` wrapper to look up the value of a configuration variable:

```
def on_command(self, msg):
    return msg.settings.get('foo.bar', 'default value')
```

The value of a configuration variable may change while the bot is running (see *Reloading*). If a plugin needs to update its internal state on these changes, it can do so by defining a `configure` callback, which is passed the current bot settings:

```
def configure(self, settings):
    self.process(settings.get('foo.bar'))
```

By convention, plugin configuration variable names should share a common prefix ending with a period (`.`). Undotted names are reserved for Omnipresence core variables.

4.3 Command base classes

Omnipresence provides classes for common types of command plugins. As with the standard *EventPlugin* class, they are intended to be subclassed, not instantiated.

class `omnipresence.plugin.`**`SubcommandEventPlugin`**`(`*bot*`)`

A base class for command plugins that invoke subcommands given in the first argument by invoking one of the following methods:

1. `on_empty_subcommand(msg)`, if no arguments are present. The default implementation raises a *UserVisibleError* asking the user to provide a valid subcommand.
2. `on_subcommand_KEYWORD(msg, remaining_args)`, if such a method exists.
3. Otherwise, `on_invalid_subcommand(msg, keyword, remaining_args)`, which by default raises an "unrecognized command" *UserVisibleError*.

`on_cmdhelp` is similarly delegated to `on_subcmdhelp` methods:

1. `on_empty_subcmdhelp(msg)`, if no arguments are present. The default implementation lists all available subcommands.
2. `on_subcmdhelp_KEYWORD(msg)`, if such a method exists.
3. Otherwise, `on_invalid_subcmdhelp(msg, keyword)`, which by default simply calls `on_empty_subcmdhelp`.

As with `on_cmdhelp`, the subcommand keyword is automatically added to the help string, after the containing command's keyword and before the rest of the string.

4.4 Writing tests

...

CHAPTER 5

API reference

5.1 Connections

class `omnipresence.connection.`**`Connection`**

Omnipresence's core IRC client protocol.

See Twisted's IRCClient documentation for details on methods used to perform basic actions.

`reply`(*string*, *request*, *tail=''*)

Send a reply *string*, truncated to *`MAX_REPLY_LENGTH`* characters, with `tail` appended. If the request venue is a channel, send the reply to the venue as a standard message addressed to *request*'s *`target`*, formatted using the *`venue`*'s reply format. Otherwise, send the reply as a notice to *request*'s *`actor`*.

`is_channel`(*name*)

Return `True` if *name* belongs to a channel, according to the server-provided list of channel prefixes, or `False` otherwise.

`suspend_joins`()

Suspend all channel joins until *`resume_joins`* is called.

`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.

`case_mapping`

The *`CaseMapping`* currently in effect on this connection. Defaults to `rfc1459` if none is explicitly provided by the server.

`venues`

A mapping of venue names to *`VenueInfo`* objects.

`parser`

The *`RawMessageParser`* being used on this connection.

`reactor`

The reactor in use on this client. This may be overridden when a deterministic clock is needed, such as in unit tests.

class `omnipresence.connection.`**`VenueInfo`**(*case_mapping=None*)

A container for information about a venue.

`modes` **= None**

A mapping of modes currently active on this channel to one of `False` (not set or invalid), `True` (set, for modes that take no arguments), a single number or string, or a set of `Hostmask` objects.

nicks = None
A dictionary mapping nicks to *VenueUserInfo* objects.

topic = None
This channel's topic, or the empty string if none is set.

class omnipresence.connection.**VenueUserInfo**
A container for information about a user's state in a particular venue.

reply_buffer = None
This user's current channel reply buffer.

omnipresence.connection.**MAX_REPLY_LENGTH = 288**
The maximum length of a single command reply, in bytes.

5.2 Messages

class omnipresence.message.**Message**(*connection*, *outgoing*, *action*, *actor=None*, *venue=None*, *target=None*, *subaction=None*, *content=None*)
Represents a message, loosely defined as an event to which plugins can respond. Messages have the following basic attributes:

connection
The *Connection* on which the message was received.

outgoing
A boolean indicating whether this message resulted from a bot action.

action
This message's *MessageType*. A string containing a message type name may be passed to the constructor, but the property always contains an enumeration member.

actor
A *Hostmask* corresponding to the message prefix, indicating the message's true origin. In some cases, `unknown` messages will set this attribute to `None` if a prefix could not be parsed.

venue
target
subaction
content
Optional attributes, whose presence and meaning depend on the message type. An attribute is `None` if and only if it is not used by the current message type, and a string value otherwise.

raw
If this message was created by parsing a raw message with *RawMessageParser.parse*, the original raw IRC message string passed to that function. Otherwise, `None`.

Note: All string values are byte strings, not Unicode strings, and therefore must be appropriately decoded when necessary.

The following additional properties are derived from the values of one or more basic attributes, and are included for convenience:

private
`True` if this message has a venue and that venue is not a public channel. Otherwise, `False`.

`encoding`
The character encoding in effect for this message's venue.

New message objects can be created using either the standard constructor, or by parsing a raw IRC message string using *`RawMessageParser.parse`*.

`Message` is a `namedtuple` type, and thus its instances are immutable. To create a new object based on the attributes of an existing one, use an instance's `_replace` method.

class `omnipresence.message.`**`MessageType`**
An enumeration of valid values of *`Message.action`*.

The following message types directly correspond to incoming or outgoing IRC messages (also see **RFC 1459#section-4**):

`action` **= <MessageType.action: 1>**
Represents a CTCP ACTION (`/me`). All attributes are as for the *`privmsg`* type.

`ctcpquery` **= <MessageType.ctcpquery: 2>**
Represents an otherwise unrecognized CTCP query wrapped in a `PRIVMSG`. *`venue`* is the nick or channel name of the recipient. *`subaction`* is the CTCP message tag. *`content`* is a string containing any trailing arguments.

Note: Omnipresence does not support mixed messages containing both normal and CTCP extended content.

`ctcpreply` **= <MessageType.ctcpreply: 3>**
Represents an unrecognized CTCP reply wrapped in a `NOTICE`. All attributes are as for the *`ctcpquery`* type.

`join` **= <MessageType.join: 4>**
Represents a channel join. *`venue`* is the channel being joined.

`kick` **= <MessageType.kick: 5>**
Represents a kick. *`venue`* is the channel the kick took place in. *`target`* is the nick of the kicked user. *`content`* is the kick message.

`mode` **= <MessageType.mode: 6>**
Represents a mode change. *`venue`* is the affected channel or nick. *`content`* is the mode change string.

`nick` **= <MessageType.nick: 7>**
Represents a nick change. *`content`* is the new nick.

`notice` **= <MessageType.notice: 8>**
Represents a notice. All attributes are as for the *`privmsg`* type.

`part` **= <MessageType.part: 9>**
Represents a channel part. *`venue`* is the channel being departed from. *`content`* is the part message.

`privmsg` **= <MessageType.privmsg: 10>**
Represents a typical message. *`venue`* is the nick or channel name of the recipient. (*`private`* can also be used to determine whether a message was sent to a single user or a channel.) *`content`* is the text of the message.

`quit` **= <MessageType.quit: 11>**
Represents a client quit from the IRC network. *`content`* is the quit message.

`topic` **= <MessageType.topic: 12>**
Represents a topic change. *`venue`* is the affected channel. *`content`* is the new topic, or an empty string if the topic is being unset.

`unknown` = <MessageType.unknown: 0>
: Represents a message that is not of any known type, or that could not be correctly parsed. `subaction` is the IRC command name or numeric. `content` is a string containing any trailing arguments.

Omnipresence defines additional message types for synthetic events:

`connected` = <MessageType.connected: 9001>
: Created when the server has responded with `RPL_WELCOME`. No optional arguments are specified.

`disconnected` = <MessageType.disconnected: 9002>
: Created when the connection to the server has been closed or lost. No optional arguments are specified.

`command` = <MessageType.command: 9003>
: Represents a *command invocation*. `venue` is as for the `privmsg` type. `target` is a string containing the reply redirection target, or the actor's nick if none was specified. `subaction` is the command keyword. `content` is a string containing any trailing arguments.

`cmdhelp` = <MessageType.cmdhelp: 9004>
: Represents a command help request. `venue` and `target` are as for the `command` type. `subaction` is the command keyword for which help was requested. `content` is a string containing any trailing arguments.

class `omnipresence.message.parser.`**`RawMessageParser`**
: An implementation of the parsing rules for a specific version of the IRC protocol.

In most cases, you should use the `parser` attribute of a `Connection` to retrieve an instance of this class.

`parse`(*connection*, *outgoing*, *raw*, ***kwargs*)
: Parse a raw IRC message string and return a corresponding `Message` object. Any keyword arguments override field values returned by the parser.

5.2.1 Message formatting

Operations on mIRC-style message formatting.

`omnipresence.message.formatting.`**`remove_formatting`**(*string*)
: Return *string* with mIRC-style formatting control codes removed.

`omnipresence.message.formatting.`**`unclosed_formatting`**(*string*)
: Return a `frozenset` containing any mIRC-style formatting codes that remain in effect at the end of *string*.

5.3 Hostmasks

Operations on IRC hostmasks.

class `omnipresence.hostmask.`**`Hostmask`**
: Represents an IRC hostmask (sometimes called a message prefix) of the form `nick!user@host`. The *user* and *host* attributes are optional, and default to `None` if not present.

classmethod `from_string`(*string*)
: Return a new `Hostmask` object parsed from *string*, according to the definition of `<prefix>` in **RFC 1459#section-2.3.1**.

`has_wildcard`
: `True` if this hostmask contains a wildcard or any `None` components. Otherwise, `False` otherwise.

`matches`(*other*, *case_mapping=None*)
: Check whether this hostmask matches the pattern in *other*, which can be a `Hostmask` object or a string in

the form `"nick!user@host"`, according to the wildcard expansion rules in **RFC 2812#section-2.5**. A *`CaseMapping`* object may optionally be provided, in which case nicks are compared case-insensitively according to the mapping's rules; otherwise, nick comparisons are fully case-sensitive. Users and hosts are always compared case-insensitively, using normal ASCII case folding rules.

Briefly, `*` and `?` wildcards match zero or more and exactly one non-delimiter character, respectively; a backslash can be used to escape these special characters. Components that equal `None` are assumed to match all possible values for that component.

Warning: This method only matches wildcards in *other*, not the *`Hostmask`* object it is called on:

```
>>> Hostmask.from_string('nick!user@host').matches('nick!*@*')
True
>>> Hostmask.from_string('nick!*@*').matches('nick!user@host')
False
```

5.4 Case mappings

Operations on IRC case mappings.

class `omnipresence.case_mapping.`**`CaseMapping`**(*lower*, *upper*)

Provides convenience methods for bidirectional string translation given a mapping of characters from *lower* to *upper*.

classmethod **`by_name`**(*name*)

Return an IRC case mapping given the *name* used in the `CASEMAPPING` parameter of a `RPL_ISUPPORT` IRC message (numeric 005). The following mapping names are recognized:

`ascii`
Treats the letters *A-Z* as uppercase versions of the letters *a-z*.

`strict-rfc1459`
Extends the `ascii` case mapping to further treat *{}|* as the lowercase versions of *[]*. This matches the rules specified in **RFC 1459#section-2.2**.

`rfc1459`
Extends the `strict-rfc1459` case mapping to further treat ~ as the lowercase version of ^. This corresponds to most servers' actual implementation of the RFC 1459 rules.

`ValueError` is raised on an unrecognized mapping name.

`equates`(*one*, *two*)

Return a boolean value indicating whether *a* and *b* are equal under this case mapping.

`lower`(*string*)

Return a copy of *string* with uppercase characters converted to lowercase according to this case mapping.

`upper`(*string*)

Return a copy of *string* with lowercase characters converted to uppercase according to this case mapping.

class `omnipresence.case_mapping.`**`CaseMappedDict`**(*initial=None*, *case_mapping=None*)

A dictionary whose keys are treated case-insensitively according to a *`CaseMapping`* or mapping name string (as given to *`by_name`*) provided on instantiation.

5.5 Web resource interactions

`omnipresence.web.html.`**`textify`**`(`*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, mIRC-style formatting codes are added to simulate common element styles.

`omnipresence.web.http.`**`read_json_body`**`(`*response*`)`

Return a `Deferred` yielding a Python object deserialized from the Twisted Web *response* containing JSON data in its body.

5.6 Human-readable output helpers

Functions for presenting data in human-readable forms.

`omnipresence.humanize.`**`ago`**`(`*then*, *now=None*`)`

Given a `datetime` object, return an English string giving an approximate relative time, such as "5 days ago".

`omnipresence.humanize.`**`andify`**`(`*seq*, *two_comma=False*`)`

Join the elements of a sequence 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.humanize.`**`duration_to_timedelta`**`(`*duration*`)`

Convert a duration string of the form "_w_d_h_m_s" into a `timedelta` object.

`omnipresence.humanize.`**`readable_duration`**`(`*duration*`)`

Convert a duration string of the form "_w_d_h_m_s" to a plain English representation such as "2 weeks, 5 days, and 20 hours".

- List of modules
- genindex

o

omnipresence.case_mapping, 23
omnipresence.connection, 19
omnipresence.hostmask, 22
omnipresence.humanize, 24
omnipresence.message, 20
omnipresence.message.formatting, 22
omnipresence.message.parser, 22
omnipresence.plugin, 15
omnipresence.plugins, 9
omnipresence.plugins.anidb, 10
omnipresence.plugins.autorejoin, 10
omnipresence.plugins.autovoice, 10
omnipresence.plugins.dice, 10
omnipresence.plugins.geonames, 11
omnipresence.plugins.google, 12
omnipresence.plugins.help, 9
omnipresence.plugins.more, 9
omnipresence.plugins.mstranslate, 12
omnipresence.plugins.url, 12
omnipresence.plugins.vndb, 13
omnipresence.plugins.wwwjdic, 13
omnipresence.web, 24

A

action (omnipresence.message.Message attribute), 20
action (omnipresence.message.MessageType attribute), 21
actor (omnipresence.message.Message attribute), 20
ago() (in module omnipresence.humanize), 24
andify() (in module omnipresence.humanize), 24

B

banks (omnipresence.plugins.dice.Default attribute), 11
by_name() (omnipresence.case_mapping.CaseMapping class method), 23

C

case_mapping (omnipresence.connection.Connection attribute), 19
CaseMappedDict (class in omnipresence.case_mapping), 23
CaseMapping (class in omnipresence.case_mapping), 23
cmdhelp (omnipresence.message.MessageType attribute), 22
command (omnipresence.message.MessageType attribute), 22
connected (omnipresence.message.MessageType attribute), 22
Connection (class in omnipresence.connection), 19
connection (omnipresence.message.Message attribute), 20
content (omnipresence.message.Message attribute), 20
ctcpquery (omnipresence.message.MessageType attribute), 21
ctcpreply (omnipresence.message.MessageType attribute), 21

D

Default (class in omnipresence.plugins.anidb), 10
Default (class in omnipresence.plugins.autorejoin), 10
Default (class in omnipresence.plugins.autovoice), 10
Default (class in omnipresence.plugins.dice), 10
Default (class in omnipresence.plugins.google), 12
Default (class in omnipresence.plugins.help), 9
Default (class in omnipresence.plugins.more), 9
Default (class in omnipresence.plugins.mstranslate), 12
Default (class in omnipresence.plugins.url), 12
Default (class in omnipresence.plugins.vndb), 13
Default (class in omnipresence.plugins.wwwjdic), 13
disconnected (omnipresence.message.MessageType attribute), 22
duration_to_timedelta() (in module omnipresence.humanize), 24

E

encoding (omnipresence.message.Message attribute), 20
equates() (omnipresence.case_mapping.CaseMapping method), 23
EventPlugin (class in omnipresence.plugin), 15

F

from_string() (omnipresence.hostmask.Hostmask class method), 22

H

has_wildcard (omnipresence.hostmask.Hostmask attribute), 22
Hostmask (class in omnipresence.hostmask), 22

I

is_channel() (omnipresence.connection.Connection method), 19

J

join (omnipresence.message.MessageType attribute), 21

K

kick (omnipresence.message.MessageType attribute), 21

L

lower() (omnipresence.case_mapping.CaseMapping method), 23

M

matches() (omnipresence.hostmask.Hostmask method), 22
MAX_REPLY_LENGTH (in module omnipresence.connection), 20
Message (class in omnipresence.message), 20
MessageType (class in omnipresence.message), 21
mode (omnipresence.message.MessageType attribute), 21
modes (omnipresence.connection.VenueInfo attribute), 19

N

nick (omnipresence.message.MessageType attribute), 21
nicks (omnipresence.connection.VenueInfo attribute), 19
notice (omnipresence.message.MessageType attribute), 21

O

omnipresence.case_mapping (module), 23
omnipresence.connection (module), 19
omnipresence.hostmask (module), 22
omnipresence.humanize (module), 24
omnipresence.message (module), 20
omnipresence.message.formatting (module), 22
omnipresence.message.parser (module), 22
omnipresence.plugin (module), 15
omnipresence.plugins (module), 9
omnipresence.plugins.anidb (module), 10
omnipresence.plugins.autorejoin (module), 10
omnipresence.plugins.autovoice (module), 10
omnipresence.plugins.dice (module), 10
omnipresence.plugins.geonames (module), 11
omnipresence.plugins.google (module), 12
omnipresence.plugins.help (module), 9
omnipresence.plugins.more (module), 9
omnipresence.plugins.mstranslate (module), 12
omnipresence.plugins.url (module), 12
omnipresence.plugins.vndb (module), 13
omnipresence.plugins.wwwjdic (module), 13
omnipresence.web (module), 24
outgoing (omnipresence.message.Message attribute), 20

P

parse() (omnipresence.message.parser.RawMessageParser method), 22
parser (omnipresence.connection.Connection attribute), 19
part (omnipresence.message.MessageType attribute), 21
private (omnipresence.message.Message attribute), 20
privmsg (omnipresence.message.MessageType attribute), 21

Q

quit (omnipresence.message.MessageType attribute), 21

R

random (omnipresence.plugins.dice.Default attribute), 11
raw (omnipresence.message.Message attribute), 20
RawMessageParser (class in omnipresence.message.parser), 22
reactor (omnipresence.connection.Connection attribute), 19
read_json_body() (in module omnipresence.web.http), 24
readable_duration() (in module omnipresence.humanize), 24
remove_formatting() (in module omnipresence.message.formatting), 22
reply() (omnipresence.connection.Connection method), 19
reply_buffer (omnipresence.connection.VenueUserInfo attribute), 20
resume_joins() (omnipresence.connection.Connection method), 19
RFC
    RFC 1459#section-2.2, 23
    RFC 1459#section-2.3.1, 22
    RFC 1459#section-4, 21
    RFC 2812#section-2.5, 23
roll_dice() (omnipresence.plugins.dice.Default method), 11

S

subaction (omnipresence.message.Message attribute), 20
SubcommandEventPlugin (class in omnipresence.plugin), 18
suspend_joins() (omnipresence.connection.Connection method), 19

T

target (omnipresence.message.Message attribute), 20
textify() (in module omnipresence.web.html), 24
Time (class in omnipresence.plugins.geonames), 11
topic (omnipresence.connection.VenueInfo attribute), 20
topic (omnipresence.message.MessageType attribute), 21

U

unclosed_formatting() (in module omnipresence.message.formatting), 22
unknown (omnipresence.message.MessageType attribute), 21
upper() (omnipresence.case_mapping.CaseMapping method), 23
UserVisibleError, 17

V

venue (omnipresence.message.Message attribute), 20
VenueInfo (class in omnipresence.connection), 19
venues (omnipresence.connection.Connection attribute), 19

VenueUserInfo (class in omnipresence.connection), 20

W

Weather (class in omnipresence.plugins.geonames), 11