csbot Documentation

Release alpha

#cs-york

Feb 19, 2022

Contents

1 How to write plugins **3**
- 1.1 Anatomy of a plugin . . . 3
- 1.2 Events . . . 3
- 1.3 Commands . . . 4
- 1.4 Responding: the `BotProtocol` object . . . 4
- 1.5 Configuration . . . 5
- 1.6 Database . . . 5

2 Events **7**
- 2.1 Raw events . . . 7
- 2.2 Bot events . . . 7
- 2.3 Message events . . . 8
- 2.4 Channel events . . . 8
- 2.5 User events . . . 8

3 csbot **9**
- 3.1 csbot package . . . 9

4 Indices and tables **45**

Python Module Index **47**

Index **49**

Contents:

CHAPTER 1

How to write plugins

1.1 Anatomy of a plugin

Plugins are automatically discovered if they match the right pattern. They must

- subclass *csbot.plugin.Plugin*, and
- live under the package specified by `csbot.core.Bot.PLUGIN_PACKAGE` (`csbot.plugins` by default).

For example, a minimal plugin that does nothing might live in `csbot/plugins/nothing.py` and look like:

```
from csbot.plugin import Plugin

class Nothing(Plugin):
    pass
```

A plugin's name is its class name in lowercase[1] and must be unique, so plugin classes should be named meaningfully. Changing a plugin name will cause it to lose access to its associated configuration and database, so try not to do that unless you're prepared to migrate these things.

The vast majority of interaction with the outside world is through subscribing to events and registering commands.

1.2 Events

Root events are generated when the bot receives data from the IRC server, and further events may be generated while handling an event.

All events are represented by the *Event* class, which is a dictionary of event-related information with some additional helpful attributes. See *Events* for further information on the *Event* class and available events.

Events are hooked with the *Plugin.hook()* decorator. The decorated method will be called for every event that matches the specified *event_type*, with the event object as the only argument. For example, a basic logging plugin that prints sent and received data:

[1] This can be changed by overriding the `plugin_name()` class method if absolutely necessary.

```
class Logger(Plugin):
    @Plugin.hook('core.raw.sent')
    def sent(self, e):
        print('<-- ' + e['message'])

    @Plugin.hook('core.raw.received')
    def received(self, e):
        print('--> ' + e['message'])
```

A single handler can hook more than one event:

```
class MessagePrinter(Plugin):
    @Plugin.hook('core.message.privmsg')
    @Plugin.hook('core.message.notice')
    def got_message(self, e):
        """Print out all messages, ignoring if they were PRIVMSG or NOTICE."""
        print(e['message'])
```

1.3 Commands

Registering commands provides a more structured way for users to interact with a plugin. A command can be any unique, non-empty sequence of non-whitespace characters, and are invoked when prefixed with the bot's configured command prefix. Command events use the *CommandEvent* class, extending a `core.message.privmsg` *Event* and adding the *arguments()* method and the `command` and `data` items.

```
class CommandTest(Plugin):
    @Plugin.command('test')
    def hello(self, e):
        print(e['command'] + ' invoked with arguments ' + repr(e.arguments()))
```

A single handler can be registered for more than one command, e.g. to give aliases, and commands and hooks can be freely mixed.

```
class Friendly(Plugin):
    @Plugin.hook('core.channel.joined')
    @Plugin.command('hello')
    @Plugin.command('hi')
    def hello(self, e):
        e.protocol.msg(e['channel'], 'Hello, ' + nick(e['user']))
```

1.4 Responding: the `BotProtocol` object

In the above example the `Event.protocol` attribute was used to respond back to the IRC server. This attribute is an instance of `BotProtocol`, which subclasses twisted.words.protocols.irc.IRCClient for IRC protocol support. The documentation for IRCClient is the best place to find out what methods are supported when responding to an event or command.

1.5 Configuration

Basic string key/value configuration can be stored in an INI-style file. A plugin's *config* attribute is a shortcut to a configuration section with the same name as the plugin. The Python 3 `configparser` is used instead of the Python 2 `ConfigParser` because it supports the mapping access protocol, i.e. it acts like a dictionary in addition to supporting its own API.

An example of using plugin configuration:

```
class Say(Plugin):
    @Plugin.command('say')
    def say(self, e):
        if self.config.getboolean('shout', False):
            e.protocol.msg(e['reply_to'], e['data'].upper() + '!')
        else:
            e.protocol.msg(e['reply_to'], e['data'])
```

For even more convenience, automatic fallback values are supported through the *CONFIG_DEFAULTS* attribute when using the *config_get()* or *config_getboolean()* methods instead of the corresponding methods on *config*. This is encouraged, since it makes it clear what configuration the plugin supports and what the default values are by looking at just one part of the plugin source code. The above example would look like this:

```
class Say(Plugin):
    CONFIG_DEFAULTS = {
        'shout': False,
    }

    @Plugin.command('say')
    def say(self, e):
        if self.config_getboolean('shout'):
            e.protocol.msg(e['reply_to'], e['data'].upper() + '!')
        else:
            e.protocol.msg(e['reply_to'], e['data'])
```

Configuration can be changed at runtime, but won't be saved. This allows for temporary state changes, whilst ensuring the startup state of the bot reflects the configuration file. For example, the above plugin could be modified with a toggle for the "shout" mode:

```
class Say(Plugin):
    # ...
    @Plugin.command('toggle')
    def toggle(self, e):
        self.config['shout'] = not self.config_get('shout')
```

1.6 Database

The bot supports easy access to MongoDB through PyMongo. Plugins have a `db` attribute which is a `pymongo.database.Database`, unique to the plugin and created as needed. Refer to the PyMongo documentation for further guidance on using the API.

CHAPTER 2

Events

All events are represented by *Event* instances. Every event has the following attributes:

`Event`**.bot = None**
The *Bot* which triggered the event.

`Event`**.event_type = None**
The name of the event.

`Event`**.datetime = None**
The value of `datetime.datetime.now()` when the event was triggered.

Event instances are also dictionaries, and the keys present depend on the particular event type. The following sections describe each event, specified as `event_type(keys)`.

2.1 Raw events

These events are very low-level and most plugins shouldn't need them.

core.raw.connected
Client established connection.

core.raw.disconnected
Client lost connection.

core.raw.sent(message)
Client sent *message* to the server.

core.raw.received(message)
Client received *message* from the server.

2.2 Bot events

These events represent changes in the bot's state.

core.self.connected
IRC connection successfully established.

core.self.joined(channel)
Client joined *channel*.

core.self.left(channel)
Client left *channel*.

2.3 Message events

These events occur when messages are received by the bot.

core.message.privmsg(channel, user, message, is_private, reply_to)
Received *message* from *user* which was sent to *channel*. If the message was sent directly to the client, i.e. *channel* is the client's nick and not a channel name, then *is_private* will be True and any response should be to *user*, not *channel*. *reply_to* is the channel/user any response should be sent to.

core.message.notice(channel, user, message, is_private, reply_to)
As `core.message.privmsg`, but representing a NOTICE rather than a PRIVMSG. Bear in mind that according to RFC 1459 "automatic replies must never be sent in response to a NOTICE message" - this definitely applies to bot functionality!

core.message.action(channel, user, message, is_private, reply_to)
Received a `CTCP ACTION` of *message* from *user* sent to *channel*. Other arguments are as for `core.message.privmsg`.

2.4 Channel events

These events occur when something about the channel changes, e.g. people joining or leaving, the topic changing, etc.

core.channel.joined(channel, user)
user joined *channel*.

core.channel.left(channel, user)
user left *channel*.

core.channel.names(channel, names, raw_names)
Received the list of users currently in the channel, in response to a `NAMES` command.

cores.channel.topic(channel, author, topic)
Fired whenever the channel topic is changed, and also immediately after joining a channel. The *author* field will usually be the server name when joining a channel (on Freenode, at least), and the nick of the user setting the topic when the topic has been changed.

2.5 User events

These events occur when a user changes state in some way, i.e. actions that aren't limited to a single channel.

core.user.quit(user, message)

core.user.renamed(oldnick, newnick)

CHAPTER 3

csbot

3.1 csbot package

3.1.1 Subpackages

csbot.plugins package

Submodules

csbot.plugins.auth module

class csbot.plugins.auth.**PermissionDB**
Bases: collections.defaultdict

A helper class for assembling the permissions database.

process(*entity*, *permissions*)
Process a configuration entry, where *entity* is an account name, `@group` name or `*` and *permissions* is a space-separated list of permissions to grant.

get_permissions(*entity*)
Get the set of permissions for *entity*.

The union of the permissions for *entity* and the universal (`*`) permissions is returned. If *entity* is `None`, only the universal permissions are returned.

check(*entity*, *permission*, *channel=None*)
Check if *entity* has *permission*.

If *channel* is present, check for a channel permission, otherwise check for a bot permission. Compatible wildcard permissions are also checked.

class csbot.plugins.auth.**Auth**(*bot*)
Bases: *csbot.plugin.Plugin*

PLUGIN_DEPENDS = ['usertrack']

setup()
Plugin setup.

- Replace all `ProvidedByPlugin` attributes.
- Fire all plugin integration methods.
- Register all commands provided by the plugin.

check(*nick*, *perm*, *channel=None*)

check_or_error(*e*, *perm*, *channel=None*)

csbot.plugins.calc module

csbot.plugins.calc.**is_too_long**(*n*)

csbot.plugins.calc.**guarded_power**(*a*, *b*)
A limited power function to make sure that commands do not take too long to process.

csbot.plugins.calc.**guarded_lshift**(*a*, *b*)

csbot.plugins.calc.**guarded_rshift**(*a*, *b*)

csbot.plugins.calc.**guarded_factorial**(*a*)

class csbot.plugins.calc.**CalcEval**
Bases: ast.NodeVisitor

visit_Module(*node*)

visit_Expr(*node*)

visit_BinOp(*node*)

visit_UnaryOp(*node*)

visit_Compare(*node*)

visit_Call(*node*)

visit_Name(*node*)

visit_Num(*node*)

visit_NameConstant(*node*)

visit_Str(*node*)

generic_visit(*node*)
Fallback visitor which always raises an exception.

We evaluate expressions by using return values of node visitors, and *generic_visit()* returns None, therefore if it's called we know this is an expression we don't support and should give an error.

exception csbot.plugins.calc.**CalcError**
Bases: Exception

class csbot.plugins.calc.**Calc**(*bot*)
Bases: *csbot.plugin.Plugin*

A plugin that calculates things.

do_some_calc(*e*)
What? You don't have a calculator handy?

csbot.plugins.cron module

class csbot.plugins.cron.**Cron**(*bot*)
Bases: *csbot.plugin.Plugin*

Time, that most mysterious of things. What is it? Is it discrete or continuous? What was before time? Does that even make sense to ask? This plugin will attempt to address some, perhaps all, of these questions.

More seriously, this plugin allows the scheduling of events. Due to computers being the constructs of fallible humans, it's not guaranteed that a callback will be run precisely when you want it to be. Furthermore, if you schedule multiple events at the same time, don't make any assumptions about the order in which they'll be called.

Example of usage:

class MyPlugin(Plugin): cron = Plugin.use('cron')

def setup(self): ... self.cron.after(

"hello world", datetime.timedelta(days=1), "callback")

def callback(self, when): self.log.info(u'I got called at {}'.format(when))

@Plugin.hook('cron.hourly') def hourlyevent(self, e):

self.log.info(u'An hour has passed')

tasks
Descriptor for plugin attributes that get (and cache) a value from another plugin.

See `Plugin.use()`.

setup()
Plugin setup.

- Replace all `ProvidedByPlugin` attributes.
- Fire all plugin integration methods.
- Register all commands provided by the plugin.

teardown()
Plugin teardown.

- Unregister all commands provided by the plugin.

fire_event(*now*, *name*)
Fire off a regular event.

This gets called by the scheduler at the appropriate time.

provide(*plugin_name*)
Return the crond for the given plugin.

match_task(*owner*, *name=None*, *args=None*, *kwargs=None*)
Create a MongoDB search for a task definition.

schedule(*owner*, *name*, *when*, *interval=None*, *callback=None*, *args=None*, *kwargs=None*)
Schedule a new task.

Parameters

- **owner** – The plugin which created the task
- **name** – The name of the task
- **when** – The datetime to trigger the task at
- **interval** – Optionally, reschedule at when + interval when triggered. Gives rise to repeating tasks.
- **callback** – Call owner.callback when triggered; if None, call owner.name.
- **args** – Callback positional arguments.
- **kwargs** – Callback keyword arguments.

The signature of a task is `(owner, name, args, kwargs)`, and trying to create a task with the same signature as an existing task will raise *DuplicateTaskError*. Any subset of the signature can be used to *unschedule()* all matching tasks (`owner` is mandatory).

unschedule(*owner*, *name=None*, *args=None*, *kwargs=None*)
Unschedule a task.

Removes all existing tasks that match based on the criteria passed as arguments (see *match_task()*).

This could result in the scheduler having nothing to do in its next call, but this isn't a problem as it's not a very intensive function, so there's no point in rescheduling it here.

schedule_event_runner()
Schedule the event runner.

Set up a delayed call for *event_runner()* to happen no sooner than is required by the next scheduled task. If a different call already exists it is replaced.

event_runner()
Run pending tasks.

Run all tasks which have a trigger time in the past, and then reschedule self to run in time for the next task.

exception csbot.plugins.cron.**DuplicateTaskError**
Bases: `Exception`

Task with a given signature already exists.

This can be raised by *Cron.schedule()* if a plugin tries to register two events with the same name.

class csbot.plugins.cron.**PluginCron**(*cron*, *plugin*)
Bases: `object`

Interface to the cron methods restricted to *plugin* as the task owner..

All of the scheduling functions have a signature of the form (name, time, method_name, *args, **kwargs).

This means that at the appropriate time, the method plugin.method_name will be called with the arguments (time, *args, **kwargs), where the time argument is the time it was supposed to be run by the scheduler (which may not be identical to teh actual time it is run).

These functions will raise a DuplicateNameException if you try to schedule two events with the same name.

schedule(*name*, *when*, *interval=None*, *callback=None*, *args=None*, *kwargs=None*)
Pass through to *Cron.schedule()*, adding *owner* argument.

after(*_delay*, *_name*, *_method_name*, **args*, ***kwargs*)
Schedule an event to occur after the timedelta delay has passed.

at(*_when*, *_name*, *_method_name*, **args*, ***kwargs*)
Schedule an event to occur at a given time.

every(*_freq*, *_name*, *_method_name*, **args*, ***kwargs*)
Schedule an event to occur every time the delay passes.

unschedule(*name*, *args=None*, *kwargs=None*)
Pass through to *Cron.unschedule()*, adding *owner* argument.

unschedule_all()
Unschedule all tasks for this plugin.

This could be supported by *unschedule()*, but it's nice to prevent code accidentally wiping all of a plugin's tasks.

csbot.plugins.csyork module

class csbot.plugins.csyork.**CSYork**(*bot*)
Bases: *csbot.plugin.Plugin*

Amusing replacements for various #cs-york members

respond(*e*)

csbot.plugins.github module

GitHub Deployment Tracking

GitHub's Deployments API allows a repository to track deployment activity. For example, deployments of the main instance of csbot can be seen at https://github.com/HackSoc/csbot/deployments.

Getting csbot to report deployments to your repository during bot startup requires the following:

- `SOURCE_COMMIT` environment variable set to the current git revision (the Docker image has this baked in)
- `--env-name` command-line option (defaults to `development`)
- `--github-repo` command-line option with the repository to report deployments to (e.g. `HackSoc/csbot`)
- `--github-token` command-line option with a GitHub "personal access token" that has `repo_deployment` scope

Note: Deployments API functionality is implemented in *csbot.cli*, not here.

GitHub Webhooks

The GitHub plugin provides a webhook endpoint that will turn incoming events into messages that are sent to IRC channels. To use the GitHub webhook, the *webserver* and *webhook* plugins must be enabled in addition to this one, and the csbot webserver must be exposed to the internet somehow.

Follow the GitHub documentation to create a webhook on the desired repository, with the following settings:

- **Payload URL**: see *webhook* for how webhook URL routing works
- **Content type**: `application/json`
- **Secret**: the same value as chosen for the `secret` plugin option, for signing payloads
- **Which events ...**: Configure for whichever events you want to handle

Configuration

The following configuration options are supported in the `[github]` config section:

Setting	Description
`secret`	The secret used when creating the GitHub webhook. **Optional**, will not verify payload signatures if unset.
`notify`	Space-separated list of IRC channels to send messages to.
`fmt/[...]`	Format strings to use for particular events, for turning an event into an IRC message. See below.
`fmt.[...]`	Re-usable format string fragments. See below.

`secret` and `notify` can be overridden on a per-repository basis, in a `[github/{repo}]` config section, e.g. `[github/HackSoc/csbot]`.

Event format strings

When writing format strings to handle GitHub webhook events, it's essential to refer to the GitHub Event Types & Payloads documentation.

Each event `event_type`, and possibly an `event_subtype`. The `event_type` always corresponds to the "Webhook event name" defined by GitHub's documentation, e.g. `release` for *ReleaseEvent*. The `event_subtype` is generally the `action` from the payload, if that event type has one (but see below for exceptions).

The plugin will attempt to find the most specific config option that exists to supply a format string:

- For an event with `event_type` and `event_subtype`, will try `fmt/event_type/event_subtype`, `fmt/event_type/*` and `fmt/*`
- For an event with no `event_subtype`, will try `fmt/event_type` and `fmt/*`

The first config option that exists will be used, and if that format string is empty (zero-length string, None, False) then no message will be sent. This means it's possible to set a useful format string for `fmt/issues/*`, but then set an empty format for `fmt/issues/labeled` and `fmt/issues/unlabeled` to ignore some unwanted noise.

The string is formatted with the context of the entire webhook payload, plus additional keys for `event_type`, `event_subtype` and `event_name` (which is `{event_type}/{event_subtype}` if there is an `event_subtype`, otherwise `{event_type}`. (But see below for exceptions where additional context exists.)

Re-usable format strings

There are a lot of recurring structures in the GitHub webhook payloads, and those will usually want to be formatted in similar ways in resulting messages. For example, it might be desirable to start every message with the repository name and user that caused the event. Instead of duplicating the same fragment of format string for each event type, which makes the format strings long and hard to maintain, a format string fragment can be defined as a `fmt.name` config option, and referenced in another format string as `{fmt.name}`. These fragments will get formatted with the same context as the top-level format string.

Customised event handling

To represent certain events more clearly, additional processing is required, either to extend the string format context or to introduce an `event_subtype` where there is no `action` in the payload. This is the approach needed

when thinking "I wish string formatting had conditionals". Implementing such handling is done by creating a `handle_{event_type}` method, which should ultimately call `generic_handler` with appropriate arguments.

There is already customised handling for the following:

- **push**
  - Sets `event_subtype`: `forced` for forced update of a ref, and `pushed` for regular pushes
  - Sets `count`: number of commits pushed to the ref
  - Sets `short_ref`: only the final element of the long ref name, e.g. `v1.0` from `refs/tags/v1.0`
- **pull_request**
  - Overrides `event_subtype` with `merged` if PR was `closed` due to a merge
- **pull_request_review**
  - Sets `review_state` to a human-readable version of the review state

Module contents

class csbot.plugins.github.**GitHub**(*bot*)

Bases: *csbot.plugin.Plugin*

PLUGIN_DEPENDS = ['webhook']

CONFIG_DEFAULTS = {'debug_payloads': False, 'fmt/*': None, 'notify': '', 'secret':

CONFIG_ENVVARS = {'secret': ['GITHUB_WEBHOOK_SECRET']}

config_get(*key*, *repo=None*)

A special implementation of `Plugin.config_get()` which looks at a repo-based configuration subsection before the plugin's configuration section.

classmethod find_by_matchers(*matchers*, *d*, *default=<object object>*)

generic_handler(*data*, *event_type*, *event_subtype=None*, *event_subtype_key='action'*, *context=None*)

handle_pull_request(*data*, *event_type*)

handle_pull_request_review(*data*, *event_type*)

handle_push(*data*, *event_type*)

webhook(*e*)

class csbot.plugins.github.**MessageFormatter**(*config_get*)

Bases: `string.Formatter`

get_field(*field_name*, *args*, *kwargs*)

csbot.plugins.helix module

class csbot.plugins.helix.**Helix**(*bot*)

Bases: *csbot.plugin.Plugin*

The premier csbot plugin, allowing mere mortals to put questions to the mighty helix, and receive his divine wisdom.

Notes:

- The popular online version basically just selects a random outcome, and saves it with a random url so that it can be reused if the same question is asked.
- I'm lazy, so I'm just going to hash whatever the person puts in and mod the resulting value (taken from hex) to pick out an element of the outcomes list. That way if the same questions gets asked twice, it gets (hopefully) the same answer.

outcomes = ['It is certain', 'It is decidedly so', 'Without a doubt', 'Yes definitely'

setup()
Plugin setup.

- Replace all `ProvidedByPlugin` attributes.
- Fire all plugin integration methods.
- Register all commands provided by the plugin.

ask_the_almighty_helix(*e*)
Ask and you shall recieve.

csbot.plugins.hoogle module

class csbot.plugins.hoogle.**Hoogle**(*bot*)
Bases: *csbot.plugin.Plugin*

CONFIG_DEFAULTS = {'results': 5}

setup()
Plugin setup.

- Replace all `ProvidedByPlugin` attributes.
- Fire all plugin integration methods.
- Register all commands provided by the plugin.

search_hoogle(*e*)
Search Hoogle with a given string and return the first few (exact number configurable) results.

csbot.plugins.imgur module

exception csbot.plugins.imgur.**ImgurError**
Bases: Exception

class csbot.plugins.imgur.**Imgur**(*bot*)
Bases: *csbot.plugin.Plugin*

CONFIG_DEFAULTS = {'client_id': None, 'client_secret': None}

CONFIG_ENVVARS = {'client_id': ['IMGUR_CLIENT_ID'], 'client_secret': ['IMGUR_CLIENT_

integrate_with_linkinfo(*linkinfo*)

csbot.plugins.last module

class csbot.plugins.last.**Last**(*bot*)
Bases: *csbot.plugin.Plugin*

Utility plugin to record the last message (and time said) of a user. Records both messages and actions individually, and allows querying on either.

db
Descriptor for plugin attributes that get (and cache) a value from another plugin.

See `Plugin.use()`.

last(*nick*, *channel=None*, *msgtype=None*)
Get the last thing said (including actions) by a given nick, optionally filtering by channel.

last_message(*nick*, *channel=None*)
Get the last message sent by a nick, optionally filtering by channel.

last_action(*nick*, *channel=None*)
Get the last action sent by a nick, optionally filtering by channel.

last_command(*nick*, *channel=None*)
Get the last command sent by a nick, optionally filtering by channel.

record_message(*event*)
Record the receipt of a new message.

record_command(*event*)
Record the receipt of a new command.

record_action(*event*)
Record the receipt of a new action.

record(*event*, *nick*, *channel*, *msgtype*, *msg*)
Record a new message, of a given type.

show_seen(*event*)

csbot.plugins.linkinfo module

class csbot.plugins.linkinfo.**LinkInfoHandler**(*filter: Callable[[urllib.parse.ParseResult], LinkInfoFilterResult], handler: Callable[[urllib.parse.ParseResult, LinkInfoFilterResult], Optional[LinkInfoResult]], exclusive: bool*)

Bases: `typing.Generic`

class csbot.plugins.linkinfo.**LinkInfoResult**(*url: str*, *text: str*, *is_error: bool = False*, *nsfw: bool = False*, *is_redundant: bool = False*)

Bases: `object`

url
The URL requested

text
Information about the URL

is_error
Is an error?

nsfw
URL is not safe for work?

is_redundant
URL information is redundant? (e.g. duplicated in URL string)

get_message()

class csbot.plugins.linkinfo.**LinkInfo**(**args*, ***kwargs*)
Bases: *csbot.plugin.Plugin*

class Config(*raw_data=None*, *trusted_data=None*, *deserialize_mapping=None*, *init=True*, *partial=True*, *strict=True*, *validate=False*, *app_data=None*, *lazy=False*, ***kwargs*)
Bases: *csbot.config.Config*

scan_limit = <IntType() instance on Config as 'scan_limit'>

minimum_slug_length = <IntType() instance on Config as 'minimum_slug_length'>

max_file_ext_length = <IntType() instance on Config as 'max_file_ext_length'>

minimum_path_match = <FloatType() instance on Config as 'minimum_path_match'>

rate_limit_time = <IntType() instance on Config as 'rate_limit_time'>

rate_limit_count = <IntType() instance on Config as 'rate_limit_count'>

max_response_size = <IntType() instance on Config as 'max_response_size'>

register_handler(*filter*, *handler*, *exclusive=False*)
Add a URL handler.

filter should be a function that returns a True-like or False-like value to indicate whether *handler* should be run for a particular URL. The URL is supplied as a `urlparse:ParseResult` instance.

If *handler* is called, it will be as `handler(url, filter(url))`. The filter result is useful for accessing the results of a regular expression filter, for example. The result should be a *LinkInfoResult* instance. If the result is None instead, the processing will fall through to the next handler; this is the best way to signal that a handler doesn't know what to do with a particular URL.

If *exclusive* is True, the fall-through behaviour will not happen, instead terminating the handling with the result of calling *handler*.

register_exclude(*filter*)
Add a URL exclusion filter.

filter should be a function that returns a True-like or False-like value to indicate whether or not a URL should be excluded from the default title-scraping behaviour (after all registered handlers have been tried). The URL is supplied as a `urlparse.ParseResult` instance.

get_link_info(*original_url*)
Get information about a URL.

Using the *original_url* string, run the chain of URL handlers and excludes to get a *LinkInfoResult*.

link_command(*e*)
Handle the "link" command.

Fetch information about a specified URL, e.g. `!link http://google.com`. The link can be explicitly marked as NSFW by including the string anywhere in the trailing string, e.g. `!link http://lots-of-porn.com nsfw`.

scan_privmsg(*e*)
Scan the data of PRIVMSG events for URLs and respond with information about them.

scrape_html_title(*url*)
Scrape the `<title>` tag contents from the HTML page at *url*.

Returns a *LinkInfoResult*.

csbot.plugins.logger module

class csbot.plugins.logger.**Logger**(*bot*)
Bases: *csbot.plugin.Plugin*

raw_log = <Logger csbot.raw_log (WARNING)>

pretty_log = <Logger csbot.pretty_log (WARNING)>

raw_received(*event*)

raw_sent(*event*)

connected(*event*)

disconnected(*event*)

signedon(*event*)

joined(*event*)

left(*event*)

user_joined(*event*)

user_left(*event*)

names(*event*)

topic(*event*)

privmsg(*event*)

notice(*event*)

action(*event*)

quit(*event*)

renamed(*event*)

command(*event*)
Tag a command to be registered by `setup()`.

Additional keyword arguments are added to a metadata dictionary that gets stored with the command. This is a good place to put, for example, the help string for the command:

```
@Plugin.command('foo', help='foo: does something amazing')
def foo_command(self, e):
    pass
```

csbot.plugins.mongodb module

class csbot.plugins.mongodb.**MongoDB**(**args*, ***kwargs*)
Bases: *csbot.plugin.Plugin*

A plugin that provides access to a MongoDB server via pymongo.

CONFIG_DEFAULTS = {'mode': 'uri', 'uri': 'mongodb://localhost:27017/csbot'}

CONFIG_ENVVARS = {'uri': ['MONGOLAB_URI', 'MONGODB_URI']}

provide(*plugin_name*, *collection*)
Get a MongoDB collection for `{plugin_name}__{collection}`.

csbot.plugins.termdates module

class csbot.plugins.termdates.**Term**(*key: str*, *start_date: datetime.datetime*)
Bases: object

first_monday

last_friday

get_week_number(*date: datetime.date*) → int
Get the "term week number" of a date relative to this term.

The first week of term is week 1, not week 0. Week 1 starts at the Monday of the term's start date, even if the term's start date is not Monday. Any date before the start of the term gives a negative week number.

get_week_start(*week_number: int*) → datetime.datetime
Get the start date of a specific week number relative to this term.

The first week of term is week 1, not week 0, although this method allows both. When referring to the first week of term, the start date is the term start date (which may not be a Monday). All other weeks start on their Monday.

class csbot.plugins.termdates.**TermDates**(*bot*)
Bases: *csbot.plugin.Plugin*

A wonderful plugin allowing old people (graduates) to keep track of the ever-changing calendar.

DATE_FORMAT = '%Y-%m-%d'

TERM_KEYS = ('aut', 'spr', 'sum')

db_terms
Descriptor for plugin attributes that get (and cache) a value from another plugin.

See `Plugin.use()`.

terms = None

setup()
Plugin setup.

- Replace all `ProvidedByPlugin` attributes.
- Fire all plugin integration methods.
- Register all commands provided by the plugin.

initialised
If no term dates have been set, the calendar is uninitialised and can't be asked about term thing.

termdates(*e*)

week(*e*)

termdates_set(*e*)

csbot.plugins.topic module

class csbot.plugins.topic.**Topic**(*bot*)
Bases: *csbot.plugin.Plugin*

PLUGIN_DEPENDS = ['auth']

CONFIG_DEFAULTS = {'end': '', 'history': 5, 'sep': '|', 'start': ''}

setup()
Plugin setup.

- Replace all `ProvidedByPlugin` attributes.
- Fire all plugin integration methods.
- Register all commands provided by the plugin.

config_get(*key*, *channel=None*)
A special implementation of `Plugin.config_get()` which looks at a channel-based configuration subsection before the plugin's configuration section.

topic_changed(*e*)

topic(*e*)

topic_history(*e*)

topic_undo(*e*)

topic_append(*e*)

topic_pop(*e*)

topic_replace(*e*)

topic_insert(*e*)

csbot.plugins.usertrack module

class csbot.plugins.usertrack.**UserDict**
Bases: `collections.defaultdict`

static create_user(*nick*)

copy_or_create(*nick*)

class csbot.plugins.usertrack.**UserTrack**(*bot*)
Bases: *csbot.plugin.Plugin*

setup()
Plugin setup.

- Replace all `ProvidedByPlugin` attributes.
- Fire all plugin integration methods.
- Register all commands provided by the plugin.

get_user(*nick*)
Get a copy of the user record for *nick*.

account_command(*e*)

csbot.plugins.webhook module

Uses *webserver* to create a generic URL for incoming webhooks so that other plugins can handle webhook events.

To act as a webhook handler, a plugin should hook the `webhook.{service}` event, for example:

```
class MyPlugin(Plugin):
    @Plugin.hook('webhook.myplugin')
    async def webhook(self, e):
        self.log.info(f'Handling {e["request"]}')
```

The `request` key of the event contains the `aiohttp.web.Request` object.

Note: The webhook plugin only responds to `POST` requests.

Configuration

The following configuration options are supported in the `[webhook]` config section:

Setting	Description
`prefix`	URL prefix for the web server sub-application. Default: `/webhook`.
`url_secret`	Extra URL component to make valid endpoints hard to guess.

URL Format & Request Handling

The URL path for a webhook is `{prefix}/{service}/{url_secret}`. The host and port elements, plus any additional prefix, are determined by the *webserver* plugin and/or any reverse-proxy that is in front of it.

For example, the main deployment of csbot received webhooks at `https://{host}/csbot/webhook/{service}/{url_secret}` and sits behind nginx with the following configuration:

```
location /csbot/ {
    proxy_pass http://localhost:8180/;
}
```

Module contents

class csbot.plugins.webhook.**Webhook**(*bot*)

Bases: *csbot.plugin.Plugin*

CONFIG_DEFAULTS = {'prefix': '/webhook', 'url_secret': ''}

CONFIG_ENVVARS = {'url_secret': ['WEBHOOK_SECRET']}

create_app(*e*)

request_handler(*request*)

csbot.plugins.webserver module

Creates a web server using `aiohttp` so that other plugins can register URL handlers.

To register a URL handler, a plugin should hook the `webserver.build` event and create a sub-application, for example:

```
class MyPlugin(Plugin):
    @Plugin.hook('webserver.build')
    def create_app(self, e):
        with e['webserver'].create_subapp('/my_plugin') as app:
            app.add_routes([web.get('/{item}', self.request_handler)])

    async def request_handler(self, request):
        return web.Response(text=f'No {request.match_info["item"]} here, oh dear!')
```

Configuration

The following configuration options are supported in the `[webserver]` config section:

Setting	Description
`host`	Hostname/IP address to listen on. Default: `localhost`.
`port`	Port to listen on. Default: `1337`.

Module contents

class csbot.plugins.webserver.**WebServer**(*bot*)

Bases: *csbot.plugin.Plugin*

CONFIG_DEFAULTS = {'host': 'localhost', 'port': 1337}

setup()

Plugin setup.

- Replace all `ProvidedByPlugin` attributes.
- Fire all plugin integration methods.
- Register all commands provided by the plugin.

teardown()

Plugin teardown.

- Unregister all commands provided by the plugin.

create_subapp(*prefix*)

csbot.plugins.whois module

class csbot.plugins.whois.**Whois**(*bot*)

Bases: *csbot.plugin.Plugin*

Associate data with a user and a channel. Users can update their own data, and it persists over nick changes.

PLUGIN_DEPENDS = ['usertrack']

whoisdb

Descriptor for plugin attributes that get (and cache) a value from another plugin.

See `Plugin.use()`.

whois_lookup(*nick*, *channel*, *db=None*)

Performs a whois lookup for a nick

whois_set(*nick*, *whois_str*, *channel=None*, *db=None*)

whois_unset(*nick*, *channel=None*, *db=None*)

whois(*e*)
Look up a user by nick, and return what data they have set for themselves (or an error message if there is no data)

setdefault(*e*)

set(*e*)
Allow a user to associate data with themselves for this channel.

unset(*e*)

unsetdefault(*e*)

identify_user(*nick*, *channel=None*)
Identify a user: by account if authed, if not, by nick. Produces a dict suitable for throwing at mongo.

csbot.plugins.xkcd module

csbot.plugins.xkcd.**fix_json_unicode**(*data*)
Attempts to fix the unicode & HTML silliness that is included in the json data. Why Randall, Why?

class csbot.plugins.xkcd.**xkcd**(*bot*)
Bases: *csbot.plugin.Plugin*

A plugin that does some xkcd things. Based on williebot xkcd plugin.

exception XKCDError
Bases: Exception

randall_is_awesome(*e*)
Well, Randall sucks at unicode actually :(

linkinfo_integrate(*linkinfo*)
Handle recognised xkcd urls.

csbot.plugins.xkcd.**get_info**(*number=None*)
Gets the json data for a particular comic (or the latest, if none provided).

csbot.plugins.youtube module

csbot.plugins.youtube.**get_yt_id**(*url*)
Gets the video ID from a urllib ParseResult object.

exception csbot.plugins.youtube.**YoutubeError**(*http_error*)
Bases: Exception

Signifies some error occurred accessing the Youtube API.

This is only used for actual errors, e.g. invalid API key, not failure to find any data matching a query.

Pass the HttpError from the API call as an argument.

class csbot.plugins.youtube.**Youtube**(*bot*)
Bases: *csbot.plugin.Plugin*

A plugin that does some youtube things. Based on williebot youtube plugin.

CONFIG_DEFAULTS = {'api_key': ''}

CONFIG_ENVVARS = {'api_key': ['YOUTUBE_DATA_API_KEY']}

RESPONSE = '"{title}" [{duration}] (by {uploader} at {uploaded}) | Views: {views}'

CMD_RESPONSE = '"{title}" [{duration}] (by {uploader} at {uploaded}) | Views: {views}

all_hail_our_google_overlords(*e*)
I for one, welcome our Google overlords.

get_video_json(*id*)

linkinfo_integrate(*linkinfo*)
Handle recognised youtube urls.

Module contents

3.1.2 Submodules

csbot.cli module

csbot.cli.**load_ini**(*f*)

csbot.cli.**load_json**(*f*)

csbot.cli.**load_toml**(*f*)

csbot.cli.**github_report_deploy**(*github_token*, *github_repo*, *env_name*, *revision*)

csbot.cli.**rollbar_report_deploy**(*rollbar_token*, *env_name*, *revision*)

csbot.config module

class csbot.config.**Config**(*raw_data=None*, *trusted_data=None*, *deserialize_mapping=None*, *init=True*, *partial=True*, *strict=True*, *validate=False*, *app_data=None*, *lazy=False*, ***kwargs*)
Bases: schematics.deprecated.Model

Base class for configuration schemas.

Use *option()*, *option_list()* and *option_map()* to create fields in the schema. Schemas are also valid option types, so deeper structures can be defined.

```
>>> class MyConfig(Config):
...     delay = option(float, default=0.5, help="Number of seconds to wait")
...     notify = option_list(str, help="Users to notify")
```

csbot.config.**ConfigError**
alias of schematics.exceptions.DataError

csbot.config.**example_mode**()
For the duration of this context manager, try to use example values before default values.

class csbot.config.**WordList**(*min_size=None*, *max_size=None*, ***kwargs*)
Bases: schematics.types.compound.ListType

A list of strings that also accepts a space-separated string instead.

convert(*value*, *context=None*)

MESSAGES = {'choices': <schematics.translator.LazyText object>, 'required': <schemat

csbot.config.**is_config**(*obj: Any*) → bool
Is *obj* a configuration class or instance?

csbot.config.**is_allowable_type**(*cls: Type[CT_co]*) → bool
Is *cls* allowed as a configuration option type?

csbot.config.**structure**(*data: Mapping[str, Any]*, *cls: Type[csbot.config.Config]*) → csbot.config.Config
Create an instance of *cls* from plain Python structure *data*.

csbot.config.**unstructure**(*obj: csbot.config.Config*) → Mapping[str, Any]
Get plain Python structured data from *obj*.

csbot.config.**loads**(*s: str*, *cls: Type[csbot.config.Config]*) → csbot.config.Config
Create an instance of *cls* from the TOML in *s*.

csbot.config.**dumps**(*obj: csbot.config.Config*) → str
Get TOML string representation of *obj*.

csbot.config.**load**(*f: TextIO*, *cls: Type[csbot.config.Config]*) → csbot.config.Config
Create an instance of *cls* from the TOML in *f*.

csbot.config.**dump**(*obj: csbot.config.Config*, *f: TextIO*)
Write TOML representation of *obj* to *f*.

csbot.config.**option**(*cls: Type[_B]*, ***, *required: bool = None*, *default: Union[None, _B, Callable[[], Union[None, _B]]] = None*, *example: Union[None, _B, Callable[[], Union[None, _B]]] = None*, *env: Union[str, List[str]] = None*, *help: str*)
Create a configuration option that contains a value of type *cls*.

Parameters

- **cls** – Option type (see *is_allowable_type()*)
- **required** – A non-None value is required? (default: False if default is None, otherwise True)
- **default** – Default value if no value is supplied (default: None)
- **example** – Default value when generating example configuration (default: None)
- **env** – Environment variables to try if no value is supplied, before using default (default: [])
- **help** – Description of option, included when generating example configuration

csbot.config.**option_list**(*cls: Type[_B]*, ***, *default: Union[None, List[_B], Callable[[], Union[None, List[_B]]]] = None*, *example: Union[None, List[_B], Callable[[], Union[None, List[_B]]]] = None*, *help: str*)
Create a configuration option that contains a list of *cls* values.

Parameters

- **cls** – Option type (see *is_allowable_type()*)
- **default** – Default value if no value is supplied (default: empty list)
- **example** – Default value when generating example configuration (default: empty list)
- **help** – Description of option, included when generating example configuration

csbot.config.**option_map**(*cls: Type[_B]*, ***, *default: Union[None, Dict[str, _B], Callable[[], Union[None, Dict[str, _B]]]] = None*, *example: Union[None, Dict[str, _B], Callable[[], Union[None, Dict[str, _B]]]] = None*, *help: str*)
Create a configuration option that contains a mapping of string keys to *cls* values.

Parameters

- **cls** – Option type (see *is_allowable_type()*)
- **default** – Default value if no value is supplied (default: empty list)
- **example** – Default value when generating example configuration (default: empty list)
- **help** – Description of option, included when generating example configuration

csbot.config.**make_example**(*cls: Type[csbot.config.Config]*) → csbot.config.Config
Create an instance of *cls* without supplying data, using "example" or "default" values for each option.

class csbot.config.**TomlExampleGenerator**(*, *commented=False*)
Bases: object

generate(*obj: Union[csbot.config.Config, Type[csbot.config.Config]], stream: TextIO, prefix: List[str] = None*)
Generate an example from *obj* and write it to *stream*.

csbot.config.**generate_toml_example**(*obj: Union[csbot.config.Config, Type[csbot.config.Config]], commented: bool = False*) → str
Generate an example configuration from *obj* as a TOML string.

csbot.core module

exception csbot.core.**PluginError**
Bases: Exception

class csbot.core.**Bot**(*config=None*, *, *plugins: Sequence[Type[csbot.plugin.Plugin]] = None*, *loop=None*)
Bases: *csbot.plugin.SpecialPlugin*, *csbot.irc.IRCClient*

class Config(*raw_data=None*, *trusted_data=None*, *deserialize_mapping=None*, *init=True*, *partial=True*, *strict=True*, *validate=False*, *app_data=None*, *lazy=False*, ***kwargs*)
Bases: *csbot.config.Config*

ircv3 = <BooleanType() instance on Config as 'ircv3'>

nickname = <StringType() instance on Config as 'nickname'>

username = <StringType() instance on Config as 'username'>

realname = <StringType() instance on Config as 'realname'>

auth_method = <StringType() instance on Config as 'auth_method'>

password = <StringType() instance on Config as 'password'>

irc_host = <StringType() instance on Config as 'irc_host'>

irc_port = <IntType() instance on Config as 'irc_port'>

command_prefix = <StringType() instance on Config as 'command_prefix'>

channels = <WordList(StringType) instance on Config as 'channels'>

plugins = <WordList(StringType) instance on Config as 'plugins'>

use_notice = <IntType() instance on Config as 'use_notice'>

client_ping = <IntType() instance on Config as 'client_ping'>

bind_addr = <StringType() instance on Config as 'bind_addr'>

rate_limit_period = <IntType() instance on Config as 'rate_limit_period'>

rate_limit_count = <IntType() instance on Config as 'rate_limit_count'>

available_plugins = None
Dictionary containing available plugins for loading, using straight.plugin to discover plugin classes under a namespace.

bot_setup()
Load plugins defined in configuration and run setup methods.

bot_teardown()
Run plugin teardown methods.

post_event(*event*)

register_command(*cmd*, *metadata*, *f*, *tag=None*)

unregister_command(*cmd*, *tag=None*)

unregister_commands(*tag*)

signedOn(*event*)

privmsg(*event*)
Handle commands inside PRIVMSGs.

show_commands(*e*)

show_plugins(*e*)

emit_new(*event_type*, *data=None*)
Shorthand for firing a new event.

emit(*event*)
Shorthand for firing an existing event.

line_sent(*line: str*)
Callback for sent raw IRC message.

Subclasses can implement this to get access to the actual message that was sent (which may have been truncated from what was passed to `send_line()`).

line_received(*line*)
Callback for received raw IRC message.

recent_messages

on_welcome()
Successfully signed on to the server.

on_joined(*channel*)
Joined a channel.

on_left(*channel*)
Left a channel.

on_privmsg(*user*, *channel*, *message*)
Received a message, either directly or in a channel.

on_notice(*user*, *channel*, *message*)
Received a notice, either directly or in a channel.

on_action(*user*, *channel*, *message*)
Received CTCP ACTION. Common enough to deserve its own event.

on_user_joined(*user*, *channel*)
User joined a channel.

on_user_left(*user*, *channel*, *message*)
User left a channel.

on_user_quit(*user*, *message*)
User disconnected.

on_user_renamed(*oldnick*, *newnick*)
User changed nick.

on_topic_changed(*user*, *channel*, *topic*)
user changed the topic of *channel* to *topic*.

irc_RPL_NAMREPLY(*msg*)

irc_RPL_ENDOFNAMES(*msg*)

on_names(*channel*, *names*, *raw_names*)
Called when the NAMES list for a channel has been received.

identify(*target*)
Find the account for a user or all users in a channel.

connection_lost(*exc*)
Handle a broken connection by attempting to reconnect.

Won't reconnect if the broken connection was deliberate (i.e. `close()` was called).

connection_made()
Callback for successful connection.

Register with the IRC server.

fire_command(*event*)
Dispatch a command event to its callback.

irc_354(*msg*)
Handle "formatted WHO" responses.

on_user_identified(*user*, *account*)

irc_ACCOUNT(*msg*)
Account change notification from `account-notify` capability.

irc_JOIN(*msg*)
Re-implement `JOIN` handler to account for `extended-join` info.

reply(*to*, *message*)
Reply to a nick/channel.

This is not implemented because it should be replaced in the constructor with a reference to a real method, e.g. `self.reply = self.msg`.

classmethod write_example_config(*f*, *plugins=None*, *commented=False*)

csbot.events module

class csbot.events.**HybridEventRunner**(*get_handlers*, *loop=None*)
Bases: `object`

A hybrid synchronous/asynchronous event runner.

get_handlers is called for each event passed to *post_event()*, and should return an iterable of callables to handle that event, each of which will be called with the event object.

Events are processed in the order they are received, with all handlers for an event being called before the handlers for the next event. If a handler returns an awaitable, it is added to a set of asynchronous tasks to wait on.

The future returned by *post_event()* completes only when all events have been processed and all asynchronous tasks have completed.

Parameters

- **get_handlers** – Get functions to call for an event
- **loop** – asyncio event loop to use (default: use current loop)

post_event(*event*)
Post *event* to be handled soon.

event is added to the queue of events.

Returns a future which resolves when the handlers of *event* (and all events generated during those handlers) have completed.

class csbot.events.**Event**(*bot*, *event_type*, *data=None*)
Bases: dict

IRC event information.

Events are dicts of event information, plus some attributes which are applicable for all events.

bot = None
The *Bot* which triggered the event.

event_type = None
The name of the event.

datetime = None
The value of datetime.datetime.now() when the event was triggered.

classmethod extend(*event*, *event_type=None*, *data=None*)
Create a new event by extending an existing event.

The main purpose of this classmethod is to duplicate an event as a new event type, preserving existing information. For example:

reply(*message*)
Send a reply.

For messages that have a reply_to key, instruct the *bot* to send a reply.

class csbot.events.**CommandEvent**(*bot*, *event_type*, *data=None*)
Bases: *csbot.events.Event*

classmethod parse_command(*event*, *prefix*, *nick*)
Attempt to create a *CommandEvent* from a core.message.privmsg event.

A command is signified by *event["message"]* starting with the command prefix string followed by one or more non-space characters.

Returns None if *event['message']* wasn't recognised as being a command.

arguments()
Parse *self["data"]* into a list of arguments using *parse_arguments()*. This might raise a ValueError if the string cannot be parsed, e.g. if there are unmatched quotes.

csbot.irc module

exception csbot.irc.**IRCParseError**
Bases: Exception

Raised by *IRCMessage.parse()* when a message can't be parsed.

class csbot.irc.**IRCMessage**(*raw: str, prefix: Optional[str], command: str, params: List[str], command_name: str*)
Bases: object

Represents an IRC message.

The IRC message format, paraphrased and simplified from RFC2812, is:

```
message = [":" prefix " "] command {" " parameter} [" :" trailing]
```

Has the following attributes:

Parameters

- **raw** (*str*) – The raw IRC message
- **prefix** (*str or None*) – Prefix part of the message, usually the origin
- **command** (*str*) – IRC command
- **params** (*list of str*) – List of command parameters (including trailing)
- **command_name** (*str*) – Name of IRC command (see below)

The *command_name* attribute is intended to be the "readable" form of the *command*. Usually it will be the same as *command*, but numeric replies recognised in RFC2812 will have their corresponding name instead.

raw

prefix

command

params

command_name

REGEX = re.compile('(:(?P<prefix>\\S+))?(?P<command>\\S+)(?P<params>((?!:)\\S+)*)(:
Regular expression to extract message components from a message.

FORCE_TRAILING = {'PRIVMSG', 'QUIT', 'USER'}
Commands to force trailing parameter (`:blah`) for

classmethod parse(*line*)
Create an *IRCMessage* object by parsing a raw message.

classmethod create(*command*, *params=None*, *prefix=None*)
Create an *IRCMessage* from its core components.

The *raw* and *command_name* attributes will be generated based on the message details.

pretty
Get a more readable version of the raw IRC message.

Pretty much identical to the raw IRC message, but numeric commands that have names end up being `NUMERIC/NAME`.

pad_params(*length*, *default=None*)
Pad parameters to *length* with *default*.

Useful when a command has optional parameters:

```
>>> msg = IRCMessage.parse(':nick!user@host KICK #channel other')
>>> channel, nick, reason = msg.params
Traceback (most recent call last):
  ...
ValueError: need more than 2 values to unpack
>>> channel, nick, reason = msg.pad_params(3)
```

class csbot.irc.**IRCUser**(*raw: str, nick: str, user: Optional[str], host: Optional[str]*)
Bases: object

Provide access to the parts of an IRC user string.

The following parts of the user string are available, set to *None* if that part of the string is absent:

Parameters

- **raw** – Raw user string
- **nick** – Nick of the user
- **user** – Username of the user (excluding leading ~)
- **host** – Hostname of the user

```
>>> IRCUser.parse('my_nick!some_user@host.name')
IRCUser(raw='my_nick!some_user@host.name', nick='my_nick', user='some_user', host=
↪'host.name')
```

raw

nick

user

host

REGEX = re.compile('(?P<raw>(?P<nick>[^!]+)(!~*(?P<user>[^@]+))?(@(?P<host>.+))?)')
Username parsing regex. Stripping out the "~" might be a Freenode peculiarity...

classmethod parse(*raw*)
Create an *IRCUser* from a raw user string.

class csbot.irc.**IRCCodec**
Bases: codecs.Codec

The encoding scheme to use for IRC messages.

IRC messages are "just bytes" with no encoding made explicit in the protocol definition or the messages. Ideally we'd like to handle IRC messages as proper strings.

encode(*input*, *errors='strict'*)
Encode a message as UTF-8.

decode(*input*, *errors='strict'*)
Decode a message.

IRC messages could pretty much be in any encoding. Here we just try the two most likely candidates: UTF-8, falling back to CP1252. Unfortunately, any encoding where every byte is valid (e.g. CP1252) makes it impossible to detect encoding errors - if *input* isn't UTF-8 or CP1252-compatible, the result might be a bit odd.

exception csbot.irc.**IRCClientError**
Bases: Exception

class csbot.irc.**IRCClient**(*, *loop=None*, ***kwargs*)
Bases: object

Internet Relay Chat client protocol.

A line-oriented protocol for communicating with IRC servers. It handles receiving data at several layers of abstraction:

- *line_received()*: decoded line
- *message_received()*: parsed *IRCMessage*
- irc_<COMMAND>(msg): called when msg.command == '<COMMAND>'
- on_<event>(...): specific events with specific arguments, e.g. on_quit(user, message)

It also handles sending data at several layers of abstraction:

- *send_line()*: raw IRC command, e.g. self.send_line('JOIN #cs-york-dev')
- *send()*: *IRCMessage*, e.g. self.send(IRCMessage.create('JOIN', params=['#cs-york-dev']))
- <action>(...): e.g. self.join('#cs-york-dev').

The API and implementation is inspired by irc3 and Twisted.

- TODO: NAMES
- TODO: MODE
- TODO: More sophisticated CTCP? (see Twisted)
- TODO: MOTD?
- TODO: SSL

codec = <csbot.irc.IRCCodec object>
Codec for encoding/decoding IRC messages.

static DEFAULTS()
Generate a default configuration. Easier to call this and update the result than relying on dict.copy().

available_capabilities = None
Available client capabilities

enabled_capabilities = None
Enabled client capabilities

disconnect()
Disconnect from the IRC server.

Use *quit()* for a more graceful disconnect.

line_received(*line: str*)
Callback for received raw IRC message.

line_sent(*line: str*)
Callback for sent raw IRC message.

Subclasses can implement this to get access to the actual message that was sent (which may have been truncated from what was passed to *send_line()*).

message_received(*msg*)
Callback for received parsed IRC message.

send_line(*data: str*)
Send a raw IRC message to the server.

Encodes, terminates and sends *data* to the server. If the line would be longer than the maximum allowed by the IRC specification, it is trimmed to fit (without breaking UTF-8 sequences).

If rate limiting is enabled, the message may not be sent immediately.

send(*msg*)
Send an *IRCMessage*.

class Waiter(*predicate: Callable[[csbot.irc.IRCMessage], Tuple[bool, Any]], future: _asyncio.Future*)
Bases: object

PredicateType = typing.Callable[[csbot.irc.IRCMessage], typing.Tuple[bool, typing.A

wait_for_message(*predicate: Callable[[csbot.irc.IRCMessage], Tuple[bool, Any]]*) → _asyncio.Future
Wait for a message that matches *predicate*.

predicate should return a *(did_match, result)* tuple, where *did_match* is a boolean indicating if the message is a match, and *result* is the value to return.

Returns a future that is resolved with *result* on the first matching message.

process_wait_for_message(*msg*)

request_capabilities(*, *enable: Iterable[str] = None*, *disable: Iterable[str] = None*) → Awaitable[bool]
Request a change to the enabled IRCv3 capabilities.

enable and *disable* are sets of capability names, with *disable* taking precedence.

Returns a future which resolves with True if the request is successful, or False otherwise.

set_nick(*nick*)
Ask the server to set our nick.

join(*channel*)
Join a channel.

leave(*channel*, *message=None*)
Leave a channel, with an optional message.

quit(*message=None*, *reconnect=False*)
Leave the server.

If *reconnect* is False, then the client will not attempt to reconnect after the server closes the connection.

msg(*to*, *message*)
Send *message* to a channel/nick.

act(*to*, *action*)
Send *action* as a CTCP ACTION to a channel/nick.

notice(*to*, *message*)
Send *message* as a NOTICE to a channel/nick.

set_topic(*channel*, *topic*)
Try and set a channel's topic.

get_topic(*channel*)
Ask server to send the topic for *channel*.

Will cause *on_topic_changed()* at some point in the future.

ctcp_query(*to*, *command*, *data=None*)
Send CTCP query.

ctcp_reply(*to*, *command*, *data=None*)
Send CTCP reply.

irc_RPL_WELCOME(*msg*)
Received welcome from server, now we can start communicating.

Welcome should include the accepted nick as the first parameter. This may be different to the nick we requested (e.g. truncated to a maximum length); if this is the case we store the new nick and fire the *on_nick_changed()* event.

irc_ERR_NICKNAMEINUSE(*msg*)
Attempted nick is in use, try another.

Adds an underscore to the end of the current nick. If the server truncated the nick, replaces the last non-underscore with an underscore.

irc_PING(*msg*)
IRC PING/PONG keepalive.

irc_CAP(*msg*)
Dispatch `CAP` subcommands to their own methods.

irc_CAP_LS(*msg*)
Response to `CAP LS`, giving list of available capabilities.

irc_CAP_ACK(*msg*)
Response to `CAP REQ`, acknowledging capability changes.

irc_CAP_NAK(*msg*)
Response to `CAP REQ`, rejecting capability changes.

irc_NICK(*msg*)
Somebody's nick changed.

irc_JOIN(*msg*)
Somebody joined a channel.

irc_PART(*msg*)
Somebody left a channel.

irc_KICK(*msg*)
Somebody was kicked from a channel.

irc_QUIT(*msg*)
Somebody quit the server.

irc_TOPIC(*msg*)
A channel's topic changed.

irc_RPL_TOPIC(*msg*)
Topic notification, usually after joining a channel.

irc_PRIVMSG(*msg*)
Received a `PRIVMSG`.

TODO: Implement CTCP queries.

irc_NOTICE(*msg*)
Received a `NOTICE`.

TODO: Implement CTCP replies.

on_capabilities_available(*capabilities*)
Client capabilities are available.

Called with a set of client capability names when we get a response to `CAP LS`.

on_capability_enabled(*name*)
Client capability enabled.

Called when enabling client capability *name* has been acknowledged.

on_capability_disabled(*name*)
Client capability disabled.

Called when disabling client capability *name* has been acknowledged.

on_welcome()
Successfully signed on to the server.

on_nick_changed(*nick*)
Changed nick.

on_joined(*channel*)
Joined a channel.

on_left(*channel*)
Left a channel.

on_kicked(*channel*, *by*, *reason*)
Kicked from a channel.

on_privmsg(*user*, *to*, *message*)
Received a message, either directly or in a channel.

on_notice(*user*, *to*, *message*)
Received a notice, either directly or in a channel.

on_action(*user*, *to*, *action*)
Received CTCP ACTION. Common enough to deserve its own event.

on_ctcp_query_ACTION(*user*, *to*, *data*)
Turn CTCP ACTION into *on_action()* event.

on_user_renamed(*oldnick*, *newnick*)
User changed nick.

connect()
Connect to the IRC server.

connection_lost(*exc*)
Handle a broken connection by attempting to reconnect.

Won't reconnect if the broken connection was deliberate (i.e. `close()` was called).

connection_made()
Callback for successful connection.

Register with the IRC server.

on_user_joined(*user*, *channel*)
User joined a channel.

read_loop()
Read and dispatch lines until the connection closes.

run(*run_once=False*)
Run the bot, reconnecting when the connection is lost.

on_user_left(*user*, *channel*, *message*)
User left a channel.

on_user_kicked(*user*, *channel*, *by*, *reason*)
User kicked from a channel.

on_user_quit(*user*, *message*)
User disconnected.

on_topic_changed(*user*, *channel*, *topic*)
user changed the topic of *channel* to *topic*.

csbot.irc.**main**()

csbot.plugin module

csbot.plugin.**find_plugins**()
Find available plugins.

Returns a list of discovered plugin classes.

csbot.plugin.**build_plugin_dict**(*plugins*)
Build a dictionary mapping the value of *plugin_name()* to each plugin class in *plugins*. *PluginDuplicate* is raised if more than one plugin has the same name.

class csbot.plugin.**LazyMethod**(*obj*, *name*)
Bases: object

exception csbot.plugin.**PluginDuplicate**
Bases: Exception

exception csbot.plugin.**PluginDependencyUnmet**
Bases: Exception

exception csbot.plugin.**PluginFeatureError**
Bases: Exception

exception csbot.plugin.**PluginConfigError**
Bases: Exception

class csbot.plugin.**PluginManager**(*loaded*, *available*, *plugins*, *args*)
Bases: collections.abc.Mapping

A simple plugin manager and proxy.

The plugin manager is responsible for loading plugins and proxying method calls to all plugins. In addition to accepting *loaded*, a list of existing plugin objects, it will attempt to load each of *plugins* from *available* (a mapping of plugin name to plugin class), passing *args* to the constructors.

Attempting to load missing or duplicate plugins will log errors and warnings respectively, but will not result in an exception or any change of state. A plugin class' dependencies are checked before loading and a *PluginDependencyUnmet* is raised if any are missing.

The Mapping interface is implemented to provide easy querying and access to the loaded plugins. All attributes that do not start with a _ are treated as methods that will be proxied through to every plugin in the order they were loaded (*loaded* before *plugins*) with the same arguments.

plugins = None
Loaded plugins.

class csbot.plugin.**ProvidedByPlugin**(*plugin: str, kwargs: Mapping[str, Any], name: str = None*)
Bases: object

Descriptor for plugin attributes that get (and cache) a value from another plugin.

See *Plugin.use()*.

class csbot.plugin.**PluginMeta**(*name*, *bases*, *attrs*)
Bases: type

Metaclass for *Plugin* that collects methods tagged with plugin feature decorators.

classmethod current()

class csbot.plugin.**Plugin**(*bot*)
Bases: object

Bot plugin base class.

All bot plugins should inherit from this class. It provides convenience methods for hooking events, registering commands, accessing MongoDB and manipulating the configuration file.

CONFIG_DEFAULTS = {}
Default configuration values, used automatically by *config_get()*.

CONFIG_ENVVARS = {}
Configuration environment variables, used automatically by *config_get()*.

PLUGIN_DEPENDS = []
Plugins that *missing_dependencies()* should check for.

log = None
The plugin's logger, created by default using the plugin class' containing module name as the logger name.

classmethod plugin_name()
Get the name of the plugin, by default the class name in lowercase.

classmethod qualified_name()
Get the fully qualified class name, most useful when complaining about duplicate plugins names.

classmethod missing_dependencies(*plugins*)
Return elements from *PLUGIN_DEPENDS* that are not in the container *plugins*.

This should be used with some container of already loaded plugin names (e.g. a dictionary or set) to find out which dependencies are missing.

static hook(*hook*)

static command(*cmd*, ***metadata*)
Tag a command to be registered by *setup()*.

Additional keyword arguments are added to a metadata dictionary that gets stored with the command. This is a good place to put, for example, the help string for the command:

```
@Plugin.command('foo', help='foo: does something amazing')
def foo_command(self, e):
    pass
```

static integrate_with(**otherplugins*)
Tag a method as providing integration with *otherplugins*.

During *setup()*, all methods tagged with this decorator will be run if all of the named plugins are loaded. The actual plugin objects will be passed as arguments to the method in the same order.

Note: The order that integration methods are called in cannot be guaranteed, because attribute order is not preserved during class creation.

---

static use(*other*, ***kwargs*)
Create a property that will be provided by another plugin.

Returns a *ProvidedByPlugin* instance. *PluginMeta* will collect attributes of this type, and add *other* as an implicit plugin dependency. *setup()* will replace it with a value acquired from the plugin named by *other*. For example:

```
class Foo(Plugin):
    stuff = Plugin.use('mongodb', collection='stuff')
```

will cause *setup()* to replace the `stuff` attribute with:

```
self.bot.plugins[other].provide(self.plugin_name(), **kwargs)
```

get_hooks(*hook: str*) → List[Callable]
Get a list of this plugin's handlers for *hook*.

provide(*plugin_name*, ***kwarg*)
Provide a value for a *Plugin.use()* usage.

setup()
Plugin setup.

- Replace all *ProvidedByPlugin* attributes.
- Fire all plugin integration methods.
- Register all commands provided by the plugin.

teardown()
Plugin teardown.

- Unregister all commands provided by the plugin.

config
Get the configuration section for this plugin.

Uses the `[plugin_name]` section of the configuration file, creating an empty section if it doesn't exist.

See also:

configparser

subconfig(*subsection*)
Get a configuration subsection for this plugin.

Uses the `[plugin_name/subsection]` section of the configuration file, creating an empty section if it doesn't exist.

config_get(*key*)
Convenience wrapper proxying `get()` on *config*.

Given a key, this method tries the following in order:

```
self.config[key]
for v in self.CONFIG_ENVVARS[key]:
    os.environ[v]
self.CONFIG_DEFAULTS[key]
```

`KeyError` is raised if none of the methods succeed.

config_getboolean(*key*)
Identical to *config_get()*, but proxying `getboolean`.

class csbot.plugin.**SpecialPlugin**(*bot*)
Bases: *csbot.plugin.Plugin*

A special plugin with a special name that expects to be handled specially. Probably shouldn't have too many of these or they won't feel special anymore.

classmethod plugin_name()
Change the plugin name to something that can't possibly result from a class name by prepending a `@`.

csbot.util module

csbot.util.**nick**(*user*)
Get nick from user string.

```
>>> nick('csyorkbot!~csbot@example.com')
'csyorkbot'
```

csbot.util.**username**(*user*)
Get username from user string.

```
>>> username('csyorkbot!~csbot@example.com')
'csbot'
```

csbot.util.**host**(*user*)
Get hostname from user string.

```
>>> host('csyorkbot!~csbot@example.com')
'example.com'
```

csbot.util.**is_channel**(*channel*)
Check if *channel* is a channel or private chat.

```
>>> is_channel('#cs-york')
True
>>> is_channel('csyorkbot')
False
```

csbot.util.**parse_arguments**(*raw*)
Parse *raw* into a list of arguments using `shlex`.

The `shlex` lexer is customised to be more appropriate for grouping natural language arguments by only treating `"` as a quote character. This allows `'` to be used naturally. A `ValueError` will be raised if the string couldn't be parsed.

```
>>> parse_arguments("a test string")
['a', 'test', 'string']
>>> parse_arguments("apostrophes aren't a problem")
```

(continues on next page)

(continued from previous page)

```
['apostrophes', "aren't", 'a', 'problem']
>>> parse_arguments('"string grouping" is useful')
['string grouping', 'is', 'useful']
>>> parse_arguments('just remember to "match your quotes')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: No closing quotation
```

csbot.util.**simple_http_get**(*url*, *stream=False*)

A deliberately dumb wrapper around `requests.get()`.

This should be used for the vast majority of HTTP GET requests. It turns off SSL certificate verification and sets a non-default User-Agent, thereby succeeding at most "just get the content" requests. Note that it can generate a ConnectionError exception if the url is not resolvable.

stream controls the "streaming mode" of the HTTP client, i.e. deferring the acquisition of the response body. Use this if you need to impose a maximum size or process a large response. *The entire content must be consumed or ''response.close()'' must be called.*

csbot.util.**pairwise**(*iterable*)

Pairs elements of an iterable together, e.g. s -> (s0,s1), (s1,s2), (s2, s3), ...

csbot.util.**cap_string**(*s*, *n*)

If a string is longer than a particular length, it gets truncated and has '...' added to the end.

csbot.util.**ordinal**(*value*)

Converts zero or a *postive* integer (or their string representations) to an ordinal value.

http://code.activestate.com/recipes/576888-format-a-number-as-an-ordinal/

```
>>> for i in range(1,13):
...     ordinal(i)
...
u'1st'
u'2nd'
u'3rd'
u'4th'
u'5th'
u'6th'
u'7th'
u'8th'
u'9th'
u'10th'
u'11th'
u'12th'
```

```
>>> for i in (100, '111', '112',1011):
...     ordinal(i)
...
u'100th'
u'111th'
u'112th'
u'1011th'
```

csbot.util.**pluralize**(*n*, *singular*, *plural*)

csbot.util.**is_ascii**(*s*)

Returns true if all characters in a string can be represented in ASCII.

csbot.util.**maybe_future**(*result*, *, *on_error=None*, *log=<Logger csbot.util (WARNING)>*, *loop=None*)
Make *result* a future if possible, otherwise return None.

If *result* is not None but also not awaitable, it is passed to *on_error* if supplied, otherwise logged as a warning on *log*.

csbot.util.**truncate_utf8**(*b: bytes*, *maxlen: int*, *ellipsis: bytes = b'...'*) → bytes
Trim *b* to a maximum of *maxlen* bytes (including *ellipsis* if longer), without breaking UTF-8 sequences.

csbot.util.**topological_sort**(*data: Dict[T, Set[T]]*) → Iterator[Set[T]]
Get topological ordering from dependency data.

Generates sets of items with equal ordering position.

class csbot.util.**RateLimited**(*f*, *, *period: float = 2.0*, *count: int = 5*, *loop=None*, *log=<Logger csbot.util (WARNING)>*)
Bases: object

An asynchronous wrapper around calling *f* that is rate limited to *count* calls per *period* seconds.

Calling the rate limiter returns a future that completes with the result of calling *f* with the same arguments. *start()* and *stop()* control whether or not calls are actually processed.

get_delay() → float
Get number of seconds to wait before processing the next call.

start()
Start async task to process calls.

stop(*clear=True*)
Stop async call processing.

If *clear* is True (the default), any pending calls not yet processed have their futures cancelled. If it's False, then those pending calls will still be queued when *start()* is called again.

Returns list of `(args, kwargs)` pairs of cancelled calls.

run()

csbot.util.**type_validator**(*_obj*, *attrib: attr._make.Attribute*, *value*)
An attrs validator that inspects the attribute type.

class csbot.util.**PrettyStreamHandler**(*stream=None*, *colour=None*)
Bases: logging.StreamHandler

Wrap log messages with severity-dependent ANSI terminal colours.

Use in place of logging.StreamHandler to have log messages coloured according to severity.

```
>>> handler = PrettyStreamHandler()
>>> handler.setFormatter(logging.Formatter('[%(levelname)-8s] %(message)s'))
>>> logging.getLogger('').addHandler(handler)
```

stream corresponds to the same argument to logging.StreamHandler, defaulting to stderr.

colour overrides TTY detection to force colour on or off.

This source for this class is released into the public domain.

Code author: Alan Briolat <alan.briolat@gmail.com>

COLOURS = {10: '\x1b[36m', 30: '\x1b[33m', 40: '\x1b[31m', 50: '\x1b[31;7m'}
Mapping from logging levels to ANSI colours.

COLOUR_END = '\x1b[0m'
ANSI code for resetting the terminal to default colour.

format(*record*)
Get a coloured, formatted message for a log record.

Calls `logging.StreamHandler.format()` and applies a colour to the message if appropriate.

csbot.util.**maybe_future_result**(*result*, ***kwargs*)
Get actual result from *result*.

If *result* is awaitable, return the result of awaiting it, otherwise just return *result*.

csbot.util.**simple_http_get_async**(*url*, ***kwargs*)

3.1.3 Module contents

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

csbot, 43
csbot.cli, 25
csbot.config, 25
csbot.core, 27
csbot.events, 29
csbot.irc, 31
csbot.plugin, 37
csbot.plugins, 25
csbot.plugins.auth, 9
csbot.plugins.calc, 10
csbot.plugins.cron, 11
csbot.plugins.csyork, 13
csbot.plugins.github, 13
csbot.plugins.helix, 15
csbot.plugins.hoogle, 16
csbot.plugins.imgur, 16
csbot.plugins.last, 16
csbot.plugins.linkinfo, 17
csbot.plugins.logger, 19
csbot.plugins.mongodb, 19
csbot.plugins.termdates, 20
csbot.plugins.topic, 20
csbot.plugins.usertrack, 21
csbot.plugins.webhook, 21
csbot.plugins.webserver, 22
csbot.plugins.whois, 23
csbot.plugins.xkcd, 24
csbot.plugins.youtube, 24
csbot.util, 40

Index

A

account_command() (*csbot.plugins.usertrack.UserTrack method*), 21
act() (*csbot.irc.IRCClient method*), 34
action() (*csbot.plugins.logger.Logger method*), 19
after() (*csbot.plugins.cron.PluginCron method*), 12
all_hail_our_google_overlords() (*csbot.plugins.youtube.Youtube method*), 25
arguments() (*csbot.events.CommandEvent method*), 30
ask_the_almighty_helix() (*csbot.plugins.helix.Helix method*), 16
at() (*csbot.plugins.cron.PluginCron method*), 12
Auth (*class in csbot.plugins.auth*), 9
auth_method (*csbot.core.Bot.Config attribute*), 27
available_capabilities (*csbot.irc.IRCClient attribute*), 33
available_plugins (*csbot.core.Bot attribute*), 27

B

bind_addr (*csbot.core.Bot.Config attribute*), 27
Bot (*class in csbot.core*), 27
bot (*csbot.events.Event attribute*), 30
Bot.Config (*class in csbot.core*), 27
bot_setup() (*csbot.core.Bot method*), 28
bot_teardown() (*csbot.core.Bot method*), 28
build_plugin_dict() (*in module csbot.plugin*), 37

C

Calc (*class in csbot.plugins.calc*), 10
CalcError, 10
CalcEval (*class in csbot.plugins.calc*), 10
cap_string() (*in module csbot.util*), 41
channels (*csbot.core.Bot.Config attribute*), 27
check() (*csbot.plugins.auth.Auth method*), 10
check() (*csbot.plugins.auth.PermissionDB method*), 9
check_or_error() (*csbot.plugins.auth.Auth method*), 10
client_ping (*csbot.core.Bot.Config attribute*), 27
CMD_RESPONSE (*csbot.plugins.youtube.Youtube attribute*), 25
codec (*csbot.irc.IRCClient attribute*), 33
COLOUR_END (*csbot.util.PrettyStreamHandler attribute*), 42
COLOURS (*csbot.util.PrettyStreamHandler attribute*), 42
command (*csbot.irc.IRCMessage attribute*), 31
command() (*csbot.plugin.Plugin static method*), 38
command() (*csbot.plugins.logger.Logger method*), 19
command_name (*csbot.irc.IRCMessage attribute*), 31
command_prefix (*csbot.core.Bot.Config attribute*), 27
CommandEvent (*class in csbot.events*), 30
Config (*class in csbot.config*), 25
config (*csbot.plugin.Plugin attribute*), 39
CONFIG_DEFAULTS (*csbot.plugin.Plugin attribute*), 38
CONFIG_DEFAULTS (*csbot.plugins.github.GitHub attribute*), 15
CONFIG_DEFAULTS (*csbot.plugins.hoogle.Hoogle attribute*), 16
CONFIG_DEFAULTS (*csbot.plugins.imgur.Imgur attribute*), 16
CONFIG_DEFAULTS (*csbot.plugins.mongodb.MongoDB attribute*), 19
CONFIG_DEFAULTS (*csbot.plugins.topic.Topic attribute*), 20
CONFIG_DEFAULTS (*csbot.plugins.webhook.Webhook attribute*), 22
CONFIG_DEFAULTS (*csbot.plugins.webserver.WebServer attribute*), 23
CONFIG_DEFAULTS (*csbot.plugins.youtube.Youtube attribute*), 24
CONFIG_ENVVARS (*csbot.plugin.Plugin attribute*), 38
CONFIG_ENVVARS (*csbot.plugins.github.GitHub attribute*), 15
CONFIG_ENVVARS (*csbot.plugins.imgur.Imgur attribute*), 16
CONFIG_ENVVARS (*csbot.plugins.mongodb.MongoDB*

attribute), 19
CONFIG_ENVVARS (*csbot.plugins.webhook.Webhook attribute*), 22
CONFIG_ENVVARS (*csbot.plugins.youtube.Youtube attribute*), 24
config_get() (*csbot.plugin.Plugin method*), 39
config_get() (*csbot.plugins.github.GitHub method*), 15
config_get() (*csbot.plugins.topic.Topic method*), 21
config_getboolean() (*csbot.plugin.Plugin method*), 40
ConfigError (*in module csbot.config*), 25
connect() (*csbot.irc.IRCClient method*), 36
connected() (*csbot.plugins.logger.Logger method*), 19
connection_lost() (*csbot.core.Bot method*), 29
connection_lost() (*csbot.irc.IRCClient method*), 36
connection_made() (*csbot.core.Bot method*), 29
connection_made() (*csbot.irc.IRCClient method*), 36
convert() (*csbot.config.WordList method*), 25
copy_or_create() (*csbot.plugins.usertrack.UserDict method*), 21
create() (*csbot.irc.IRCMessage class method*), 31
create_app() (*csbot.plugins.webhook.Webhook method*), 22
create_subapp() (*csbot.plugins.webserver.WebServer method*), 23
create_user() (*csbot.plugins.usertrack.UserDict static method*), 21
Cron (*class in csbot.plugins.cron*), 11
csbot (*module*), 43
csbot.cli (*module*), 25
csbot.config (*module*), 25
csbot.core (*module*), 27
csbot.events (*module*), 29
csbot.irc (*module*), 31
csbot.plugin (*module*), 37
csbot.plugins (*module*), 25
csbot.plugins.auth (*module*), 9
csbot.plugins.calc (*module*), 10
csbot.plugins.cron (*module*), 11
csbot.plugins.csyork (*module*), 13
csbot.plugins.github (*module*), 13
csbot.plugins.helix (*module*), 15
csbot.plugins.hoogle (*module*), 16
csbot.plugins.imgur (*module*), 16
csbot.plugins.last (*module*), 16
csbot.plugins.linkinfo (*module*), 17
csbot.plugins.logger (*module*), 19
csbot.plugins.mongodb (*module*), 19
csbot.plugins.termdates (*module*), 20
csbot.plugins.topic (*module*), 20
csbot.plugins.usertrack (*module*), 21
csbot.plugins.webhook (*module*), 21
csbot.plugins.webserver (*module*), 22
csbot.plugins.whois (*module*), 23
csbot.plugins.xkcd (*module*), 24
csbot.plugins.youtube (*module*), 24
csbot.util (*module*), 40
CSYork (*class in csbot.plugins.csyork*), 13
ctcp_query() (*csbot.irc.IRCClient method*), 35
ctcp_reply() (*csbot.irc.IRCClient method*), 35
current() (*csbot.plugin.PluginMeta class method*), 38

D

DATE_FORMAT (*csbot.plugins.termdates.TermDates attribute*), 20
datetime (*csbot.events.Event attribute*), 30
db (*csbot.plugins.last.Last attribute*), 17
db_terms (*csbot.plugins.termdates.TermDates attribute*), 20
decode() (*csbot.irc.IRCCodec method*), 32
DEFAULTS() (*csbot.irc.IRCClient static method*), 33
disconnect() (*csbot.irc.IRCClient method*), 33
disconnected() (*csbot.plugins.logger.Logger method*), 19
do_some_calc() (*csbot.plugins.calc.Calc method*), 10
dump() (*in module csbot.config*), 26
dumps() (*in module csbot.config*), 26
DuplicateTaskError, 12

E

emit() (*csbot.core.Bot method*), 28
emit_new() (*csbot.core.Bot method*), 28
enabled_capabilities (*csbot.irc.IRCClient attribute*), 33
encode() (*csbot.irc.IRCCodec method*), 32
Event (*class in csbot.events*), 30
event_runner() (*csbot.plugins.cron.Cron method*), 12
event_type (*csbot.events.Event attribute*), 30
every() (*csbot.plugins.cron.PluginCron method*), 12
example_mode() (*in module csbot.config*), 25
extend() (*csbot.events.Event class method*), 30

F

find_by_matchers() (*csbot.plugins.github.GitHub class method*), 15
find_plugins() (*in module csbot.plugin*), 37
fire_command() (*csbot.core.Bot method*), 29
fire_event() (*csbot.plugins.cron.Cron method*), 11
first_monday (*csbot.plugins.termdates.Term attribute*), 20

fix_json_unicode() (*in module csbot.plugins.xkcd*), 24
FORCE_TRAILING (*csbot.irc.IRCMessage attribute*), 31
format() (*csbot.util.PrettyStreamHandler method*), 43

G

generate() (*csbot.config.TomlExampleGenerator method*), 27
generate_toml_example() (*in module csbot.config*), 27
generic_handler() (*csbot.plugins.github.GitHub method*), 15
generic_visit() (*csbot.plugins.calc.CalcEval method*), 10
get_delay() (*csbot.util.RateLimited method*), 42
get_field() (*csbot.plugins.github.MessageFormatter method*), 15
get_hooks() (*csbot.plugin.Plugin method*), 39
get_info() (*in module csbot.plugins.xkcd*), 24
get_link_info() (*csbot.plugins.linkinfo.LinkInfo method*), 18
get_message() (*csbot.plugins.linkinfo.LinkInfoResult method*), 17
get_permissions() (*csbot.plugins.auth.PermissionDB method*), 9
get_topic() (*csbot.irc.IRCClient method*), 34
get_user() (*csbot.plugins.usertrack.UserTrack method*), 21
get_video_json() (*csbot.plugins.youtube.Youtube method*), 25
get_week_number() (*csbot.plugins.termdates.Term method*), 20
get_week_start() (*csbot.plugins.termdates.Term method*), 20
get_yt_id() (*in module csbot.plugins.youtube*), 24
GitHub (*class in csbot.plugins.github*), 15
github_report_deploy() (*in module csbot.cli*), 25
guarded_factorial() (*in module csbot.plugins.calc*), 10
guarded_lshift() (*in module csbot.plugins.calc*), 10
guarded_power() (*in module csbot.plugins.calc*), 10
guarded_rshift() (*in module csbot.plugins.calc*), 10

H

handle_pull_request() (*csbot.plugins.github.GitHub method*), 15
handle_pull_request_review() (*csbot.plugins.github.GitHub method*), 15
handle_push() (*csbot.plugins.github.GitHub method*), 15
Helix (*class in csbot.plugins.helix*), 15
Hoogle (*class in csbot.plugins.hoogle*), 16
hook() (*csbot.plugin.Plugin static method*), 38
host (*csbot.irc.IRCUser attribute*), 32
host() (*in module csbot.util*), 40
HybridEventRunner (*class in csbot.events*), 29

I

identify() (*csbot.core.Bot method*), 29
identify_user() (*csbot.plugins.whois.Whois method*), 24
Imgur (*class in csbot.plugins.imgur*), 16
ImgurError, 16
initialised (*csbot.plugins.termdates.TermDates attribute*), 20
integrate_with() (*csbot.plugin.Plugin static method*), 38
integrate_with_linkinfo() (*csbot.plugins.imgur.Imgur method*), 16
irc_354() (*csbot.core.Bot method*), 29
irc_ACCOUNT() (*csbot.core.Bot method*), 29
irc_CAP() (*csbot.irc.IRCClient method*), 35
irc_CAP_ACK() (*csbot.irc.IRCClient method*), 35
irc_CAP_LS() (*csbot.irc.IRCClient method*), 35
irc_CAP_NAK() (*csbot.irc.IRCClient method*), 35
irc_ERR_NICKNAMEINUSE() (*csbot.irc.IRCClient method*), 35
irc_host (*csbot.core.Bot.Config attribute*), 27
irc_JOIN() (*csbot.core.Bot method*), 29
irc_JOIN() (*csbot.irc.IRCClient method*), 35
irc_KICK() (*csbot.irc.IRCClient method*), 35
irc_NICK() (*csbot.irc.IRCClient method*), 35
irc_NOTICE() (*csbot.irc.IRCClient method*), 35
irc_PART() (*csbot.irc.IRCClient method*), 35
irc_PING() (*csbot.irc.IRCClient method*), 35
irc_port (*csbot.core.Bot.Config attribute*), 27
irc_PRIVMSG() (*csbot.irc.IRCClient method*), 35
irc_QUIT() (*csbot.irc.IRCClient method*), 35
irc_RPL_ENDOFNAMES() (*csbot.core.Bot method*), 29
irc_RPL_NAMREPLY() (*csbot.core.Bot method*), 29
irc_RPL_TOPIC() (*csbot.irc.IRCClient method*), 35
irc_RPL_WELCOME() (*csbot.irc.IRCClient method*), 35
irc_TOPIC() (*csbot.irc.IRCClient method*), 35
IRCClient (*class in csbot.irc*), 33
IRCClient.Waiter (*class in csbot.irc*), 34
IRCClientError, 33
IRCCodec (*class in csbot.irc*), 32
IRCMessage (*class in csbot.irc*), 31
IRCParseError, 31
IRCUser (*class in csbot.irc*), 32

ircv3 (*csbot.core.Bot.Config attribute*), 27
is_allowable_type() (*in module csbot.config*), 26
is_ascii() (*in module csbot.util*), 41
is_channel() (*in module csbot.util*), 40
is_config() (*in module csbot.config*), 25
is_error (*csbot.plugins.linkinfo.LinkInfoResult attribute*), 17
is_redundant (*csbot.plugins.linkinfo.LinkInfoResult attribute*), 17
is_too_long() (*in module csbot.plugins.calc*), 10

J

join() (*csbot.irc.IRCClient method*), 34
joined() (*csbot.plugins.logger.Logger method*), 19

L

Last (*class in csbot.plugins.last*), 16
last() (*csbot.plugins.last.Last method*), 17
last_action() (*csbot.plugins.last.Last method*), 17
last_command() (*csbot.plugins.last.Last method*), 17
last_friday (*csbot.plugins.termdates.Term attribute*), 20
last_message() (*csbot.plugins.last.Last method*), 17
LazyMethod (*class in csbot.plugin*), 37
leave() (*csbot.irc.IRCClient method*), 34
left() (*csbot.plugins.logger.Logger method*), 19
line_received() (*csbot.core.Bot method*), 28
line_received() (*csbot.irc.IRCClient method*), 33
line_sent() (*csbot.core.Bot method*), 28
line_sent() (*csbot.irc.IRCClient method*), 33
link_command() (*csbot.plugins.linkinfo.LinkInfo method*), 18
LinkInfo (*class in csbot.plugins.linkinfo*), 18
LinkInfo.Config (*class in csbot.plugins.linkinfo*), 18
linkinfo_integrate() (*csbot.plugins.xkcd.xkcd method*), 24
linkinfo_integrate() (*csbot.plugins.youtube.Youtube method*), 25
LinkInfoHandler (*class in csbot.plugins.linkinfo*), 17
LinkInfoResult (*class in csbot.plugins.linkinfo*), 17
load() (*in module csbot.config*), 26
load_ini() (*in module csbot.cli*), 25
load_json() (*in module csbot.cli*), 25
load_toml() (*in module csbot.cli*), 25
loads() (*in module csbot.config*), 26
log (*csbot.plugin.Plugin attribute*), 38
Logger (*class in csbot.plugins.logger*), 19

M

main() (*in module csbot.irc*), 37
make_example() (*in module csbot.config*), 27
match_task() (*csbot.plugins.cron.Cron method*), 11
max_file_ext_length (*csbot.plugins.linkinfo.LinkInfo.Config attribute*), 18
max_response_size (*csbot.plugins.linkinfo.LinkInfo.Config attribute*), 18
maybe_future() (*in module csbot.util*), 41
maybe_future_result() (*in module csbot.util*), 43
message_received() (*csbot.irc.IRCClient method*), 33
MessageFormatter (*class in csbot.plugins.github*), 15
MESSAGES (*csbot.config.WordList attribute*), 25
minimum_path_match (*csbot.plugins.linkinfo.LinkInfo.Config attribute*), 18
minimum_slug_length (*csbot.plugins.linkinfo.LinkInfo.Config attribute*), 18
missing_dependencies() (*csbot.plugin.Plugin class method*), 38
MongoDB (*class in csbot.plugins.mongodb*), 19
msg() (*csbot.irc.IRCClient method*), 34

N

names() (*csbot.plugins.logger.Logger method*), 19
nick (*csbot.irc.IRCUser attribute*), 32
nick() (*in module csbot.util*), 40
nickname (*csbot.core.Bot.Config attribute*), 27
notice() (*csbot.irc.IRCClient method*), 34
notice() (*csbot.plugins.logger.Logger method*), 19
nsfw (*csbot.plugins.linkinfo.LinkInfoResult attribute*), 17

O

on_action() (*csbot.core.Bot method*), 28
on_action() (*csbot.irc.IRCClient method*), 36
on_capabilities_available() (*csbot.irc.IRCClient method*), 36
on_capability_disabled() (*csbot.irc.IRCClient method*), 36
on_capability_enabled() (*csbot.irc.IRCClient method*), 36
on_ctcp_query_ACTION() (*csbot.irc.IRCClient method*), 36
on_joined() (*csbot.core.Bot method*), 28
on_joined() (*csbot.irc.IRCClient method*), 36
on_kicked() (*csbot.irc.IRCClient method*), 36
on_left() (*csbot.core.Bot method*), 28
on_left() (*csbot.irc.IRCClient method*), 36
on_names() (*csbot.core.Bot method*), 29
on_nick_changed() (*csbot.irc.IRCClient method*), 36
on_notice() (*csbot.core.Bot method*), 28

on_notice() (*csbot.irc.IRCClient method*), 36
on_privmsg() (*csbot.core.Bot method*), 28
on_privmsg() (*csbot.irc.IRCClient method*), 36
on_topic_changed() (*csbot.core.Bot method*), 29
on_topic_changed() (*csbot.irc.IRCClient method*), 37
on_user_identified() (*csbot.core.Bot method*), 29
on_user_joined() (*csbot.core.Bot method*), 28
on_user_joined() (*csbot.irc.IRCClient method*), 36
on_user_kicked() (*csbot.irc.IRCClient method*), 37
on_user_left() (*csbot.core.Bot method*), 28
on_user_left() (*csbot.irc.IRCClient method*), 37
on_user_quit() (*csbot.core.Bot method*), 29
on_user_quit() (*csbot.irc.IRCClient method*), 37
on_user_renamed() (*csbot.core.Bot method*), 29
on_user_renamed() (*csbot.irc.IRCClient method*), 36
on_welcome() (*csbot.core.Bot method*), 28
on_welcome() (*csbot.irc.IRCClient method*), 36
option() (*in module csbot.config*), 26
option_list() (*in module csbot.config*), 26
option_map() (*in module csbot.config*), 26
ordinal() (*in module csbot.util*), 41
outcomes (*csbot.plugins.helix.Helix attribute*), 16

P

pad_params() (*csbot.irc.IRCMessage method*), 31
pairwise() (*in module csbot.util*), 41
params (*csbot.irc.IRCMessage attribute*), 31
parse() (*csbot.irc.IRCMessage class method*), 31
parse() (*csbot.irc.IRCUser class method*), 32
parse_arguments() (*in module csbot.util*), 40
parse_command() (*csbot.events.CommandEvent class method*), 30
password (*csbot.core.Bot.Config attribute*), 27
PermissionDB (*class in csbot.plugins.auth*), 9
Plugin (*class in csbot.plugin*), 38
PLUGIN_DEPENDS (*csbot.plugin.Plugin attribute*), 38
PLUGIN_DEPENDS (*csbot.plugins.auth.Auth attribute*), 9
PLUGIN_DEPENDS (*csbot.plugins.github.GitHub attribute*), 15
PLUGIN_DEPENDS (*csbot.plugins.topic.Topic attribute*), 20
PLUGIN_DEPENDS (*csbot.plugins.whois.Whois attribute*), 23
plugin_name() (*csbot.plugin.Plugin class method*), 38
plugin_name() (*csbot.plugin.SpecialPlugin class method*), 40
PluginConfigError, 37
PluginCron (*class in csbot.plugins.cron*), 12
PluginDependencyUnmet, 37
PluginDuplicate, 37
PluginError, 27
PluginFeatureError, 37
PluginManager (*class in csbot.plugin*), 37
PluginMeta (*class in csbot.plugin*), 38
plugins (*csbot.core.Bot.Config attribute*), 27
plugins (*csbot.plugin.PluginManager attribute*), 37
pluralize() (*in module csbot.util*), 41
post_event() (*csbot.core.Bot method*), 28
post_event() (*csbot.events.HybridEventRunner method*), 30
PredicateType (*csbot.irc.IRCClient.Waiter attribute*), 34
prefix (*csbot.irc.IRCMessage attribute*), 31
pretty (*csbot.irc.IRCMessage attribute*), 31
pretty_log (*csbot.plugins.logger.Logger attribute*), 19
PrettyStreamHandler (*class in csbot.util*), 42
privmsg() (*csbot.core.Bot method*), 28
privmsg() (*csbot.plugins.logger.Logger method*), 19
process() (*csbot.plugins.auth.PermissionDB method*), 9
process_wait_for_message() (*csbot.irc.IRCClient method*), 34
provide() (*csbot.plugin.Plugin method*), 39
provide() (*csbot.plugins.cron.Cron method*), 11
provide() (*csbot.plugins.mongodb.MongoDB method*), 19
ProvidedByPlugin (*class in csbot.plugin*), 38

Q

qualified_name() (*csbot.plugin.Plugin class method*), 38
quit() (*csbot.irc.IRCClient method*), 34
quit() (*csbot.plugins.logger.Logger method*), 19

R

randall_is_awesome() (*csbot.plugins.xkcd.xkcd method*), 24
rate_limit_count (*csbot.core.Bot.Config attribute*), 27
rate_limit_count (*csbot.plugins.linkinfo.LinkInfo.Config attribute*), 18
rate_limit_period (*csbot.core.Bot.Config attribute*), 27
rate_limit_time (*csbot.plugins.linkinfo.LinkInfo.Config attribute*), 18
RateLimited (*class in csbot.util*), 42
raw (*csbot.irc.IRCMessage attribute*), 31
raw (*csbot.irc.IRCUser attribute*), 32
raw_log (*csbot.plugins.logger.Logger attribute*), 19

raw_received() (*csbot.plugins.logger.Logger method*), 19
raw_sent() (*csbot.plugins.logger.Logger method*), 19
read_loop() (*csbot.irc.IRCClient method*), 36
realname (*csbot.core.Bot.Config attribute*), 27
recent_messages (*csbot.core.Bot attribute*), 28
record() (*csbot.plugins.last.Last method*), 17
record_action() (*csbot.plugins.last.Last method*), 17
record_command() (*csbot.plugins.last.Last method*), 17
record_message() (*csbot.plugins.last.Last method*), 17
REGEX (*csbot.irc.IRCMessage attribute*), 31
REGEX (*csbot.irc.IRCUser attribute*), 32
register_command() (*csbot.core.Bot method*), 28
register_exclude() (*csbot.plugins.linkinfo.LinkInfo method*), 18
register_handler() (*csbot.plugins.linkinfo.LinkInfo method*), 18
renamed() (*csbot.plugins.logger.Logger method*), 19
reply() (*csbot.core.Bot method*), 29
reply() (*csbot.events.Event method*), 30
request_capabilities() (*csbot.irc.IRCClient method*), 34
request_handler() (*csbot.plugins.webhook.Webhook method*), 22
respond() (*csbot.plugins.csyork.CSYork method*), 13
RESPONSE (*csbot.plugins.youtube.Youtube attribute*), 25
rollbar_report_deploy() (*in module csbot.cli*), 25
run() (*csbot.irc.IRCClient method*), 37
run() (*csbot.util.RateLimited method*), 42

S

scan_limit (*csbot.plugins.linkinfo.LinkInfo.Config attribute*), 18
scan_privmsg() (*csbot.plugins.linkinfo.LinkInfo method*), 18
schedule() (*csbot.plugins.cron.Cron method*), 11
schedule() (*csbot.plugins.cron.PluginCron method*), 12
schedule_event_runner() (*csbot.plugins.cron.Cron method*), 12
scrape_html_title() (*csbot.plugins.linkinfo.LinkInfo method*), 18
search_hoogle() (*csbot.plugins.hoogle.Hoogle method*), 16
send() (*csbot.irc.IRCClient method*), 34
send_line() (*csbot.irc.IRCClient method*), 34
set() (*csbot.plugins.whois.Whois method*), 24
set_nick() (*csbot.irc.IRCClient method*), 34
set_topic() (*csbot.irc.IRCClient method*), 34
setdefault() (*csbot.plugins.whois.Whois method*), 24
setup() (*csbot.plugin.Plugin method*), 39
setup() (*csbot.plugins.auth.Auth method*), 10
setup() (*csbot.plugins.cron.Cron method*), 11
setup() (*csbot.plugins.helix.Helix method*), 16
setup() (*csbot.plugins.hoogle.Hoogle method*), 16
setup() (*csbot.plugins.termdates.TermDates method*), 20
setup() (*csbot.plugins.topic.Topic method*), 20
setup() (*csbot.plugins.usertrack.UserTrack method*), 21
setup() (*csbot.plugins.webserver.WebServer method*), 23
show_commands() (*csbot.core.Bot method*), 28
show_plugins() (*csbot.core.Bot method*), 28
show_seen() (*csbot.plugins.last.Last method*), 17
signedOn() (*csbot.core.Bot method*), 28
signedon() (*csbot.plugins.logger.Logger method*), 19
simple_http_get() (*in module csbot.util*), 41
simple_http_get_async() (*in module csbot.util*), 43
SpecialPlugin (*class in csbot.plugin*), 40
start() (*csbot.util.RateLimited method*), 42
stop() (*csbot.util.RateLimited method*), 42
structure() (*in module csbot.config*), 26
subconfig() (*csbot.plugin.Plugin method*), 39

T

tasks (*csbot.plugins.cron.Cron attribute*), 11
teardown() (*csbot.plugin.Plugin method*), 39
teardown() (*csbot.plugins.cron.Cron method*), 11
teardown() (*csbot.plugins.webserver.WebServer method*), 23
Term (*class in csbot.plugins.termdates*), 20
TERM_KEYS (*csbot.plugins.termdates.TermDates attribute*), 20
TermDates (*class in csbot.plugins.termdates*), 20
termdates() (*csbot.plugins.termdates.TermDates method*), 20
termdates_set() (*csbot.plugins.termdates.TermDates method*), 20
terms (*csbot.plugins.termdates.TermDates attribute*), 20
text (*csbot.plugins.linkinfo.LinkInfoResult attribute*), 17
TomlExampleGenerator (*class in csbot.config*), 27
Topic (*class in csbot.plugins.topic*), 20
topic() (*csbot.plugins.logger.Logger method*), 19
topic() (*csbot.plugins.topic.Topic method*), 21
topic_append() (*csbot.plugins.topic.Topic method*), 21

topic_changed() (*csbot.plugins.topic.Topic method*), 21
topic_history() (*csbot.plugins.topic.Topic method*), 21
topic_insert() (*csbot.plugins.topic.Topic method*), 21
topic_pop() (*csbot.plugins.topic.Topic method*), 21
topic_replace() (*csbot.plugins.topic.Topic method*), 21
topic_undo() (*csbot.plugins.topic.Topic method*), 21
topological_sort() (*in module csbot.util*), 42
truncate_utf8() (*in module csbot.util*), 42
type_validator() (*in module csbot.util*), 42

U

unregister_command() (*csbot.core.Bot method*), 28
unregister_commands() (*csbot.core.Bot method*), 28
unschedule() (*csbot.plugins.cron.Cron method*), 12
unschedule() (*csbot.plugins.cron.PluginCron method*), 13
unschedule_all() (*csbot.plugins.cron.PluginCron method*), 13
unset() (*csbot.plugins.whois.Whois method*), 24
unsetdefault() (*csbot.plugins.whois.Whois method*), 24
unstructure() (*in module csbot.config*), 26
url (*csbot.plugins.linkinfo.LinkInfoResult attribute*), 17
use() (*csbot.plugin.Plugin static method*), 39
use_notice (*csbot.core.Bot.Config attribute*), 27
user (*csbot.irc.IRCUser attribute*), 32
user_joined() (*csbot.plugins.logger.Logger method*), 19
user_left() (*csbot.plugins.logger.Logger method*), 19
UserDict (*class in csbot.plugins.usertrack*), 21
username (*csbot.core.Bot.Config attribute*), 27
username() (*in module csbot.util*), 40
UserTrack (*class in csbot.plugins.usertrack*), 21

V

visit_BinOp() (*csbot.plugins.calc.CalcEval method*), 10
visit_Call() (*csbot.plugins.calc.CalcEval method*), 10
visit_Compare() (*csbot.plugins.calc.CalcEval method*), 10
visit_Expr() (*csbot.plugins.calc.CalcEval method*), 10
visit_Module() (*csbot.plugins.calc.CalcEval method*), 10
visit_Name() (*csbot.plugins.calc.CalcEval method*), 10
visit_NameConstant() (*csbot.plugins.calc.CalcEval method*), 10
visit_Num() (*csbot.plugins.calc.CalcEval method*), 10
visit_Str() (*csbot.plugins.calc.CalcEval method*), 10
visit_UnaryOp() (*csbot.plugins.calc.CalcEval method*), 10

W

wait_for_message() (*csbot.irc.IRCClient method*), 34
Webhook (*class in csbot.plugins.webhook*), 22
webhook() (*csbot.plugins.github.GitHub method*), 15
WebServer (*class in csbot.plugins.webserver*), 23
week() (*csbot.plugins.termdates.TermDates method*), 20
Whois (*class in csbot.plugins.whois*), 23
whois() (*csbot.plugins.whois.Whois method*), 24
whois_lookup() (*csbot.plugins.whois.Whois method*), 23
whois_set() (*csbot.plugins.whois.Whois method*), 23
whois_unset() (*csbot.plugins.whois.Whois method*), 24
whoisdb (*csbot.plugins.whois.Whois attribute*), 23
WordList (*class in csbot.config*), 25
write_example_config() (*csbot.core.Bot class method*), 29

X

xkcd (*class in csbot.plugins.xkcd*), 24
xkcd.XKCDError, 24

Y

Youtube (*class in csbot.plugins.youtube*), 24
YoutubeError, 24