latest

Release 0.2.6

August 19, 2015

Contents

Created by Stephen McDonald

Gnotty ties the knot between the web and IRC. It is designed to assist open source projects that host an IRC channel for collaboration on their project. Gnotty is BSD licensed.

Gnotty is comprised of several parts. Primarily Gnotty provides a modern web client and server for communicating with an IRC channel via a web browser. The web server uses gevent and WebSockets, which provides the communication layer between the IRC channel and the web browser. Twitter's Bootstrap is used to style the web interface, providing a fully responsive layout, suitable for use with mobile devices. Customisable templates are also provided for skinning the web interface.

Check out the Gnotty live demo to see the web interface in action.

Secondly, Gnotty provides the ability to run a highly customisable IRC bot. Different classes of bots can be configured on startup, and bots can perform different services such as message logging and interacting with users in the IRC channel. Bots also contain webhooks, which allows bots to receive and act on input over HTTP from external services.

Gnotty also provides an optional Django application that archives IRC messages, for browsing and searching via a web interface. By default the IRC bot uses Python's logging module to provide configurable logging handlers for IRC messages. When the Django application is used, a logging handler is added that logs all IRC messages to the Django project's database. The Django application then provides all the necessary views and templates for messages to be searched by keyword, or browsed by date using a calendar interface.

Note that the Django application is entirely optional. Gnotty can be run without using Django at all, as a stand-alone gevent web server that provides the web interface to an IRC channel, with configurable IRC bots.

CHAPTER 1

Installation

The easiest way to install Gnotty is directly from PyPi using pip by running the command below:

```
$ pip install -U gnotty
```

Otherwise you can obtain Gnotty from the GitHub or Bitbucket repositories, and install it directly from source:

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

CHAPTER 2

Configuration

Gnotty is configured via a handful of settings. When integrated with Django, these settings can be defined in your Django project's `settings.py` module. When Gnotty is run as a stand-alone client, these same settings can be defined via the command-line, or in a separate Python configuration module. See the "Stand-Alone Web Client" section below for details.

- `GNOTTY_HTTP_HOST` - HTTP host address to serve from. *string, default: 127.0.0.1*
- `GNOTTY_HTTP_PORT` - HTTP port to serve from. *integer, default: 8080*
- `GNOTTY_IRC_HOST` - IRC host address to connect to. *string, default: irc.freenode.net*
- `GNOTTY_IRC_PORT` - IRC port to connect to. *integer, default: 6667*
- `GNOTTY_IRC_CHANNEL` - IRC channel to join. *string, default: #gnotty*
- `GNOTTY_IRC_CHANNEL_KEY` - Optional key required to access the IRC channel. *string, default: None*
- `GNOTTY_BOT_CLASS` - Dotted Python path to the IRC client bot class to run. *string, default: gnotty.bots.BaseBot*
- `GNOTTY_BOT_NICKNAME` - IRC nickname the logging client will use. *string, default: gnotty*
- `GNOTTY_BOT_PASSWORD` - Optional IRC password for the bot. *string, default: None*
- `GNOTTY_LOGIN_REQUIRED` - Django login required for all URLs (Django only) *boolean, default: False*
- `GNOTTY_DAEMON` - run in daemon mode. *boolean, default: False*
- `GNOTTY_PID_FILE` - path to write PID file to when in daemon mode. *string, default: [tmp]/gnotty-[http-host]-[http-port].pid*
- `GNOTTY_LOG_LEVEL` - Log level to use. `DEBUG` will spew out all IRC data. *string, default: INFO*

To be clear: the IRC host and port are for specifying the IRC server to connect to. The HTTP host and port are what will be used to host the gevent/WebSocket server.

CHAPTER 3

Django Integration

With the above settings defined in your Django project's `settings.py` module, a few more steps are required. As with most Django apps, add `gnotty` to your `INSTALLED_APPS` setting, and `gnotty.urls` to your project's `urls.py` module:

```
# settings.py
INSTALLED_APPS = (
    # other apps here
    'gnotty',
)

# urls.py
from django.conf.urls.defaults import patterns, include, url
urlpatterns = patterns('',
    # other patterns here
    ('^irc/', include('gnotty.urls')),
)
```

As you can see we've mounted all of the urls Gnotty provides under the prefix `/irc/` - feel free to use whatever suits you here. With this prefix, the URL on our Django development server http://127.0.0.1:8000/irc/ will load the chat interface to the IRC channel, along with a search form for searching the message archive, and links to browsing the archive by date.

The final step when integrated with Django is to run the Gnotty server itself. The Gnotty server is backed by gevent, and will host the WebSocket bridge to the IRC channel. It will also start up the IRC bot that will connect to the channel, and log all of the messages in the channel to the database archive.

Running the Gnotty server when integrated with Django is simply a matter of running the `gnottify` Django management command:

```
$ python manage.py gnottify [options]
```

Note that each of the configuration options can also be specified as arguments to the `gnottify` management command. The names and formats used in this context are the same as those described next for the stand-alone web client. Any options provided as command-line arguments take precendence over those defined in your Django project's `settings.py` module.

The `gnottify_runserver` command is also included, which will run both the Gnotty server and Django's `runserver` command at once, which is useful during development.

CHAPTER 4

Stand-Alone Web Client

As mentioned, Gnotty can also be run as a stand-alone web client without using Django at all. In this mode, only the web interface to the IRC channel is provided, along with whichever IRC bot class is configured. Message logging can be configured using standard handlers for the `logging` module in Python's standard library.

Once installed, the command `gnottify` should be available on your system, and all of the options described earlier can be provided as arguments to it:

```
$ gnottify --help
Usage: gnottify [options]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -a HOST, --http-host=HOST
                        HTTP host address to serve from
                        [default: 127.0.0.1]
  -p PORT, --http-port=PORT
                        HTTP port to serve from
                        [default: 8080]
  -A HOST, --irc-host=HOST
                        IRC host address to connect to
                        [default: irc.freenode.net]
  -P PORT, --irc-port=PORT
                        IRC port to connect to
                        [default: 6667]
  -C CHANNEL, --irc-channel=CHANNEL
                        IRC channel to join
                        [default: #gnotty]
  -K CHANNEL_KEY, --irc-channel-key=CHANNEL_KEY
                        Optional key required to access the IRC channel
  -c DOTTED_PYTHON_PATH, --bot-class=DOTTED_PYTHON_PATH
                        Dotted Python path to the IRC client bot class to run
                        [default: gnotty.bots.LoggingBot]
  -n NICKNAME, --bot-nickname=NICKNAME
                        IRC nickname the bot will use
                        [default: gnotty]
   -x PASSWORD, --bot-password=PASSWORD
                        Optional IRC password for the bot
                        [default: None]
  -D, --daemon          run in daemon mode
  -k, --kill            Shuts down a previously started daemon
  -F FILE_PATH, --pid-file=FILE_PATH
                        path to write PID file to when in daemon mode
```

```
  -l INFO|DEBUG, --log-level=INFO|DEBUG
                        Log level to use. DEBUG will spew out all IRC
                        data.
                        [default: INFO]
  -f FILE_PATH, --conf-file=FILE_PATH
                        path to a Python config file to load options from
```

Note the final argument in the list, `--conf-file`. This can be used to provide the path to a Python config module, that contains each of the settings described earlier. Any options provided via command-line arguments will take precedence over any options defined in the Python configuration module.

CHAPTER 5

Daemon Mode

Gnotty can be configured to run as a background process when the `GNOTTY_DAEMON` setting is set to `True` (the `--daemon` arg when running stand-alone). When in daemon mode, Gnotty will write its process ID to the absolute file path specfified by the `GNOTTY_PID_FILE` setting (the `--pid-file` arg when running stand-alone). If the PID file path is not configured, Gnotty will use a file name based on the HTTP host and port, in your operating system's location for temporary files.

When run in daemon mode, Gnotty will check for an existing PID file and if found, will attempt to shut down a previously started server with the same PID file.

CHAPTER 6

IRC Bots

When running, Gnotty hosts an IRC bot that will connect to the configured IRC channel. The `gnotty.bots.BaseBot` bot is run by default, which implements message logging and support for commands issued within the IRC channel, and webhooks, which allow the IRC bot to receive data over HTTP.

You can implement your own IRC bot simply by subclassing `gnotty.bots.BaseBot` and defining the Python dotted path to it on startup, via the `GNOTTY_BOT_CLASS` setting (the `--bot-class` arg when running standalone).

The `gnotty.bots.BaseBot` class is derived from the third-party `irclib` package's `irc.client.SimpleIRCClient` class (and translated into a Python new-style class for sanity). The IRC bot will have all of the same methods and events available as the `SimpleIRCClient` class.

These are the built-in IRC bot classes provided by the `gnotty.bots` package:

- `gnotty.bots.BaseBot` - The default bot class that implements logging and handling for commands and webhooks. Your custom bot should subclass this.
- `gnotty.bots.ChatBot` - A bot that demonstrates interacting with the IRC channel by greeting and responding to other users. Requires the `nltk` package to be installed.
- `gnotty.bots.commits.CommitMixin` - A base bot mixin for receiving commit information for version control systems via bot webhooks, and relaying the commits to the IRC channel. Used as the base for the `GitHubBot` and `BitBucketBot` bots.
- `gnotty.bots.GitHubBot` - `CommitMixin` subclass for GitHub
- `gnotty.bots.BitBucketBot` - `CommitMixin` subclass for Bitbucket
- `gnotty.bots.CommandBot` - A bot that implements a handful of basic commands that can be issued by users in the channel.
- `gnotty.bots.RSSBot` - A bot that watches RSS feeds and posts new items from them to the IRC channel.
- `gnotty.bots.Voltron` - All of the available bots, merged into one super bot.

Take a look at the source code for the `gnotty.bots` package. You'll see that the different features from all of the available bots are implemented as mixins, which you can mix and match together when building your own bot classes.

CHAPTER 7

Bot Events

Gnotty's IRC bots make use of an event handling system. Event handlers are implemented as methods on any of the classes used to build a bot. Each event handler gets wrapped with the decorator `gnotty.bots.events.on`, which takes a single positional argument for the event name and marks the method as being a handler for that event. The decorator may also accept optional keyword arguments depending on the type of event. Several types of events are available:

- IRC channel events, as implemented by the `irclib` package's `irc.client.SimpleIRCClient` class.
- IRC commands, which are custom commands that can be triggered by users in the IRC channel. Each of these take a `command` keyword argument, which defines the command name.
- Timer events, which are handlers that are periodically run at a defined time interval. Each of these take a `seconds` keyword argument, which defines the time interval.
- Webhooks, which are handlers that accept data over HTTP. Each of take a `urlpattern` keyword argument which defines a regular expression to match against the webhook's URL, similar to Django's `urlpatterns`.

Here's an example IRC bot that implements all the above event types:

```
from gnotty.bots import BaseBot, events

class MyBot(BaseBot):

    @events.on("join")
    def my_join_handler(self, connection, event):
        """IRC channel event - someone joined the channel."""
        nickname = self.get_nickname(event)
        self.message_channel("Hello %s" nickname)

    @events.on("timer", seconds=10)
    def my_timer(self):
        """Do something every 10 seconds."""
        msg = "Another 10 seconds has passed, are you annoyed yet?"
        self.message_channel(msg)

    @events.on("command", command="!time")
    def my_time_command(self, connection, event):
        """Write the time to the channel when someone types !time"""
        from datetime import datetime
        return "The date and time is %s" % datetime.now()

    @events.on("webhook", urlpattern="^/webhook/do/something/$")
    def my_webhook_handler(self, environ, url, params):
        """Tell the channel that someone hit the webhook URL."""
```

```
        ip = environ["REMOTE_ADDR"]
        self.message_channel("The IP %s hit the URL %s" (ip, url))
```

Given the above class in an importable Python module named `my_bot.py`, Gnotty can be started with the bot using the following arguments:

```
$ gnottify --http-host=127.0.0.1 --http-port=8000 --bot-class=my_bot.MyBot
```

CHAPTER 8

Channel Events

As described above, each of the IRC channel events implemented by the `irclib` package's `irc.client.SimpleIRCClient` class are available as event handlers for an IRC bot. Consult the `irclib` docs or source code for details about each of the IRC channel events that are implemented, as documenting all of these is beyond the scope of this document. Here are some of the common events to get you started:

- `events.on("welcome")`: Bot has connected to the server but not yet joined the channel.
- `events.on("namreply")`: Bot receives the initial list of nicknames in the channel once joined.
- `events.on("join")`: Someone new joined the channel.
- `events.on("quit")`: Someone left the channel.
- `events.on("pubmsg")`: A message was sent to the channel.

Each of the channel events receive a `connection` and `event` argument, which are objects for the connection to the IRC server, and information about the event that occurred.

CHAPTER 9

Command Events

Event handlers for simple commands can be implemented using the `command` event name for the `gnotty.bots.events.on` decorator. The decorator then takes a `command` keyword argument, which is the command name itself. Command events are then triggered by any public messages in the channel that contain the command name as the first word in the message. Each subsequent word in the message after the command is then passed as a separate argument to the event handler method for the command. In each command event handler method, the bot can then perform some task, and return a message back to the channel.

CHAPTER 10

Timer Events

These event handlers are defined using the `timer` event name for the `gnotty.bots.events.on` decorator, and simply run repeatedly at a given interval. A `seconds` keyword argument to the decorator defines the interval in seconds as an integer. Note that the `seconds` keyword argument is used to `sleep` after each time the event handler is run, in order to implement the interval, so an interval of 30 seconds won't necessarily run precisely twice per minute, since the event handler itself will take time to execute, particularly if it accesses external resources over a network.

CHAPTER 11

Webhook Events

IRC bots run by Gnotty contain the ability to receive data over HTTP via webhooks. Webhooks are methods defined on the bot class with the `webhook` event handler name specified for the `gnotty.bots.events.on` decorator. The decorator also requires a `urlpattern` keyword argument, which is a regular expression for matching the webhook URL.

The gevent web server will intercept any URLs prefixed with the path `/webhook/`, and pass the request onto the bot which will match the URL to any of the URL patterns defined by webhook event handlers on the running bot class. A webhook event handler receives the following arguments:

- `environ` - The raw environment dict supplied by the gevent web server that contains all information about the HTTP request.
- `url` - The actual URL accessed.
- `params` - A dictionary containing all of the POST and GET data.

Note that the `url` and `params` arguments are simply provided for extra convenience, as their values (and all other environment information) are already available via the `environ` argument.

CHAPTER 12

Message Logging

By default, each IRC message in the channel is logged by the IRC bot run by Gnotty. Logging occurs using Python's logging module, to the logger named `irc.message`.

Each log record contains the following attributes, where `record` is the log record instance:

- `record.server` - The IRC server the message occurred on.
- `record.channel` - The IRC channel the message occurred on.
- `record.nickname` - The nickname of the user who sent the message.
- `record.msg` - The actual message string itself.
- `record.join_or_leave` - `True` if the record was for a user joining or leaving the channel, otherwise `False`.

Here's an example of adding an extra logging handler for IRC messages:

```
from logging import getLogger, StreamHandler

class MyLogHandler(StreamHandler):
    def emit(self, record):
        # Do something cool with the log record.
        print record.msg

getLogger("irc.message").addHandler(MyLogHandler())
```

CHAPTER 13

JavaScript Client

The web client that Gnotty provides includes all the necessary JavaScript files for communicating with the WebSocket server, such as Douglas Crockford's `json2.js`, and the `socket.io.js` library itself. Also provided is the file `gnotty.js` which implements a couple of public functions used by the web interface. The first is the `gnotty` JavaScript function, which deals directly with the HTML structure of the chat template:

```
// Start up the default UI. This function isn't very
// interesting, since it's bound to the HTML provided
// by Gnotty's chat template.
gnotty({
    httpHost:     '127.0.0.1',
    httpPort:     '8080',
    ircHost:      'irc.freenode.net',
    ircPort:      '6667',
    ircChannel:   '#gnotty'
});
```

The second interface is the `IRCClient` function. This is of particular interest if you'd like to create your own chat interface, as it deals exclusively with communication between the web browser and the WebSocket server. Here's an example client that simply writes events out to the console:

```
// Prompt the user for a nickname and password,
// and create an IRC client.
var client = new IRCClient({
    httpHost:     '127.0.0.1',
    httpPort:     '8080',
    ircHost:      'irc.freenode.net',
    ircPort:      '6667',
    ircChannel:   '#gnotty',
    ircNickname:  prompt('Enter a nickname:')
    ircPassword:  prompt('Enter a password (optional):')
});

// When the client first joins the IRC channel,
// send a message to the channel to say hello.
client.onJoin = function() {
    console.log('joined the channel');
    client.message('Hello, is it me you\'re looking for?');
};

// When someone joins or leaves the channel, we're given the
// entire user list. It'a an array of objects, each with a
// nickname and color property.
client.onNicknames = function(nicknames) {
```

```
    nicknames = $.map(nicknames, function(obj) {
        return obj.nickname;
    }).join(', ');
    console.log('The user list changed, here it is: ' + nicknames);
});

// Whenever a message is received from the channel, it's an
// object with nickname, message and color properties.
client.onMessage = function(data) {
    console.log(data.nickname + ' wrote: ' + data.message);
});

// When we leave, reload the page.
client.onLeave = function() {
    location.reload();
};

// The IRC server rejected the nickname.
client.onInvalid = function() {
    console.log('Invalid nickname, please try again.');
};
```

As you may have guessed, the server-side settings configured for Gnotty are passed directly into the `gnotty` JavaScript function, which then creates its own `IRCClient` instance.

You'll also see the data sent to the `onMessage` and `onNickname` events contain color values that the interface can use for colorizing nicknames. These are calculated on the server, so that both the chat interface and message archive produce consistent colors every time a particular nickname is displayed.

CHAPTER 14

Hosting Private Chat Rooms

Creating a private login-protected chat room for your team members to collaborate on is a breeze using Gnotty. By setting the `GNOTTY_LOGIN_REQUIRED` setting to `True`, Gnotty will require each user to have a Django user account which they can authenticate with. The following steps should get you started:

- Create a Django project with Gnotty installed, using the steps described above under *Django Integration*. Take a look at the `example_project` directory within Gnotty, which contains a working Django project with Gnotty configured.
- Install an IRC server such as ngIRCd. ngIRCd can be installed on both Linux or OSX with a single command (this works great for local development on OSX). Be sure to configure ngIRCd to only allow local connections, so that only Gnotty can connect to it.

With the above setup, all that is then needed are the following Gnotty settings configured in your Django project's `settings.py` module:

```
GNOTTY_IRC_HOST = "127.0.0.1"
GNOTTY_LOGIN_REQUIRED = True
GNOTTY_IRC_CHANNEL = "#mychannel"  # This can be anything really.
```