
listenclosely Documentation

Release 0.1.1

Juan Madurga

January 15, 2016

1	listenclosely	3
1.1	Documentation	3
1.2	Quickstart	3
1.3	Features	4
1.4	Running Tests	4
2	Installation	5
3	Usage	7
3.1	How it works	8
4	Customization	11
4.1	Agent strategy	11
4.2	Message Service Backend	11
5	Contributing	13
5.1	Types of Contributions	13
5.2	Get Started!	14
5.3	Pull Request Guidelines	14
5.4	Tips	15
6	Credits	17
6.1	Development Lead	17
6.2	Contributors	17
7	History	19
7.1	0.1.0 (2016-01-14)	19

Contents:

listenclosely

CI: PyPI: Docs: Listenclosely is a django-app that works as a middleman to connect instant messaging clients. Think on a Call Center/Customer Service using using instant messaging... exactly what it does.

- It is simple, connects *Askers* with online *Agents* until the *Chat* is considered as terminated and the *Agent* is released to attend other *Asker* chats.
- It is flexible, so you can define your own strategies to assign *Agents* to *Askers* and your own messaging backend services.

Messaging Services integrated:

- Whatsapp <https://github.com/jlmadurga/listenclosely-whatsapp>
- Telegram <https://github.com/jlmadurga/listenclosely-telegram>

1.1 Documentation

The full documentation is at <https://listenclosely.readthedocs.org>.

- Asker1 is chatting with the Busy Agent
- Asker2 try to chat but no free Agent was free so is waiting with a Pending chat to be attended by an agent
- Asker3 is opening a chat and Online Agent will be assigned to the chat

1.2 Quickstart

Install listenclosely:

```
pip install listenclosely
```

Then use it in a project:

```
import listenclosely
```

Add it to django apps and migrate:

```
INSTALLED_APPS = [  
    ...  
    'listenclosely',  
    ...  
]
```

```
]
python manage.py migrate
```

Select, install and configure service backend

```
LISTENCLOSELY_MESSAGE_SERVICE_BACKEND = "listenclosely_telegram.service.TelegramMessageServiceBackend"
```

Define your agent strategy or define your own:

```
LISTENCLOSELY_AGENT_STRATEGY = 'listenclosely.strategies.first_free.FirstFreeAgentStrategy'
```

Add step to your celery app:

```
from listenclosely.celery import ListenCloselyAppStep
app.steps['worker'].add(ListenCloselyAppStep)
```

Start your celery app usign gevent:

```
celery --app=demo_app.celery:app worker -P gevent
```

Call listen task or define a celery scheduler to execute:

```
from listenclosely import tasks
tasks.listen.delay()
```

1.3 Features

- Connects *Askers* and *Agents* in chats to establish a *Chat*
- Strategies to find *Agent* to attend new *Asker* chat. Define your own strategies
- Messaging Service Backend: Define your own messaging service backend implementations.
- Cron tasks for attending pending chats and to terminate obsolete chats to release *Agents*

1.4 Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install -r requirements/test.txt
(myenv) $ make test
```

Installation

At the command line:

```
$ pip install listenclosely
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv listenclosely  
$ pip install listenclosely
```

Usage

Then use it in a project:

```
import listenclosely
```

Add it to django apps and migrate:

```
INSTALLED_APPS = [  
    ...  
    'listenclosely',  
    ...  
]  
python manage.py migrate
```

Select, install and configure service backend

```
LISTENCLOSELY_MESSAGE_SERVICE_BACKEND = "listenclosely_telegram.service.TelegramMessageServiceBackend"
```

Define your agent strategy or define your own:

```
LISTENCLOSELY_AGENT_STRATEGY = 'listenclosely.strategies.first_free.FirstFreeAgentStrategy'
```

Add step to your celery app:

```
from listenclosely.celery import ListenCloselyAppStep  
app.steps['worker'].add(ListenCloselyAppStep)
```

Start your celery app usign gevent:

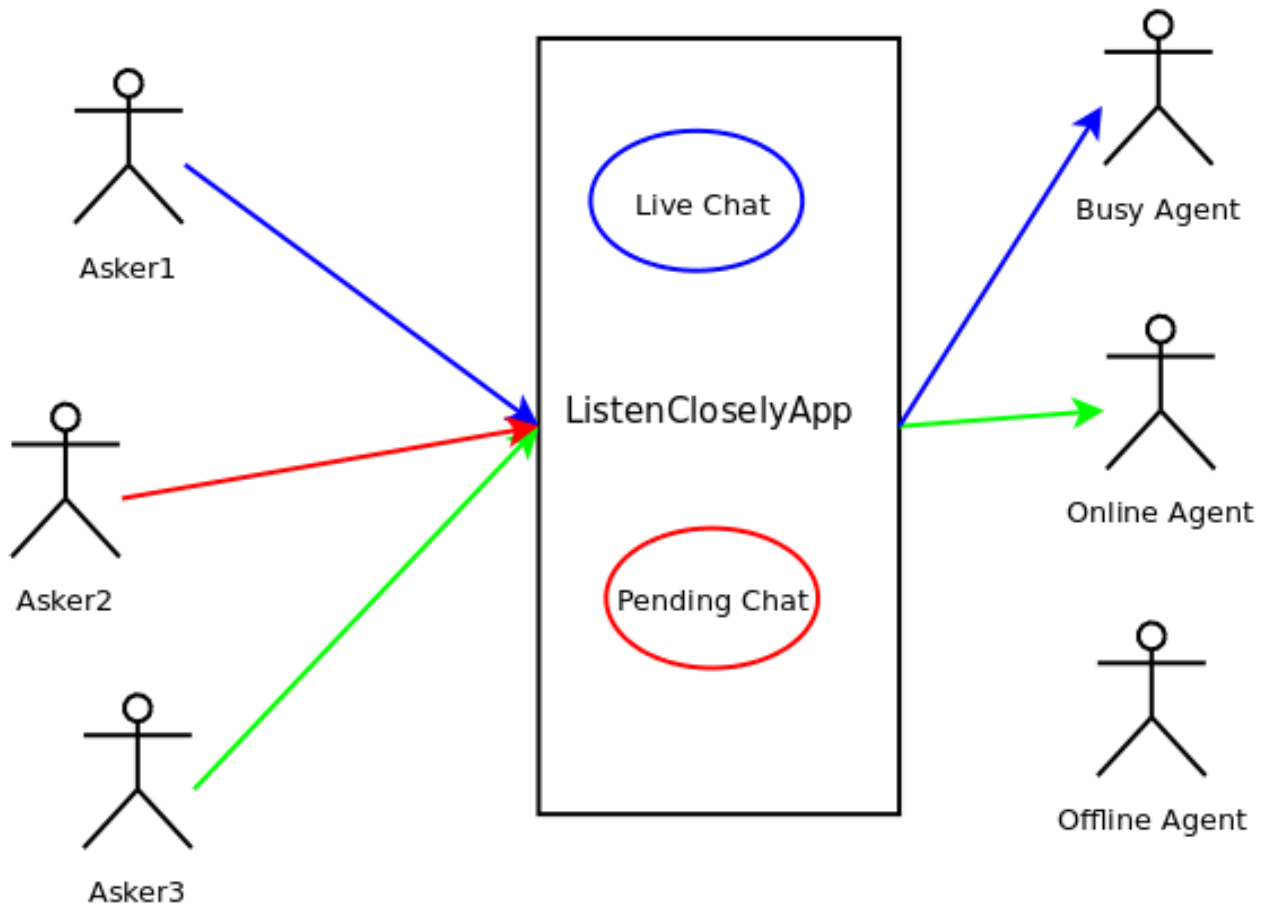
```
celery --app=demo_app.celery:app worker -P gevent
```

Call listen task or define a celery scheduler to execute:

```
from listenclosely import tasks  
tasks.listen.delay()
```

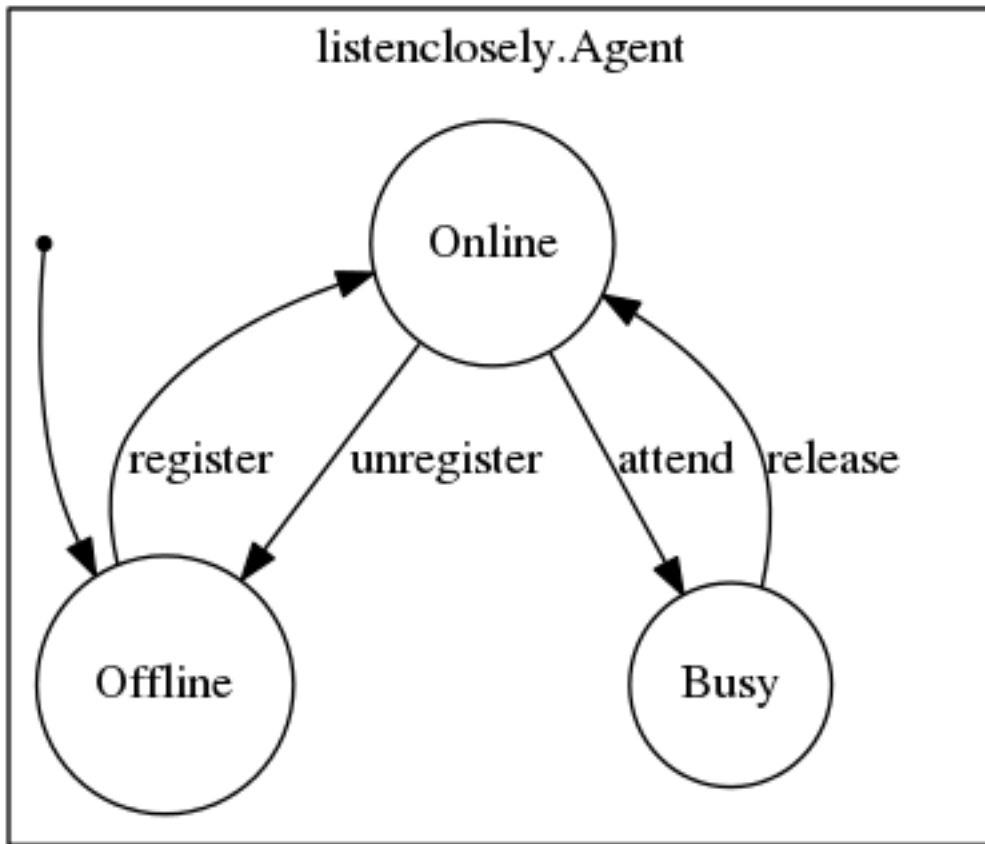
NOTE: listenclosely comes with a demo with celery configuration example.

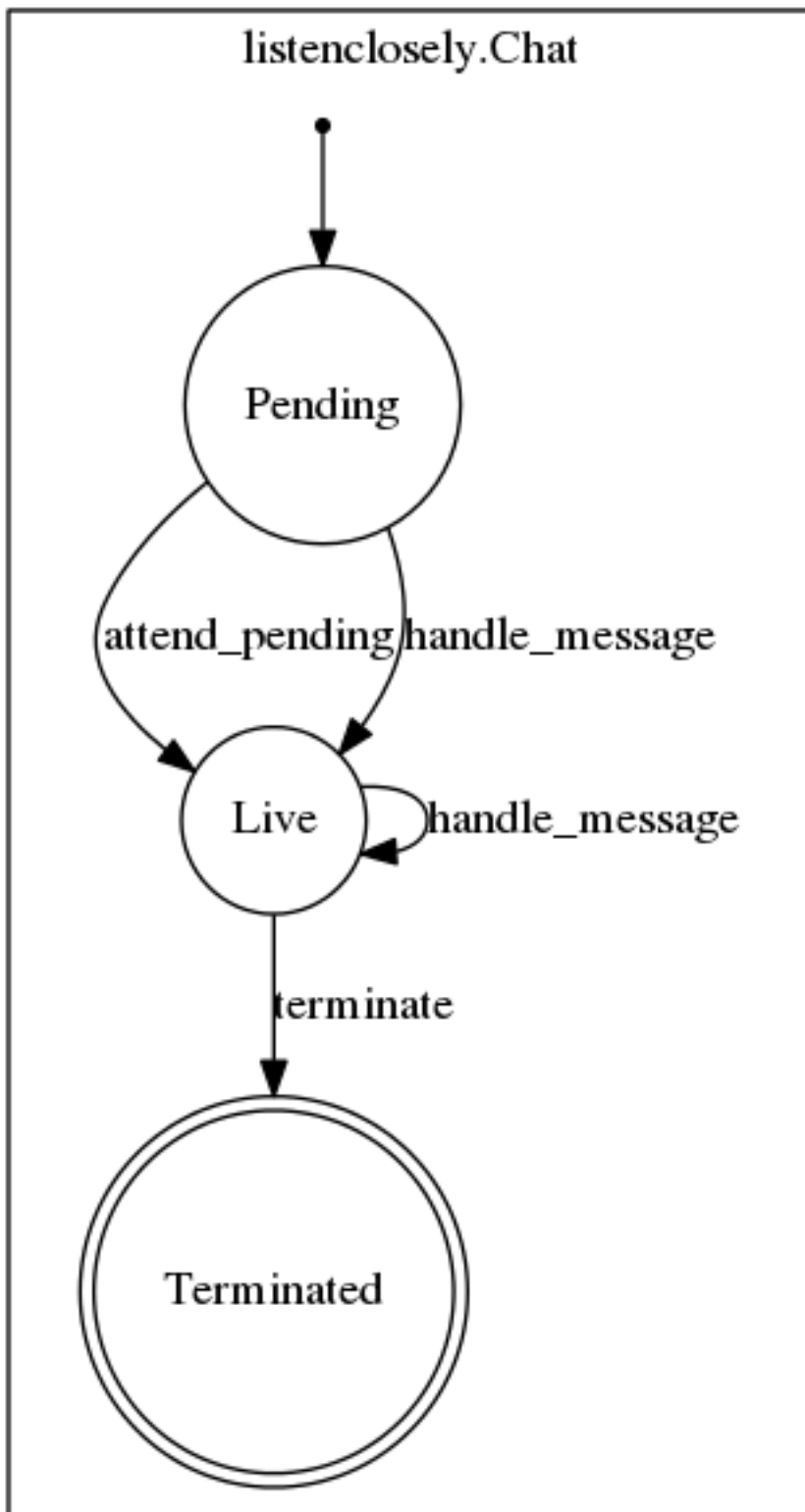
3.1 How it works



- Asker1 is chatting with the Busy Agent
- Asker2 try to chat but no free Agent was free so is waiting with a Pending chat to be attended by an agent
- Asker3 is opening a chat and Online Agent will be assigned to the chat

State machines of *Agent* and *Chat*:





Customization

ListenClosely is easy to be customized with your own requirements

4.1 Agent strategy

Just extend *strategies.base.BaseAgentStrategy* and define your own *free_agent* function:

```
class FirstFreeAgentStrategy(BaseAgentStrategy):
    """
    Choose first free agent
    """

    def free_agent(self):
        free_agents = Agent.online.all()
        if free_agents:
            return free_agents[0]
        return None
```

Then configure settings:

```
LISTENCLOSELY_AGENT_STRATEGY = 'your_strategy.YourAgentStrategy'
```

4.2 Message Service Backend

Extend *services.base.BaseMessageServiceBackend*. You must implement some methods:

```
def listen(self):
    """
    Connect to service and listen for receive messages.
    To implement in concrete services
    """
    raise NotImplementedError('subclasses of BaseMessageServiceBackend must override listen() method')

def send_message(self, id_service, content):
    """
    Send message to a instant messages service
    To implement in concrete services
    :rtype string message_id: identifier for message service
    """
    raise NotImplementedError('subclasses of BaseMessageServiceBackend must override send_message() method')
```

```
def disconnect(self):  
    """  
    Disconnect to service.  
    To implement in concrete services  
    """  
    raise NotImplementedError('subclasses of BaseMessageServiceBackend must override disconnect() method')
```

Use other services as example. At the moment:

- Whatsapp: <https://github.com/jlmadurga/listenclosely-whatsapp>
- Telegram: <https://github.com/jlmadurga/listenclosely-telegram>

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at <https://github.com/jlmadurga/listenclosely/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

5.1.4 Write Documentation

listenclosely could always use more documentation, whether as part of the official listenclosely docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/jlmadurga/listenclosely/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *listenclosely* for local development.

1. Fork the *listenclosely* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/listenclosely.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv listenclosely
$ cd listenclosely/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 listenclosely tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/jlmadurga/listenclosely/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_listenclosely
```

Credits

6.1 Development Lead

- Juan Madurga <jlmadurga@gmail.com>

6.2 Contributors

None yet. Why not be the first?

History

7.1 0.1.0 (2016-01-14)

- First release on PyPI.