Jigna Documentation

Release 0.10.1

Jigna developers

Jul 22, 2019

Contents

Jigna allows you to create a HTML/CSS/Javascript based user interface for a Python application. The UI can be viewed either in a Qt-based application or entirely on a web-browser.

CHAPTER 1

Introduction

Jigna aims to provide an HTML based solution for creating beautiful user interfaces for Python applications, as opposed to widget based toolkits like Qt/wx or native toolkits. It provides a seamless two-way data binding between the Python model and the HTML view by creating a Python-JS communication bridge. This ensures that the view is always live as it can automatically update itself when the model changes, and update the model when user actions take place on the UI. The Jigna view can be rendered in an in-process Qt widget or over the web in a browser.

The primary use of jigna is to provide a way to write desktop GUI applications using HTML, and we use the Qt backend for that. You can also use jigna to write GUIs that run on the browser. This is done via the web backend.

CHAPTER 2

Installation

Jigna works with Python 2.x and Python 3.x (although for 3.x, you have to install the dependencies yourself).

The setup procedure is simple. Make sure you have a Python installation to which you have write access and run the following command:

```
$ pip install jigna
```

This will not pull in any Qt requirements but will require tornado as it is easy to install. The test requirements can be installed via:

```
$ pip install jigna[test]
```

This will install, PySide, nose, mock, coverage, and selenium if you do not already have them.

You may also use the `requirements.txt` file to install the necessary dependencies. This does not install PyQt or PySide though.

CHAPTER 3

Examples

There are several examples to get started, look in the `examples/` directory and run them.

CHAPTER 4

Tests

Running the tests requires selenium and nose. The tests are in `jigna/tests/`. You may run them from the root directory of the sources using `nosetests`.

CHAPTER 5

Getting Started

Let us say we have a nice application model written in Python (specifically, in Traits):

```
from traits.api import HasTraits, Str

class Person(HasTraits):
    name = Str
    greeting = Str
    def _name_changed(self):
        self.greeting = "Hello " + self.name

person = Person(name='Fred')
```

Jigna lets you visualize this model using HTML such that the model and the view are fully connected. Here is a sample HTML (an AngularJS template):

```
body_html = """
    Name: <input ng-model="person.name"/><br>
    Greeting:
    <h1>{{person.greeting}}</h1>
"""
```

Notice how the HTML is directly referencing model attributes via `person.name` and `person.greeting`. We now bind this declarative view to the model and create a Qt based UI:

```
from jigna.api import HTMLWidget, Template
template = Template(body_html=body_html)

from jigna.qt import QtGui
app = QtGui.QApplication([])
widget = HTMLWidget(template=template, context={'person': person})
widget.show()
app.exec_()
```

This produces a Qt window containing an HTML UI which responds automatically to any changes in the model and vice-versa. It can optionally be styled with CSS and made interactive with Javascript. This provides a nice way of

easily building rich, live user interfaces for Python apps. Note that the Qt imports were made using the `jigna.qt` module. This module allows one to switch between PySide and PyQt easily by exporting an environment variable `QT_API`, if this is set to `pyside` then `PySide` will be used and if it is set to `pyqt` then `PyQt` will be used.

This is nice for several reasons:

- The view code is declarative and hence easy to read.
- The binding between the model and the view is automatic.
- HTML/CSS/JS today is very powerful
  - there are many JS libraries for a variety of tasks
  - your development team doesn't have to worry about creating widgets or the limitations in the toolkit's widget set as there are thousands of developers worldwide creating awesome CSS/JS widgets for you.
- Much easier to find people who know HTML/CSS/JS than Qt or a native toolkit.
- There is a complete separation of view from the model and this allows us to hand off the entire UI to an HTML/CSS/JS guru.

And since this is HTML, the jigna `Template` can also be easily served on a web browser if you don't have Qt installed. The wiring changes a bit for this, so the final example looks like this:

```
from tornado.ioloop import IOLoop
from jigna.web_app import WebApp

ioloop = IOLoop.instance()

app = WebApp(template=template, context={'person': person})
app.listen(8000)

# Serving the app on http://localhost:8000/.
ioloop.start()
```

This starts up a web server to which one can connect multiple browsers to see and interact with the model.

Please note that any imports from `jigna.api` will try and import Qt modules if they are installed, so if you need an application with a pure web-based UI and wish to avoid those imports, you may wish to use the following imports:

```
from jigna.template import Template
from jigna.web_app import WebApp
```

The above example was very simple as the purpose of that was to show how jigna is wired together. For further use cases of jigna, please refer to the examples in the `examples` directory. They are numbered and are meant to act as a tutorial if followed in sequence.

One may also use Vue.js to build the HTML UI. The approach is very similar to that using AngularJS. A Vue template is available from:

```
from jigna.vue_template import VueTemplate
```

An example showing how to use this with a Qt app is in examples/ex22_vuejs_demo.py with the corresponding HTML file in examples/ex22_vuejs_demo.html.

A similar example showing how to use this with a pure web-based app is in examples/ex23_vuejs_demo.py. The HTML file is the same.

CHAPTER 6

Changelog

6.1 0.10.1

- Fix a bug with methods not being exposed in Python-3.x.
- Fix and improve web-UI examples to run without Qt.
- Improve test stability a little.

6.2 0.10.0

- Python3 support.
- Jigna is now tested on Linux, OS X and Windows.
- Fix problem with `jigna.threaded` calls.
- Work with PyQt and PySide – currently Qt4 only though.
- Use a better version of guess_type to guess file mime type.
- Add full async support for web UIs. The sync backend is the default.
- Documentation and example cleanup.
- Update documentation, README, setup.py.
- Make available on PyPI.

6.3 0.9.9

These changes are for all previous releases.

- Add support for Vue.js HTML templates.

- Avoid Qt imports for pure web-based UI access.
- Fix broken page reloads.
- Fix issues with serving files having spaces.
- Fix loading of jigna.js on Windows.

6.4 0.9.1

- First public release.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search