PyIRE Documentation

Release 0.19.2

Mark Hall

November 14, 2015

Contents

1 Setting up PyIRE **3**
- 1.1 Installation . . . 3
- 1.2 Configuration . . . 4
- 1.3 Deployment . . . 5

2 Developing Interactive Information Retrieval Experiments **7**
- 2.1 Setting up an IIR experiment . . . 7
- 2.2 Creating new IIR UI components . . . 11

3 Indices and tables **13**

This is the documentation for the Python Interactive Information Retrieval Evaluation workbench (PyIRE).

CHAPTER 1

Setting up PyIRE

1.1 Installation

It is generally recommended that PyIRE is installed into its own `virtualenv` (see **'https://pypi.python.org/pypi/virtualenv'_**).

1.1.1 Installing the Core Software

It is recommended that you install the latest stable release, but if you wish to contribute back to the project, please consider installing the latest development version from source.

Stable releases

To install the latest stable release run:

```
pip install pyire
```

To install an older release, download the release from https://bitbucket.org/mhall/pyire/downloads and then install using:

```
pip install pyire-x.y.z.tar.gz
```

Source code

To install the latest development version from source, clone the Mercurial repository:

```
hg clone hg clone https://bitbucket.org/mhall/pyire
```

1.1.2 Additional Required Packages

PyIRE additionally requires further packages to be fully operational:

- a cryptography library to support secure session cookies;
- a database access library.

Secure session handling

You need to install either *pycryptopp* or *PyCrypto* packages to enable the use of secure session cookies:

```
pip install pycryptopp
```

or

Download PyCrypto from https://www.dlitz.net/software/pycrypto/ and install using `pip install pycrypto-X.Y.tar.gz`.

Database access

PyQuestionnaire uses SQLAlchemy as its database access library and thus in theory supports any database supported by SQLAlchemy. The following databases have been tested with PyQuestionnaire:

PostgreSQL Recommended, fully supported and tested with the `psycopg2` database adapter:

```
pip install psycopg2
```

MySQL Fully supported and tested with the `mysql-python` database adapter:

```
pip install mysql-python
```

SQLite Only supported for testing purposes. SQLite does not support all DDL statements required to upgrade the database for future releases, thus it is not recommended to use it in a production environment.

All other databases Should in theory be supported, but have not been tested. If you have deployed PyIRE using another database backend, please let us know and the documentation will be updated.

1.2 Configuration

The next step is to set up PyQuestionnaire so that it is ready to run. For this you need to:

1. Generate a configuration file
2. Initialise the database

Both tasks are done using the configuration application included in PyIRE. To see all options provided by the configuration application run:

```
PyIRE -h
```

1.2.1 Generate the Configuration

To generate a configuration file run:

```
PyIRE generate-config
```

You will be asked to provide the **'SQLAlchemy connection string'_** for your database. If you don't know it yet, you can accept the default test database and change the configuration setting later.

You can also set the following parameters on the command-line:

- –sqla-connection-string <SQL Alchemy connection string>
- –filename <Configuration Filename defaults to production.ini>

1.2.2 Initialise the Database

The next step is to initialise the database:

```
PyIRE initialise-database <Configuration File>
```

and the required database tables will be created. During testing you might want to re-create the initial database. In that case run:

```
PyIRE initialise-database <Configuration File> --drop-existing
```

and the old tables will be removed and the database re-created in its initial state.

If you change the **'SQLAlchemy connection string'_**, then you need to re-run the database initialisation.

1.3 Deployment

In-production deployment has been tested using Apache2 and mod_wsgi. However, you can also use the internal server provided by **'Pyramid'_**, either directly or behind a reverse proxy.

1.3.1 Deploying with Apache2 & mod_wsgi

To deploy PyIRE via Apache2 and mod_wsgi add the following settings to the VirtualHost configuration:

```
WSGIDaemonProcess pyire user=www-data group=www-data processes=1 threads=10 python-path=/path/to/virt
WSGIScriptAlias /pyire /path/to/the/application.wsgi
<Location /pyire>
    WSGIProcessGroup pyire
</Location>
```

Note: Leave the `processes` value at 1. Use the `threads` option to specify how many parallel requests to support.

Then create the following script to to run the application via WSGI. Adapt it by replacing the paths with the paths to where PyIRE is installed:

```
import os
os.chdir(os.path.dirname(__file__))
import site
import sys

# Remember original sys.path.
prev_sys_path = list(sys.path)

site.addsitedir('/path/to/virtualenv/lib/python2.7/site-packages')

# Reorder sys.path so new directories at the front.
new_sys_path = []
for item in list(sys.path):
    if item not in prev_sys_path:
        new_sys_path.append(item)
        sys.path.remove(item)
sys.path[:0] = new_sys_path

from pyramid.paster import get_app
from paste.script.util.logging_config import fileConfig
fileConfig('/path/to/the/application/pyire.ini')
application = get_app('/path/to/the/application/pyire.ini', 'main')
```

CHAPTER 2

Developing Interactive Information Retrieval Experiments

This section covers developing an Interactive Information Retrieval (IIR) experiment using PyIRE. It covers both setting up an experiment using existing components and creating new components.

2.1 Setting up an IIR experiment

This document takes you through the steps needed to set up a new IIR experiment using existing user interface (UI) components. If you want to create your own components, read Creating new IIR UI components.

2.1.1 Creating the experiment configuration file

Creating the PyIRE uses experiment configuration files to be able to host multiple experiments at the same time. To create an initial configuration file use the `PyIRE` utility:

```
> PyIRE generate-experiment -h

usage: PyIRE generate-experiment [-h] [--filename FILENAME]
                                 [--template {default,left_column,right_column,three_column}]

optional arguments:
  -h, --help            show this help message and exit
  --filename FILENAME   Configuration file name
  --template {default,left_column,right_column,three_column}
                        Template to use for the new configuration file
```

By default it will generate a configuration file 'experiment.ini', with a simple default template, but you can change that using the `--filename` and `--template` parameters. Running `PyIRE generate-experiment -h` will list the available templates.

The experiment configuration file uses a standard INI file style, in which sections are defined using square brackets `[Section Name]` and each section then has a number of configuration settings. The only section that PyIRE requires is the `[General]` section that acts as the entry point into the configuration:

```
[General]
layouts =
```

2.1.2 Loading the experiment

To load the experiment, the experiment configuration file created above must be linked in the main PyIRE configuration file. To do this, update the `pyire.apps` setting in the PyIRE configuration file to include the new experiment. For each experiment you want to load into PyIRE you must specify a `application_name:config_file.ini` pair:

```
pyire.apps = application_name:path/to/experiment.ini, ...
```

The `application_name` will be used to generate URLs, so should consist only of letters, numbers, and underscores. Restart PyIRE and the new experiment will be loaded into PyIRE. You can check that it has been loaded by browsing to the root URL at which you have installed PyIRE. That will bring up the status page, which lists all loaded experiments.

2.1.3 Structure of an experiment

PyIRE experiments consist of two parts. *Layouts* define a set of components that are shown as one page to the experiment participants. An experiment must have at least one, but can have as many *layouts* as required. Each *layout* consists of a set of *components* that generate the user interface (UI) that is shown to the experiment participants.

Components have a unique name which they use to store their current state for each participant. *Components* that are used in multiple *layouts* with the same unique name will share that participant state, making it possible to create multiple UI layouts where the displayed information is consistent. An example of where this could be used is to implement a post-experiment review layout. This would be implemented in a separate *layout* to the main experiment *layout*, but because the components would re-use the names, the review layout's components' initial state would be the final state of the main experiment layout's components.

2.1.4 Configuring the experiment layouts

To add a layout to the experiment, add a new section to the configuration file with the new layout's configuration name. Then add that name to the list of layouts in the `General` section:

```
[General]
layouts = MainLayout

[MainLayout]
```

Layout sections have two required configuration settings. Then `name` is the unique name that is used by PyIRE to generate the URL at which the layout is accessed. This can be the same as the section name, but it does not have to be. The `components` is the list of components to add to the layout:

```
[MainLayout]
name = main
components = ...
```

2.1.5 Configuring individual components

Individual components are added to the layout in the same way that the layout is added to the experiment. For each component, a new section is added to the configuration file and that section is listed in the layout's list of components. Components have two required settings. The name used to store the component's current state and the handler class that implements the desired component functionality:

```
[MainLayout]
name = main
components = LeftBar
```

```
[LeftBar]
name = left
handler = pyire.components.core.Container
```

Depending on the component, there might be further required settings. For the default components provided by PyIRE check below to see the available configuration settings.

Laying out components

PyIRE provides a 12 x 12 grid within which the components can be laid out. To specify how many grid cells a component uses add a `class` setting to the component's section. In the `class` setting use `grid-X` and `vgrid-Y` values to specify the component's extent:

```
[LeftBar]
name = left
handler = pyire.components.core.Container
class = grid-3 vgrid-12
```

In the example above a *Container* component is configured to cover a quarter of the available horizontal space and all of the vertical space. This creates a full-page bar, which due to this being the first listed component, will be placed on the left-hand side of the page.

Due to the way the grid is implemented, components will be place from left to right and top to bottom. Due to the way HTML layouts work, components in the next row will be placed vertically below the previous row's component with the largest `vgrid-Y` value. To implement layouts where components in different columns have different heights, you must first add Container components for each of the columns and then add the actual UI components to the respective containers. Each Container implements its own 12 x 12 grid that is used to lay out its child components.

In addition to the `vgrid-Y` vertical extent setting, there are also a `vgrid-fixed` and `vgrid-expand` settings. `vgrid-fixed` ensures that the component's height is fixed to the minimum required to show all its content. `vgrid-expand` is the opposite to this and will expand the components vertical space to the maximum amount of space left after laying out all `vgrid-Y` and `vgrid-fixed` components.

To see an example of this in action, create a new experiment using the `sample_search` template.

Linking together components

2.1.6 Default components

Container

Settings

Signals

Task

Settings

Signals

Search Box

Settings

Signals

Search Results

Settings

Signals

Pagination

Settings

Signals

Search Results Summary

Settings

Signals

History

Settings

Signals

Book Bag

Settings

Signals

Filler

Settings

CHAPTER 3

Indices and tables

- genindex
- modindex
- search