Flowr Documentation

Release 0.2.0

Christopher Trudeau

September 03, 2015

Contents

Most state machine libraries are "static" and require the flow in the state machine to be definied programmatically. Flowr is designed so that you can build state machine flows and store them in a database. There are two key concepts: rule graphs and state machines. The programmer defines one or more sets of rules that describe the allowed flow between states, the user can then use the GUI tools to construct state machines that follow these rules and store the machines in the database. The state machines can then be instantiated for processing the flow which triggers call-back mechanisms in the rule objects on entering and leaving a state.

CHAPTER 1

Installation

Add ‘flowr’ to your `settings.INSTALLED_APPS` field.

Run

```
$ manage.py makemigrations
$ manage.py migrate
```

CHAPTER 2

Demo Installation

A full django project is included in the repository that is used for testing and can give you a quick idea what flowr is about. The project is available in `extras/sample_site`

```
$ cd django-flowr
$ pip install -r requirements.txt
$ cd extras/sample_site
$ pip install -r requirements.txt
$ ./resetdb.sh
$ ./runserver.sh
```

This will create an `sqlite` database with some sample rules. Point your browser at `http://localhost:8000/admin` and login with the username `admin` and the password `admin`. Use the django admin screens to view the flows and rules in the system.

CHAPTER 3

Docs

Docs available at: http://django-flowr.readthedocs.org/en/latest/

Version: 0.2.0

CHAPTER 4

Contents

4.1 Introduction

Flowr is based on two key concepts: rules and flow graphs. A flow graph is the dynamic state machine that the user can create and is stored in the database. Multiple instances of states can then be active for any given flow graph. One or more sets of rules is defined by the programmer which limits the progression of possible states in a flow graph.

Rules are defined by subclassing the `Rule` class and filling in the "children" attribute. Cycles are allowed, so `Rule` instances can point to each other or themselves. Once a hierarchy of `Rule` classes have been defined, a `RuleSet` can be created and stored in the database. `Flow` objects are created with the GUI and describe the flow through a state machine with the state flow graph being a subset of those defined by the collection of `Rule` objects that were registered in the `RuleSet` instance.

The `State` object is an instantiation of a traversal of the state machine represented by a `Flow`. Once there are `State` objects for a `Flow`, the `Flow` can no longer be edited.

Definitions:

- **`Rule` – a base class for rules that the programmer defines which** specifies what other Rule objects can be connected to and what happens when a state is entered and exited
- `RuleSet` – a registery for the collections of Rule subclasses
- **`Flow` – user defined state machine based on a `RuleSet`** instance
- `State` – an instance of a state machine and its current state

4.1.1 Example

A user defines the following `Rule` subclasses:

```
class A(Rule):
    children = [B, C]

class B(Rule):
    children = []

class C(Rule):
    children = [D, E]
    multiple_paths = True

class D(Rule):
    children = []
```

```
class E(Rule):
    children = []
```

A `RuleSet` object is created using the factory and passing in the single starting point of our allowed flows:

```
RuleSet.factory('My Rules', A)
```

Using these rules, you could create some flows:

- A -> B
- A -> C -> D
- A -> C -> D or E

You would not be able to create:

- A -> D
- A -> B -> C

The first of these is not allowed because "D" is not a direct child of "A" in the `Rule` definitions. The second is not allowed because "B" has no allowed children.

4.2 Models

CHAPTER 5

Indices and tables

- genindex
- modindex
- search