
django-river Documentation

Release 3.0.0

Ahmet DAL

Nov 18, 2019

Contents

1	Getting Started	3
2	Contents	5
2.1	Getting Started	5
2.2	Overview	6
2.3	Administration	8
2.4	API Guide	12
2.5	Authorization	17
2.6	Hooking Guide	17
2.7	Migration Guide	21
2.8	Change Logs	22
3	Indices and tables	27

River is an open source and always free workflow framework for Django which support on the fly changes instead of hardcoding states, transitions and authorization rules.

The main goal of developing this framework is **to be able to edit any workflow item on the fly**. This means that all the elements in a workflow like states, transitions or authorizations rules are editable at any time so that no changes requires a re-deploying of your application anymore.

CHAPTER 1

Getting Started

You can easily get started with `django-river` by following *Getting Started*.

2.1 Getting Started

1. Install and enable it

```
pip install django-river
```

```
INSTALLED_APPS=[  
    ...  
    'river'  
    ...  
]
```

2. Create your first state machine in your model and migrate your db

```
from django.db import models  
from river.models.fields.state import StateField  
  
class MyModel(models.Model):  
    my_state_field = StateField()
```

3. Create all your states on the admin page
4. Create a workflow with your model (MyModel - my_state_field) information on the admin page
5. Create your transition metadata within the workflow created earlier, source and destination states
6. Create your transition approval metadata within the workflow created earlier and authorization rules along with their priority on the admin page
7. Enjoy your django-river journey.

```
my_model=MyModel.objects.get(...)  
  
my_model.river.my_state_field.approve(as_user=transactioner_user)
```

(continues on next page)

(continued from previous page)

```
my_model.river.my_state_field.approve(as_user=transactioner_user, next_  
↔state=State.objects.get(label='re-opened'))  
  
# and much more. Check the documentation
```

Note: Whenever a model object is saved, it's state field will be initialized with the state is given at step-4 above by django-river.

2.2 Overview

Main goal of developing this framework is **to be able to edit any workflow item on the fly**. This means, all elements in workflow like states, transitions, user authorizations(permission), group authorization are editable. To do this, all data about the workflow item is persisted into DB. **Hence, they can be changed without touching the code and re-deploying your application.**

There is ordering aprovmnts for a transition functionality in django-river. It also provides skipping specific transition of a specific objects.

Playground: There is a fake jira example repository as a playground of django-river. <https://github.com/javrasya/fakejira>

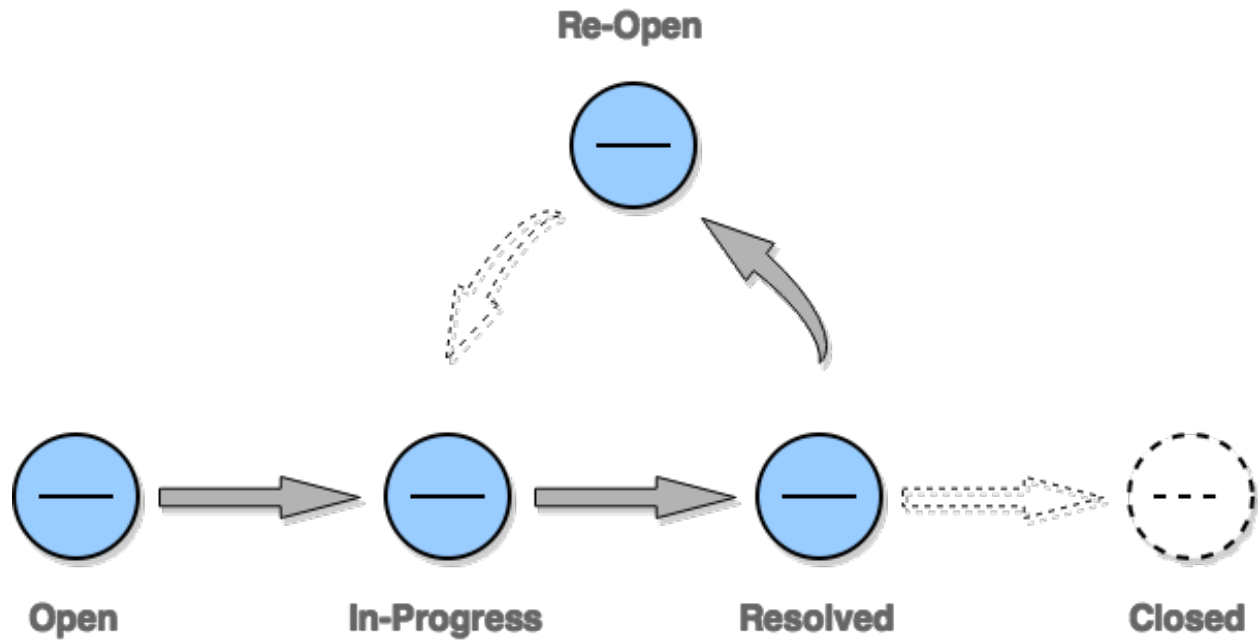
2.2.1 Requirements

- Python (2.7, 3.4, 3.5, 3.6)
- Django (1.11, 2.0, 2.1, 2.2)
- Django >= 2.0 is supported for Python >= 3.5

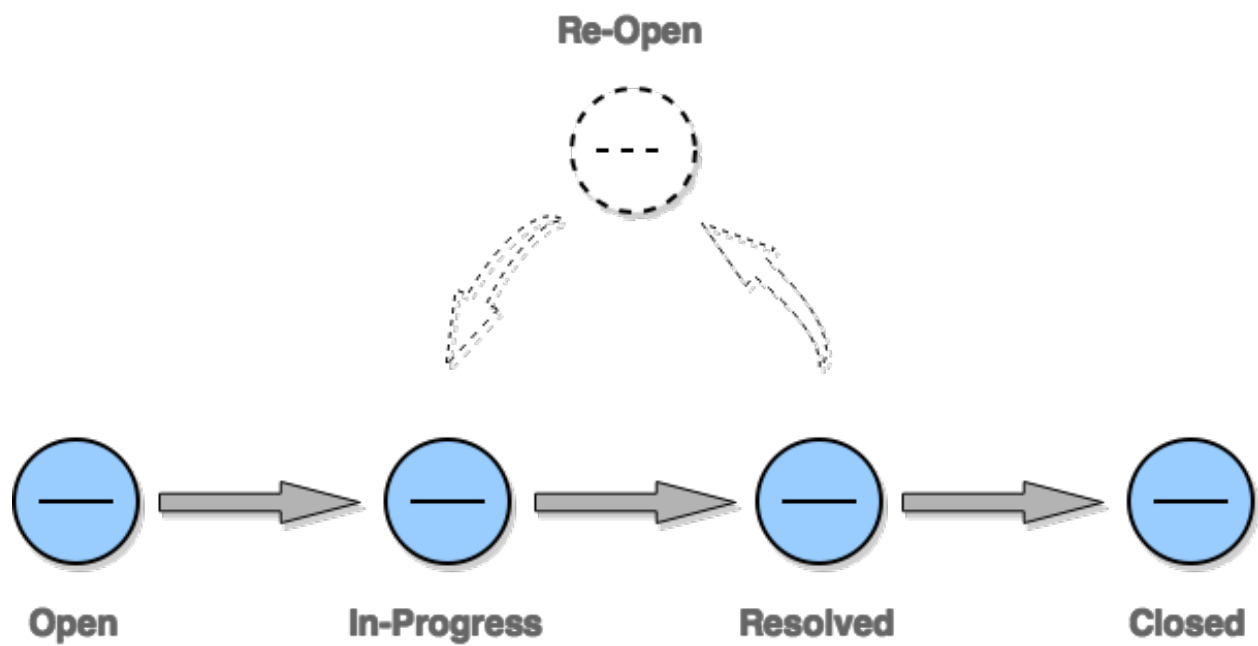
2.2.2 Example Scenarios

Simple Issue Tracking System

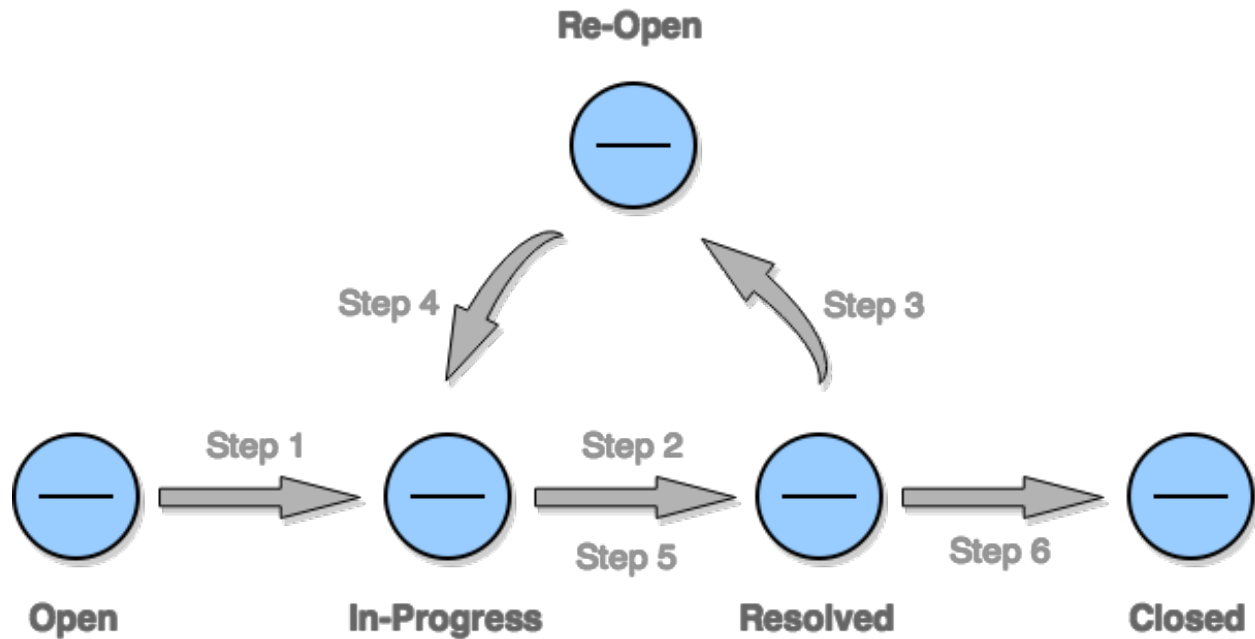
Re-Open case



Closed without Re-Open case



Closed with Re-Open case



2.3 Administration

Since `django-river` keeps all the data needed in a database, those should be pre-created before your first model object is created. Otherwise **your app will crash first time you create a model object**. Here are all needed models that you need to provide. `django-river` will register an Administration for those model for you. All you need to do is to provide them by using their Django admin pages.

2.3.1 State Administration

Field	Default	Optional	Format	Description
label	NaN	False	String (w+)	A name for the state
description	Empty string	True	String (w+)	A description for the state

2.3.2 Transition Meta Administration

Field	Default	Optional	Format	Description
workflow		False False	Choice of Strings	Your model class along with the field that you want to use this transition approval meta for. django-river will list all the possible model and fields you can pick on the admin page
source_state		False	State	Source state of the transition
destination_state		False	State	Destination state of the transition

2.3.3 Transition Approval Meta Administration

Field	Default	Optional	Format	Description
workflow		False	Choice of Strings	Your model class along with the field that you want to use this transition approval meta for. <code>django-river</code> will list all the possible model and fields you can pick on the admin page
transition_meta		False	TransitionMete	Transition information that contains source and destination states
permissions	Empty List	True	List<Permission>	List of permissions which will be authorized to approve this transition
groups	Empty List	True	List<UserGroup>	List of use groups which will be authorized to approve this transition
priority	0	False	Number	The priority of the transition approval. Since there can be more than one transition approval to make that transition which means that some users should approve before some other users can approve the same transition.
2.3. Administration				The closer to zero, the more priort the transition approval is.

2.4 API Guide

2.4.1 Class API

This page will be covering the class level API. It is all the function that you can access through your model class like in the example below;

```
>>> MyModel.river.my_state_field.<function>(*args)
```

get_on_approval_objects

This is the function that helps you to fetch all model objects waiting for a users approval.

```
>>> my_model_objects == MyModel.river.my_state_field.get_on_approval_objects(as_
↳user=team_leader)
True
```

	Type	Default	Optional	Format	Description
as_user	input	NaN	False	Django User	A user to find all the model objects waiting for a user's approvals
	Output			List<MyModel>	List of available my model objects

initial_state

This is a property that is the initial state in the workflow

```
>>> State.objects.get(label="open") == MyModel.river.my_state_field.initial_state
True
```

Type	Format	Description
Output	State	The initial state in the workflow

final_states

This is a property that is the list of final state in the workflow

```
>>> State.objects.filter(Q(label="closed") | Q(label="cancelled")) == MyModel.river.
↳my_state_field.final_states
True
```


Type	Format	Description
Output	List<State>	List of the final states in the workflow

2.4.2 Instance API

This page will be covering the instance level API. It is all the function that you can access through your model object like in the example below;

```
my_model=MyModel.objects.get(...)  
  
my_model.river.my_state_field.<function>(*args)
```

approve

This is the function that helps you to approve next approval of the object easily. `django-river` will handle all the availability and the authorization issues.

```
>>> my_model.river.my_state_field.approve(as_user=team_leader)  
>>> my_model.river.my_state_field.approve(as_user=team_leader, next_state=State.  
↳ objects.get(name='re_opened_state'))
```

	Type	Default	Optional	Format	Description
as_user	input	NaN	False	Django User	<p>A user to make the transaction.</p> <p>django-river will check if this user is authorized to make next action by looking at this user's permissions and user groups.</p>
next_state	input	NaN	True/False	State	<p>This parameter is redundant as long as there is only one next state from the current state. But if there is multiple possible next state in place,</p> <p>django-river is naturally is unable know which one is actually supposed to be picked. If the given next state is not a valid next state a <i>RiverException</i> will be thrown.</p>

get_available_approvals

This is the function that helps you to fetch all available approvals waiting for a specific user according to given source and destination states. If the source state is not provided, django-river will pick the current objects source state.

```
>>> transition_approvals = my_model.river.my_state_field.get_available_approvals(as_
↳user=manager)
>>> transition_approvals = my_model.river.my_state_field.get_available_approvals(as_
↳user=manager, source_state=State.objects.get(name='in_progress'))
>>> transition_approvals = my_model.river.my_state_field.get_available_approvals(
    as_user=manager,
    source_state=State.objects.get(name='in_progress'),
    destination_state=State.objects.get(name='resolved'),
)
```

	Type	Default	Optional	Format	Description
as_user	input	NaN	False	Django User	A user to find all the approvals by user's permissions and groups
source_state	input	Current Object's Source State	True	State	A base state to find all available approvals comes after. Default is current object's source state
destination_state	input	NaN	True	State	A specific destination state to fetch all available state. If it is not provided, the approvals will be found for all available destination states
	Output			List<TransitionApproval>	List of available transition approvals

recent_approval

This is a property that the transition approval which has recently been approved for the model object.

```
>>> transition_approval = my_model.river.my_state_field.last_approval
```

Type	Format	Description
Output	TransitionApproval	Last approved transition approval for the model object

next_approvals

This is a property that the list of transition approvals as a next step.

```
>>> transition_approvals == my_model.river.my_state_field.next_approvals
True
```

Type	Format	Description
Output	List<TransitionApproval>	List of transition approvals comes after last approved transition approval

on_initial_state

This is a property that indicates if object is on initial state.

```
>>> my_model.river.my_state_field.on_initial_state
True
```

Type	Format	Description
Output	Boolean	True if object is on initial state

on_final_state

This is a property that indicates if object is on final state.

```
>>> my_model.river.my_state_field.on_final_state
True
```

Type	Format	Description
Output	Boolean	True if object is on final state which also means that the workflow is complete

2.5 Authorization

`django-river` provides system users an ability to configure the authorizations at three level. Those are permissions, user group or a specific user at any step. If the user is not authorized, they are not entitled to see and approve those approvals. These three authorization mechanisms are also not blocking each other. An authorized user by any of them is entitled to see and approve the approvals.

2.5.1 Permission Based Authorization

Multiple permission can be specified on the *transition approval metadata* admin page and `django-river` will allow only the users who have the given permission. Given multiple permissions are issued in OR fashion meaning that it is enough to have one of the given permissions to be authorized for the user. This can be configurable on the admin page provided by *django-river*

2.5.2 User Group Based Authorization

Multiple user group can be specified on the *transition approval metadata* admin page and `django-river` will allow only the users who are in the given user groups. Like permission based authorization, given multiple user groups are issued in OR fashion meaning that it is enough to be in one of the given user groups to be authorized for the user. This can be configurable on the admin page provided by *django-river*

2.5.3 User Based Authorization

Only one specific user can be assigned and no matter what permissions the user has or what user groups the user is in, the user will be authorized. Unlike the other methods, `django-river` doesn't provide an admin interface for that. But this can be handled within the repositories that is using *django-river*. The way how to do this is basically setting the `transactioner` column of the related `TransitionApproval` object as the user who is wanted to be authorized on this approval either programmatically or through a third party admin page on this model.

2.6 Hooking Guide

2.6.1 Functions

Functions are the description in Python of what you want to do on certain events happen. So you define them once and you can use them with multiple hooking up. Just go to `/admin/river/function/` admin page and create your functions there. “`django-river`” function admin support python code highlighting as well if you enable the `codemirror2` app. Don't forget to collect statics for production deployments.

```
INSTALLED_APPS=[
    ...
    codemirror2
    river
    ...
]
```

Here is an example function;

```
from datetime import datetime

def handle(context):
    print(datetime.now())
```

Important: YOUR FUNCTION SHOULD BE NAMED AS `handle`. Otherwise `django-river` won't execute your function.

Context Parameter

`django-river` will pass a context down to your function in order for you to know why the function is triggered or for which object or so. And the `context` will look different for different type of events. But it also has some common parts for all the events. Let's look at how it looks;

`context.hook ->>`

Key	Type	Format	Description
type	String	<ul style="list-style-type: none"> * on-approved * on-transit * on-complete 	The event type that is hooked up. The payload will likely differ according to this value
when	String	<ul style="list-style-type: none"> * BEFORE * AFTER 	Whether it is hooked right before the event happens or right after
payload	dict		This is the context content that will differ for each event type. The information that can be gotten from payload is describe in the table below

Context Payload

On-Approved Event Payload

Key	Type	Description
workflow	Workflow Model	The workflow that the transition currently happening
workflow_object	Your Workflow Object	The workflow object of the model that has the state field in it
transition_approval	Transition Approval	The approval object that is currently approved which contains the information of the transition(meta) as well as who approved it etc.

On-Transit Event Payload

Key	Type	Description
workflow	Workflow Model	The workflow that the transition currently happening
workflow_object	Your Workflow Object	The workflow object of the model that has the state field in it
transition_approval	Transition Approval	The last transition approval object which contains the information of the transition(meta) as well as who last approved it etc.

On-Complete Event Payload

Key	Type	Description
workflow	Workflow Model	The workflow that the transition currently happening
workflow_object	Your Workflow Object	The workflow object of the model that has the state field in it

Example Function

```

from river.models.hook import BEFORE, AFTER

def _handle_my_transitions(hook):
    workflow = hook['payload']['workflow']
    workflow_object = hook['payload']['workflow_object']
    source_state = hook['payload']['transition_approval'].meta.source_state
    destination_state = hook['payload']['transition_approval'].meta.
    destination_state
    last_approved_by = hook['payload']['transition_approval'].transactioner
    if hook['when'] == BEFORE:
        print('A transition from %s to %s will soon happen on the object_
    with id:%s and field_name:%s!' % (source_state.label, destination_state.
    label, workflow_object.pk, workflow.field_name))
    elif hook['when'] == AFTER:
        print('A transition from %s to %s has just happened on the object_
    with id:%s and field_name:%s!' % (source_state.label, destination_state.
    label, workflow_object.pk, workflow.field_name))
        print('Who approved it lately is %s' % last_approved_by.username)

def _handle_my_approvals(hook):
    workflow = hook['payload']['workflow']
    workflow_object = hook['payload']['workflow_object']
    approved_by = hook['payload']['transition_approval'].transactioner
    if hook['when'] == BEFORE:
        print('An approval will soon happen by %s on the object with id:%s_
    and field_name:%s!' % ( approved_by.username, workflow_object.pk, workflow.
    field_name ))
    elif hook['when'] == AFTER:
        print('An approval has just happened by %s on the object with id:%s_
    and field_name:%s!' % ( approved_by.username, workflow_object.pk, workflow.
    field_name ))

def _handle_completions(hook):
    workflow = hook['payload']['workflow']
    workflow_object = hook['payload']['workflow_object']
    if hook['when'] == BEFORE:
        print('The workflow will soon be complete for the object with id:%s_
    and field_name:%s!' % ( workflow_object.pk, workflow.field_name ))
    elif hook['when'] == AFTER:
        print('The workflow has just been complete for the object with id:%s_
    and field_name:%s!' % ( workflow_object.pk, workflow.field_name ))

```

(continues on next page)

(continued from previous page)

```
def handle(context):
    hook = context['hook']
    if hook['type'] == 'on-transit':
        _handle_my_transitions(hook)
    elif hook['type'] == 'on-approved':
        _handle_my_approvals(hook)
    elif hook['type'] == 'on-complete':
        _handle_completions(hook)
    else:
        print("Unknown event type %s" % hook['type'])
```

2.6.2 Hook it Up

The hookings in `django-river` can be created both specifically for a workflow object or for a whole workflow. `django-river` comes with some model objects and admin interfaces which you can use to create the hooks.

- To create one for whole workflow regardless of what the workflow object is, go to
 - `/admin/river/onapprovedhook/` to hook up to an approval
 - `/admin/river/ontransithook/` to hook up to a transition
 - `/admin/river/oncompletehook/` to hook up to the completion of the workflow
- To create one for a specific workflow object you should use the admin interface for the workflow object itself. One amazing feature of `django-river` is now that

it creates a default admin interface with the hookings for your workflow model class. If you have already defined one, `django-river` enriches your already defined admin with the hooking section. It is default disabled. To enable it just define `RIVER_INJECT_MODEL_ADMIN` to be `True` in the `settings.py`.

Note: They can programmatically be created as well since they are model objects. If it is needed to be at workflow level, just don't provide the workflow object column. If it is needed to be for a specific workflow object then provide it.

Here are the list of hook models;

- `OnApprovedHook`
- `OnTransitHook`
- `OnCompleteHook`

2.7 Migration Guide

2.7.1 2.X.X to 3.0.0

`django-river` v3.0.0 comes with quite number of migrations, but the good news is that even though those are hard to determine kind of migrations, it comes with the required migrations out of the box. All you need to do is to run;

```
python manage.py migrate river
```

2.8 Change Logs

2.8.1 2.1.0 (Dev)

- **Bug** - # 106: It crashes when saving a workflow object when there is no workflow definition for a state field
- **Bug** - # 107: next_approvals api of the instance is broken
- **Bug** - # 112: Next approval after it cycles doesn't break the workflow anymore. Multiple cycles are working just fine.
- **Improvement** - # 108: Status column of transition approvals are now kept as string in the DB instead of number to maintain readability and avoid mistakenly changed ordinals.
- **Improvement** - # 109: Cancel all other peer approvals that are with different branching state.
- **Improvement** - # 110: Introduce an iteration to keep track of the order of the transitions even the cycling ones. This comes with a migration that will assess the iteration of all of your existing approvals so far. According to the tests, 250 workflow objects that have 5 approvals each will take ~1 minutes with the slowest django v1.11.
- **Improvement** - # 111: Cancel all approvals that are not part of the possible future instead of only impossible the peers when something approved and re-create the whole rest of the pipeline when it cycles
- **Improvement** - # 105: More dynamic and better way for hooks. On the fly function and hook creations, update or delete are also supported now. It also comes with useful admin interfaces for hooks and functions. This is a huge improvement for callback lovers :-)
- **Improvement** - # 113: Support defining an approval hook with a specific approval.
- **Improvement** - # 114: Support defining a transition hook with a specific iteration.
- **Drop** - # 115: Drop skipping and disabling approvals to cut the unnecessary complexity.
- **Improvement** - # 116: Allow creating transitions without any approvals. A new TransitionMeta and Transition models are introduced to keep transition information even though there is no transition approval yet.

2.8.2 2.0.0 (Stable)

- **Improvement** - [# 90, # 36]: Finding available approvals has been speeded up ~x400 times at scale
- **Improvement** - # 92 : It is mandatory to provide initial state by the system user to avoid confusion and possible mistakes
- **Improvement** - # 93 : Tests are revisited, separated, simplified and easy to maintain right now
- **Improvement** - # 94 : Support class level hooking. Meaning that, a hook can be registered for all the objects through the class api
- **Bug** - # 91 : Callbacks get removed when the related workflow object is deleted
- **Improvement** - Whole django-river source code is revisited and simplified
- **Improvement** - Support Django v2.2
- **Deprecation** - Django v1.7, v1.8, v1.9 and v1.10 supports have been dropped

2.8.3 1.0.2

- **Bug** - # 77 : Migrations for the models that have state field is no longer kept getting recreated.
- **Bug** - It is crashing when there is no workflow in the workspace.

2.8.4 1.0.1

- **Bug** - # 74 : Fields that have no transition approval meta are now logged correctly.
- **Bug** - django version is now fixed to 2.1 for coverage in the build to make the build pass

2.8.5 1.0.0

django-river is finally having it's first major version bump. In this version, all code and the APIs are revisited and are much easier to understand how it works and much easier to use it now. In some places even more performant. There are also more documentation with this version. Stay tuned :-)

- **Improvement** - Support Django2.1
- **Improvement** - Support multiple state fields in a model
- **Improvement** - Make the API very easy and useful by accessing everything via model objects and model classes
- **Improvement** - Simplify the concepts
- **Improvement** - Migrate ProceedingMeta and Transition into TransitionApprovalMeta for simplification
- **Improvement** - Rename Proceeding as TransitionApproval
- **Improvement** - Document transition and on-complete hooks
- **Improvement** - Document transition and on-complete hooks
- **Improvement** - Improve documents in general
- **Improvement** - Minor improvements on admin pages
- **Improvement** - Some performance improvements

2.8.6 0.10.0

- # 39 - **Improvement** - Django has dropped support for pypy-3. So, It should be dropped from django itself too.
- **Remove** - pypy support has been dropped
- **Remove** - Python3.3 support has been dropped
- **Improvement** - Django2.0 support with Python3.5 and Python3.6 is provided

2.8.7 0.9.0

- # 30 - **Bug** - Missing migration file which is 0007 because of Python2.7 can not detect it.
- # 31 - **Improvement** - unicode issue for Python3.
- # 33 - **Bug** - Automatically injecting workflow manager was causing the models not have default objects one. So, automatic injection support has been dropped. If anyone want to use it, it can be used explicitly.
- # 35 - **Bug** - This is huge change in django-river. Multiple state field each model support is dropped completely and so many APIs have been changed. Check documentations and apply changes.

2.8.8 0.8.2

- **Bug** - Features providing multiple state field in a model was causing a problem. When there are multiple state field, injected attributes in model class are overritten. This feature is also unpractical. So, it is dropped to fix the bug.
- **Improvement** - Initial video tutorial which is Simple jira example is added into the documentations. Also repository link of fakejira project which is created in the video tutorial is added into the docs.
- **Improvement** - No proceeding meta parent input is required by user. It is set automatically by django-river now. The field is removed from ProceedingMeta admin interface too.

2.8.9 0.8.1

- **Bug** - ProceedingMeta form was causing a problem on migrations. Accessing content type before migrations was the problem. This is fixed by defining choices in init function instead of in field

2.8.10 0.8.0

- **Deprecation** - ProceedingTrack is removed. ProceedingTracks were being used to keep any transaction track to handle even circular one. This was a workaround. So, it can be handled with Proceeding now by cloning them if there is circle. ProceedingTracks was just causing confusion. To fix this, ProceedingTrack model and its functions are removed from django-river.
- **Improvement** - Circular scenario test is added.
- **Improvement** - Admins of the workflow components such as State, Transition and ProceedingMeta are registered automatically now. Issue #14 is fixed.

2.8.11 0.7.0

- **Improvement** - Python version 3.5 support is added. (not for Django1.7)
- **Improvement** - Django version 1.9 support is added. (not for Python3.3 and PyPy3)

2.8.12 0.6.2

- **Bug** - Migration 0002 and 0003 were not working properly for postgresql (maybe oracle). For these databases, data can not be fixed. Because, django migrates each in a transactional block and schema migration and data migration can not be done in a transactional block. To fix this, data fixing and schema fixing are seperated.
- **Improvement** - Timeline section is added into documentation.
- **Improvement** - State slug field is set as slug version of its label if it is not given on saving.

2.8.13 0.6.1

- **Bug** - After content_type and field are moved into ProceedingMeta model from Transition model in version 0.6.0, finding initial and final states was failing. This is fixed.
- **Bug** - 0002 migrations was trying to set default slug field of State model. There was a unique problem. It is fixed. 0002 can be migrated now.

- **Improvement** - The way of finding initial and final states is changed. ProceedingMeta now has parent-child tree structure to present state machine. This tree structure is used to define the way. This requires to migrate 0003. This migration will build the tree of your existed ProceedingMeta data.

2.8.14 0.6.0

- **Improvement** - content_type and field are moved into ProceedingMeta model from Transition model. This requires to migrate 0002. This migrations will move value of the fields from Transition to ProceedingMeta.
- **Improvement** - Slug field is added in State. It is unique field to describe state. This requires to migrate 0002. This migration will set the field as slug version of label field value. (Re Opened -> re-opened)
- **Improvement** - State model now has natural_key as slug field.
- **Improvement** - Transition model now has natural_key as (source_state_slug , destination_state_slug) fields
- **Improvement** - ProceedingMeta model now has natural_key as (content_type, field, transition, order) fields
- **Improvement** - Changelog is added into documentation.

2.8.15 0.5.3

- **Bug** - Authorization was not working properly when the user has irrelevant permissions and groups. This is fixed.
- **Improvement** - User permissions are now retrieved from registered authentication backends instead of user . user_permissions

2.8.16 0.5.2

- **Improvement** - Removed unnecessary models.
- **Improvement** - Migrations are added
- **Bug** - content_type__0002 migrations cause failing for django1.7. Dependency is removed
- **Bug** - DatabaseHandlerBacked was trying to access database on django setup. This cause no table in db error for some django commands. This was happening; because there is no db created before some commands are executed; like makemigrations, migrate.

2.8.17 0.5.1

- **Improvement** - Example scenario diagrams are added into documentation.
- **Bug** - Migrations was failing because of injected ProceedingTrack relation. Relation is not injected anymore. But property proceeing_track remains. It still returns current one.

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`