Docflow Documentation

Release 0.2 beta

Kiran Jonnalagadda

Sep 27, 2017

Contents

This is the documentation for version 0.2, generated Sep 27, 2017.

Contents

CHAPTER 1

Introduction

Docflow is an implementation of document workflows in Python. A workflow defines *states* and *transitions*. A state is an "end point" associated with a document. A transition is a path from one state to another state (unidirectional). Docflow was inspired by repoze.workflow but aspires to be a framework-independent implementation.

A *document* in Docflow is any Python object. The document's workflow is defined separately from the document. This is a useful distinction in MVC or MTV-based web applications where the model is kept distinct from the view. The model is expected to be a dumb container of data while the view contains business logic. Workflows sit between view and models, controlling the view's view of the model.

CHAPTER 2

Usage

Documents can be defined like any other Python object:

```
>>> class MyDocument:
...     def __init__(self):
...         self.status = 0
...
... doc1 = MyDocument()
```

Documents can be simple dictionaries:

```
>>> doc2 = {'title': 'My Document', 'status': 0}
```

Or complex entities such as SQLAlchemy models:

```
>>> from sqlalchemy import Column, Integer, String
>>> from sqlalchemy.ext.declarative import declarative_base
>>> Base = declarative_base()
>>> class DatabaseDocument(Base):
...     __tablename__ = 'document'
...     id = Column(Integer, primary_key=True)
...     title = Column(String(200))
...     status = Column(Integer)
...
>>> doc3 = DatabaseDocument()
```

The important part is that the object must have an attribute or key that holds the current state, like `status` in all three examples. The state value can be any hashable Python object, typically an integer or string. The workflow defines all possible states for this document, the value that represents this state, and the transitions between states:

```
from docflow import DocumentWorkflow, WorkflowState, WorkflowStateGroup

class MyDocumentWorkflow(DocumentWorkflow):
    """
    Workflow for MyDocument
    """
```

```
    # Optional name for this workflow
    name = None

    # Attribute in MyDocument to store status in.
    # Use ``state_key`` if MyDocument is a dictionary,
    # as is typical with NoSQL JSON-based databases.
    state_attr = 'status'

    # Define a state. First parameter is the state tracking value,
    # stored in state_attr.
    draft     = WorkflowState(0, title="Draft",     description="Only owner can see it
↪")
    pending   = WorkflowState(1, title="Pending",   description="Pending review")
    published = WorkflowState(2, title="Published", description="Published")
    withdrawn = WorkflowState(3, title="Withdrawn", description="Withdrawn by owner")
    rejected  = WorkflowState(4, title="Rejected",  description="Rejected by reviewer
↪")

    # Define a state group (with either values or WorkflowState instances)
    not_published = WorkflowStateGroup([0, pending], title="Not Published")

    def permissions(self):
        """
        Return permissions available to current user. A permission can be any␣
↪hashable token.
        """
        if self.context and self.context.get('is_admin'):
            return ['can_publish']
        else:
            return []

    # Define a transition. There can be multiple transitions connecting any two␣
↪states.
    # Parameters: newstate, permission, title, description
    @draft.transition(pending, None, title='Submit')
    def submit(self):
        """
        Change workflow state from draft to pending.
        """
        # Do something here
        # ...
        pass  # State will be changed automatically if we don't raise an exception

    @pending.transition(published, 'can_publish', title="Publish")
    @withdrawn.transition(published, 'can_publish', title="Publish")
    def publish(self):
        """
        Publish the document.
        """
        # Also do something here
        # ...
        pass
```

Workflows can extend other workflows to add additional states:

```
class MyDocumentWorkflowExtraState(MyDocumentWorkflow):
    expired = WorkflowState(5, title="Expired")
```

Or override settings:

```
class MyDocumentWorkflowDict(MyDocumentWorkflow):
    state_attr = None
    state_key = 'status'
```

Workflows take an optional `context` parameter when being initialized with a document. This context is available to the `permissions()` method to determine if the caller has permission to make the transition. Once a workflow has been defined, usage is straightforward:

```
>>> wf = MyDocumentWorkflow(doc1)
>>> wf.state is wf.draft
True
>>> wf.submit() # Call a transition
>>> wf.state is wf.pending
True
>>> wf.draft() # Check if state is active
False
>>> wf.pending()
True
```

As a convenience mechanism, workflows can be linked to document classes, making it easier to retrieve the workflow for a given document:

```
class MyDocument(object):
    def __init__(self):
        self.status = 0

class MyDocumentWorkflow(DocumentWorkflow):
    state_attr = 'status'

MyDocumentWorkflow.apply_on(MyDocument)
```

After this, the workflow for a document becomes available with the `workflow` method:

```
doc = MyDocument()
wf = doc.workflow()
wf = doc.workflow('workflow-name')
```

The `workflow` method does not provide a way to supply a `context`, so it must be added later, if required:

```
wf = doc.workflow()
wf.context = context
```

The `apply_on()` method raises `WorkflowException` if the target class already has a workflow with the same name.

CHAPTER 3

API Documentation

Package `docflow`

class **`DocumentWorkflow`**(*document*)

The following attributes and methods must be overriden by subclasses of *`DocumentWorkflow`*.

`name`

The name of this workflow, default `None`. Workflows can be referred to by name when multiple workflows exist for a single document class.

`state_attr`

Refers to the attribute on the document that contains the state value. Default `None`.

`state_key`

If *`state_attr`* is `None`, *`state_key`* refers to the dictionary key in the document containing the state value. Default `None`.

`state_get`(*document*)

`state_set`(*document*, *value*)

If both *`state_attr`* and *`state_key`* are `None`, the *`state_get()`* and *`state_set()`* methods are called with the document as a parameter.

`state`

Currently active workflow state.

`permissions`([*context*])

Permissions available to caller in the given context, returned as a list of tokens.

`all_states`()

Standard method: returns a dictionary of all states in this workflow.

`transitions`([*context*])

Standard method: returns a dictionary of available transitions out of the current state.

apply_on (*docclass*)
Class method. Applies this workflow to the specified document class. The workflow can then be retrieved by calling the `workflow` method on the document.

class WorkflowState (*value*[, *title*, *description*])
Define a workflow state as an attribute on a *DocumentWorkflow*.

Parameters

- **value** – Value representing this workflow state. Can be any hashable Python object. Usually an integer or string
- **title** – Optional title for this workflow state
- **description** – Optional description for this workflow state

transition (*tostate*, *permission*[, *title*, *description*])
Decorator for a method on *DocumentWorkflow* that handles the transition from this state to another. The decorator will test for correct state and permission, and will transition the document's state if the method returns without raising an exception.

The method must take `context` as its first parameter. The context is passed to *permissions()* to check for permission.

Parameters

- **tostate** – Destination *WorkflowState*
- **permission** – Token representing permission required to call the transition. Must be present in the list returned by *permissions()*
- **title** – Optional title for this transition
- **description** – Optional description for this transition

Raises

- **WorkflowTransitionException** – If this transition is called when in some other state
- **WorkflowPermissionException** – If `permission` is not in *permissions()*

class WorkflowStateGroup (*values*[, *title*, *description*])
Like *WorkflowState* but lists more than one value. Useful to test for the current state being one of many. For example:

```
>>> class MyWorkflow(DocumentWorkflow):
...     draft = WorkflowState(0, title="Draft")
...     pending = WorkflowState(1, title="Pending")
...     published = WorkflowState(2, title="Published")
...     not_published = WorkflowStateGroup([0, pending], title="Not Published")
...
>>> wf = MyWorkflow(doc)
>>> wf.draft()
True
>>> wf.pending()
False
>>> wf.published()
False
>>> wf.not_published()
True
```

WorkflowStateGroup instances cannot have transitions.

Parameters

- **values** (*list*) – Status values or instances of *WorkflowState*
- **title** – Optional title for this workflow state
- **description** – Optional description for this workflow state

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index