Churro Documentation

Release 1.0a1

Chris Rossi

July 05, 2016

Contents

Churro is a simplistic persistent storage for Python objects which stores a tree of hierchically nested objects as folders and flat files in a filesystem. Churro uses AcidFS to provide ACID transaction semantics. Changes to any *Churro* object tree are only persisted when a transaction successfully commits. *Churro* uses JSON to serialize objects. *Churro* is meant to be lightweight and durable. Use of JSON, a universally understood and human readable text file format, insures that data stored by *Churro* is portable to other applications and platforms across space and time.

In addition to these docs, it couldn't hurt to look over the AcidFS documentation.

CHAPTER 1

Defining Persistent Types

In order for an object to be saved in a *Churro* repository, it must inherit from *Persistent* or *PersistentFolder*. Attributes of your persistent objects that you want to be persisted must be derived from *PersistentProperty*. Probably the best way to illustrate is by example, so let's say you're writing an application that saves contacts in an address book. We might write some code that looks like this:

```
from churro import Persistent
from churro import PersistentProperty
from churro import PersistentFolder

class AddressBook(PersistentFolder):
    title = PersistentProperty()

    def __init__(self, title):
        self.title = title

class Contact(Persistent):
    name = PersistentProperty()
    address = PersistentProperty()

    def __init__(self, name, address):
        self.name = name
        self.address = address
```

You can see that defining your persistent types is pretty straightforward. Next you'll want to open a repository and start storing some data.

CHAPTER 2

Adding Objects to the Repository

```
from churro import Churro

repo = Churro('/path/to/folder')
root = repo.root()
contacts = AddressBook('My Contacts')
root['contacts'] = contacts

contacts['fred'] = Contact('Fred Flintstone', '1 Rocky Road')
contacts['barney'] = Contact('Barney Rubble', '6 Bronto Lane')
```

Above, we create an instance of *Churro* where the argument is the folder in the filesystem where the repository will live. If the folder does not exist, it will be created and an empty repository will be initialized. Otherwise an existing repository will be opened. The call to *repo.root()* gets the root folder of the repository, the starting point for traversing to any other objects in the repository. From there, adding data to the repository is as easy as instantiating data objects using folders as Python dicts.

CHAPTER 3

Committing a Transaction

So far no data has actually been stored yet. You'll need to commit a transaction:

```
import transaction

transaction.commit()
```

Note: If you're using Pyramid, you should avoid committing the transaction yourself and use pyramid_tm. For other WSGI frameworks there is also repoze.tm2.

CHAPTER 4

Persistent Properties

PersistentProperty and its subclasses are responsible for serializing individual attributes of your Python objects to JSON. *PersistentProperty* can handle values of any type natively serializable to JSON. These include strings, booleans, numbers, lists, and dictionaries. Persistent properties can also hold as values other *Persistent* objects, allowing objects to be nested inside of each other.

Two additional property types, *PersistentDate* and *PersistentDatetime* are included for storing *datetime.date* and *datetime.datetime* objects respectively.

For other types you'll need to provide a means for converting the type to something serializable by JSON and then converting back to a Python object. This is done by extending *PersistentProperty* and overriding the *to_json()*, *from_json()*, and *validate()* methods. The following is an actual example from *Churro* code that illustrates this:

```
import datetime

class PersistentDate(PersistentProperty):

    def from_json(self, value):
        if value:
            return datetime.date(*map(int, value.split('-')))
        return value

    def to_json(self, value):
        if value:
            return '%s-%s-%s' % (value.year, value.month, value.day)
        return value

    def validate(self, value):
        if value is not None and not isinstance(value, datetime.date):
            raise ValueError("%s is not an instance of datetime.date")
        return value
```

You can use the new property type in your class definitions:

```
class Contact(Persistent):
    name = PersistentProperty()
    address = PersistentProperty()
    birthday = PersistentDate()

    def __init__(self, name, address):
        self.name = name
        self.address = address
```

CHAPTER 5

Mutable Property Values

Churro automatically keeps track of which objects have been mutated and saves those objects at transaction commit time. *Churro* does this by keeping track of when a setter is called on a property and marking that object as *dirty*. So simply assigning a value to a property will cause that object to get persisted at commit time:

```
daniela.birthday = datetime.date(2010, 5, 12)
```

You can find yourself in a situation, however, where the assigned value is a mutable structure and instead of assigning a new value to the property you simply mutate the structure. Let's say that we add a list of friends to our *Contact* class:

```
class Contact(Persistent):
    name = PersistentProperty()
    address = PersistentProperty()
    birthday = PersistentDate()
    friends = PersistentProperty()

    def __init__(self, name, address):
        self.name = name
        self.address = address
        self.friends = []
```

If we have a *Contact* instance that is *clean* and the only change we make is to add a friend to the list, *Churro* will not detect the mutation and the change will not be persisted at commit time:

```
# This change won't be persisted
daniela.friends.append('Katy')
```

One way to get around this problem is to call the *set_dirty()* method on the object that needs to be saved:

```
# Unless you call this method
daniela.set_dirty()
```

This brute force method is always available, whatever you're doing. *Churro* does, however, provide helpers for the two most common types of mutable data, dicts and lists. These are *PersistentDict* and *PersistentList* respectively. We could rewrite the example above to a use a `PersistentList` instead of a plain Python list:

```
from churro import PersistentList

class Contact(Persistent):
    name = PersistentProperty()
    address = PersistentProperty()
    birthday = PersistentDate()
    friends = PersistentProperty()

    def __init__(self, name, address):
```

```
        self.name = name
        self.address = address
        self.friends = PersistentList()
```

Now you don't need to call *set_dirty()* when adding a friend to a contact's friend list:

```
# Don't need to call set_dirty, this change will be persisted
daniela.friends.append('Silas')
```

CHAPTER 6

API Reference

class churro.**Churro**(*repo*, *head='HEAD'*, *factory=None*, *create=True*, *bare=False*)

Constructor Arguments

repo

The path to the repository in the real, local filesystem.

head

The name of a branch to use as the head for this transaction. Changes made using this instance will be merged to the given head. The default, if omitted, is to use the repository's current head.

factory

A callable that returns the root database object to be stored as the root when creating a new database. The default factory returns an instance of *churro.PersistentFolder*. This has no effect if the repository has already been created.

create

If there is not a Git repository in the indicated directory, should one be created? The default is *True*.

bare

If the Git repository is to be created, create it as a bare repository. If the repository is already created or *create* is False, this argument has no effect.

flush()

Writes any unsaved data to the underlying *AcidFS* filesystem without committing the transaction.

root()

Gets the root folder of the repository. This is the starting point for traversing to other objects in the repository.

class churro.**Persistent**

This is the base class from which all persistent classes for *Churro* must be derived. Only objects which are instances of a class derived from *Persistent* may be stored in a *Churro* repository.

deactivate()

Calling this method on a persistent object detaches that object, and its children, from the in memory persistent object tree, potentially allowing it to be garbage collected if there are no other references to the object.

set_dirty()

Calling this method alerts *Churro* that this object is *dirty* and should be persisted at commit time. It is usually not necesary to call this method from application code, since *Churro* tries to detect object mutation whenever possible. You may need to call this method from your application code, however, if you use

mutable data structures that are not themselves *Persistent* as values of persistent properties, as *Churro* has no way of detecting mutations to those structures.

class churro.**PersistentFolder**

Classes which derive from this class are not only persistent in *Churro* but have dict-like properties allowing them to contain children which are other persistent objects or folders. Storing an instance of *PersistentFolder* in a *Churro* repository, creates a folder in the underlying filesystem, in which child objects are stored. Instances of *PersistentFolder* are dict-like and are interacted with in the same way as standard Python dictionaries.

get(*name*, *default=None*)

Returns the child object of the given name. Returns *default* if the child is not found.

items()

Returns an iterator over (child object's name, child object) tuples.

keys()

Returns the names of child objects.

remove(*name*)

Removes the child with the given name from the folder. Raises *KeyError* if there is no child with the given name.

values()

Returns an iterator over child objects.

class churro.**PersistentDict**(**args*)

A *PersistentDict* is a Python *dict* work alike that marks its parent object as *dirty* whenever it is mutated, solving the problem of using mutable datastructures as values for persistent properties with *Churro* and eliminating the need to call *set_dirty()* in application code when updating the dictionary.

class churro.**PersistentList**(**args*)

A *PersistentList* is a Python *dict* work alike that marks its parent object as *dirty* whenever it is mutated, solving the problem of using mutable datastructures as values for persistent properties with *Churro* and eliminating the need to call *set_dirty()* in application code when updating the list.

class churro.**PersistentProperty**

The base type for all persistent properties. This property type can handle any data type as a value that is serializable natively to JSON. Other types are implemented by extending this class and overriding the *from_json*, *to_json*, and *validate* methods.

from_json(*value*)

Converts a value from its JSON representation to a Python object.

to_json(*value*)

Converts a value from a Python object to an object that can be serialized as JSON.

validate(*value*)

Used at assignment time to validate a value. If a value is not of the proper type and cannot be converted to the proper type, a *ValueError* is raised, otherwise the valua is returned, including any transformation or coercion that has been performed.

class churro.**PersistentDate**

A persistent attribute type that can store instances of *datetime.date*.

class churro.**PersistentDatetime**

A persistent attribute type that can store instances of *datetime.datetime*.

C

C

D

F

G

I

K

P

R

S

T

V