Universal Schema Documentation

Release 0.1

Bradley Arsenault

December 15, 2016

Contents

Contents:

CHAPTER 1

Introduction to Universal Schema

Universal Schema is a tool which allows you to express your data model once and get its schema represented in many different forms, and to subsequently convert between instances of those representations seamlessly. Its like an ORM designed to easily plug into other ORMS.

The problem is that we often need to express the same basic data model to multiple different libraries that do serialization in some form and sometimes in several different programming languages, such as in javascript pages with a Python backend. Similar validation routines must exist on both client and server side, and similar data models expressed for serialization and storage in backend services like task processors and databases.

Our documentation can be found at http://universal-schema.readthedocs.org/en/latest/

Universal Schema currently supports the following libraries:

Colander - schemas, data pymongo - data Ember.js-Data - schemas

Please feel free to contribute to the project and expand the number of supported libaries. Just fork on github and make a pull request when you are ready.

CHAPTER 2

Tutorial

To get started with Universal schema, first import the Model class and any number of field classes. Then define your models schema by subclassing from the Model baseclass, and defining the fields as variables within the account. This functions similar to most Python based ORM systems.:

```
from universal_schema import Model
from universal_schema.fields import DateTime, String, Binary, Integer, Email


class Account(Model):
    email = Email(min=1, max=1024)
    first_name = String(min=1, max=128)
    last_name = String(min=1, max=128)
    password_hash = String(min=1, max=128)
    current_balance = Integer()
    image = Binary()
    last_modified = DateTime()
```

Directly from the class object, you can generate schemas for alternate systems:

```
colander_schema = Account.schema("colander")
# This creates an instantiated Colander.Model object, which can be used to serialize and deserialize
# the way that colander does. By default, all variables are set as missing=colander.drop and default=
# unless another default is provided.
account_json = {'email' : 'awesome@gmail.com',
                'first_name' : "Usefuller",
                'last_name' : "Johnson",
                'current_balance' : 10.0,
                'last_modified' : '2014-05-01T22:15:38'}

deserialized = colander_schema.deserialize(account_json)
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search