XmlModels2 Documentation

Release 0.8.1

Geoff Ford

October 30, 2015

Contents

XmlModels allows you to define Models similar in nature to Django models that are backed by XML endpoints rather than a database. Using a familiar declarative definition, the fields map to values in the XML document by means of XPath expressions. With support for querying external REST APIs using a django-esque approach, we have strived to make writing and using xml backed models as close to django database models as we can, within the limitations of the available API calls.

CHAPTER 1

User Documentation

1.1 Mapping XML to Models

XML is mapped to `xml_models.Model` via `Fields`. Each field requires an xpath expression that determines which node or attribute to get the data from. Each field also has an optional `default` value for when no value can be retrieved from the XML.

1.1.1 Basic Fields

The available field mappings are

- `CharField` – returns string data
- `IntField` – returns integers
- `DateField` – returns a date from using the supplied `date_format` mask or the default ISO8601 format
- `FloatField` – returns a floating point number
- `BoolField` – returns a boolean
- `OneToOneField` – returns a `xml_model.Model` subclass
- `CollectionField` – returns a collection of either one of the above types, or an `xml_model.Model` subclass

Most of these fields are fairly self explanatory. The `CollectionField` and `OneToOneField` is where it gets interesting. This is what allows you to map instances or collections of nested entities, such as:-

```
<Person id="112">
  <firstName>Chris</firstName>
  <lastName>Tarttelin</lastName>
  <occupation>Code Geek</occupation>
  <website>http://www.pyruby.com</website>
  <contact-info>
    <contact type="telephone">
      <info>(555) 555-5555</info>
      <description>Cell phone, but no calls during work hours</description>
    </contact>
    <contact type="email">
      <info>me@here.net</info>
      <description>Where possible, contact me by email</description>
    </contact>
    <contact type="telephone">
```

```
      <info>1-800-555-5555</info>
      <description>Toll free work number for during office hours.</description>
    </contact>
  </contact-info>
</Person>
```

This can be mapped using a `Person` and a `ContactInfo` model:-

```
class Person(Model):
  id = IntField(xpath="/Person/@id")
  firstName = CharField(xpath="/Person/firstName")
  lastName = CharField(xpath="/Person/lastName")
  contacts = CollectionField(ContactInfo, order_by="contact_type", xpath="/Person/contact-info/conta

class ContactInfo(Model):
  contact_type = CharField(xpath="/contact/@type")
  info = CharField(xpath="/contact/info")
  description = CharField(xpath="/contact/description", default="No description supplied")
```

This leads to the usage of a person as :-

```
>>> person.contacts[0].info
me@here.com
```

1.1.2 Collections

When querying collections or lists, it is assumed that a collection of zero or more results are returned wrapped in an enclosing collection tag.

As some REST APIs may return lists wrapped in one or more layers of metadata, Models may also define a `collection_node` attribute. this allows the XML processor to find the relevant node.

Note: `collection_node` is the tag name only and not an xpath expression.

For example, given the following XML

```
<reponse status="200">
  <metadata count="2">
  <collection>
    <model ... />
    <model ... />
  </collection>
  </metadata>
</response>
```

We would need to define a Model with a `collection_node` like so

```
class SomeModel(Model):
  fieldA = CharField(xpath="/some/node")

  collection_node = 'collection'
```

1.1.3 Nested Collections

Similarly with Collections there may be a need where you have collections nested in metadata objects that are not relevant.

For example, given the following XML, you may only be interested in the Models. Rather than having to create a Collection model as well you can create a collection from the nested XML using the `collection_xpath` attribute.

```
<reponse status="200">
  <metadata count="2">
  <collection name="Collection1">
    <model ... />
    <model ... />
  </collection>
  <collection name="Collection2">
    <model ... />
    <model ... />
  </collection>
  </metadata>
</response>
```

```
class SomeModel(Model):
  fieldA = CharField(xpath="/model/some/node")

  collection_xpath = '//collection/model'
```

Note: `collection_xpath` will pass the enclosing tag XML to the Model. Therefore your models field definitions should start with the last tag name in the `collection_xpath` as the example does with the `model` tag.

Note: `collection_node` and `collection_xpath` are mutually exclusive

1.2 Querying Models

Querying is based on Django Model querying. Each `Model` has a class attribute `objects` which is the entry point for querying. However the querying ability is limited in scope and only supports very basic filtering.

To do anything interesting you will need to define finders on your models. There are no assumptions made about the nature of the REST API e.g. it is not even assumed that an `id` attribute can be queried.

1.2.1 Finders

An external REST api will present a limited number of options for querying data. Because the different options do not have to follow any specific convention, the model must define what finders are available and what parameters they accept. This still attempts to follow a Django-esque approach

```
class Person(xml_models.Model:
    ...
    finders = { (firstName, lastName): "http://person/firstName/%s/lastName/%s",
                (id,): "http://person/%s"}
```

The above defines two query options. The following code exercises these options

```
>>> people = Person.objects.filter(firstName='Chris', lastName='Tarttelin')
>>> people.count()
1
>>> person = Person.objects.get(id=123)
>>> person.firstName
Chris
```

Note: You can define a default finder using an empty tuple.

1.2.2 Self-signed HTTPS Endpoints

Self-signed, or endpoints signed by a non-standard CA, are supported by setting `xml_models.VERIFY` as per the requests documentation for **'SSL certificate verification <http://docs.python-requests.org/en/latest/user/advanced/#ssl-cert-verification'_**.

For example, to not verify a self-signed certificate you can use the following:

1.3 API Documentation

XmlModels is based on Django database models in terms of functionality and interface.

1.3.1 Manager

class `xml_models.xml_models.`**`ModelManager`**(*model*, *finders*)

Handles what can be queried for, and acts as the entry point for querying.

The API and usage is intended to be familiar to Django users however it does not support the complete Django ModelManager API.

`all`(***kw*)

Get all models.

Example

```
Model.objects.all()
```

How the actual HTTP request is handled is determined by *Finders*.

Parameters **`kw`** – optional key value pairs of field name and value

Returns lazy query

`count`()

Get a count

Returns int

`filter`(***kw*)

Filter models by key-value pairs.

Example

```
Model.objects.filter(attr1=value1,attr2=value2)
```

How the actual HTTP request is handled is determined by *Finders*.

Parameters **`kw`** – key value pairs of field name and value

Returns lazy query

`filter_custom`(*url*)

Set a URL to be called when querying

Parameters **`url`** – full URL

Returns lazy query

get(***kw*)
Get a single object.

This can be called directly with key-value pairs or after setting some `filters`

Parameters **kw** –

Returns

1.3.2 Model

class xml_models.xml_models.**Model**(*xml=None*, *dom=None*)
A model is a representation of the XML source, consisting of a number of Fields. It can be constructed with either an xml string, or an `etree.Element`.

Example

```
class Person(xml_models.Model):
    namespace="urn:my.default.namespace"
    name = xml_models.CharField(xpath"/Person/@Name", default="John")
    nicknames = xml_models.CollectionField(CharField, xpath="/Person/Nicknames/Name")
    addresses = xml_models.CollectionField(Address, xpath="/Person/Addresses/Address")
    date_of_birth = xml_models.DateField(xpath="/Person/@DateOfBirth", date_format="%d-%m-%Y")
```

If you define *Finders* on your model you will also be able to retreive models from an API endpoint using a familiar Django-esque object manager style of access with chainable filtering etc.

to_tree()
`etree.Element` representation of *Model*

Return type `lxml.etree.Element`

to_xml(*pretty=False*)
XML representation of Model

Return type string

validate_on_load()
Perform validation when the model is instantiated.

Override on your model to perform validation when the XML data is first passed in.

Note: You will need to raise appropriate exceptions as no checking of the return value occurs

1.3.3 Fields

class xml_models.xml_models.**BaseField**(***kw*)
Base class for Fields. Should not be used directly

class xml_models.xml_models.**CharField**(***kw*)
Returns the single value found by the xpath expression, as a string

parse(*xml*, *namespace*)

Parameters

- **xml** – the etree.Element to search in
- **namespace** – not used yet

Return type string

class xml_models.xml_models.**IntField**(***kw*)
Returns the single value found by the xpath expression, as an int

parse(*xml*, *namespace*)

Parameters

- **xml** – the etree.Element to search in
- **namespace** – not used yet

Return type DateTime, may be timezone aware or naive

class xml_models.xml_models.**FloatField**(***kw*)
Returns the single value found by the xpath expression, as a float

parse(*xml*, *namespace*)

Parameters

- **xml** – the etree.Element to search in
- **namespace** – not used yet

Return type float

class xml_models.xml_models.**DateField**(*date_format=None*, ***kw*)
Returns the single value found by the xpath expression, as a `datetime`.

By default, expects dates that match the ISO8601 date format. If a `date_format` keyword arg is supplied, that will be used instead. `date_format` should conform to `strptime` formatting options.

If the XML contains UTC offsets then a timezone aware datetime object will be returned.

parse(*xml*, *namespace*)

Parameters

- **xml** – the etree.Element to search in
- **namespace** – not used yet

Return type DateTime, may be timezone aware or naive

class xml_models.xml_models.**OneToOneField**(*field_type*, ***kw*)
Returns a subclass of *Model* from the xpath expression.

parse(*xml*, *namespace*)

Parameters

- **xml** – the etree.Element to search in
- **namespace** – not used yet

Return type as defined by `self.field_type`

class xml_models.xml_models.**CollectionField**(*field_type*, *order_by=None*, ***kw*)
Returns a collection found by the xpath expression.

Requires a field_type to be supplied, which can either be a field type, e.g. *IntField*, which returns a collection ints, or it can be a *Model* type e.g. Person may contain a collection of Address objects.

parse(*xml*, *namespace*)
Find all nodes matching the xpath expression and create objects from each the matched node.

If `order_by` has been defined then the resulting list will be ordered.

Parameters

- **xml** – the etree.Element to search in
- **namespace** – not used yet

Return type as defined by `self.field_type`

CHAPTER 2

Installation

The simplest approach is to to use `pip install xml_models2`

CHAPTER 3

A simple example

Just to get started, this is an example of taking an XML representation of an Address that might be returned from a GET request to an external REST api.

```
<Address id="2">
  <number>22</number>
  <street>Acacia Avenue</street>
  <city>Maiden</city>
  <country>England</country>
  <postcode>IM6 66B</postcode>
</Address>

class Address(xml_models.Model):
  id=xml_models.IntField(xpath="/Address/@id")
  number = xml_models.IntField(xpath="/Address/number")
  street = xml_models.CharField(xpath="/Address/street")
  city = xml_models.CharField(xpath="/Address/city")
  country = xml_models.CharField(xpath="/Address/country")
  postcode = xml_models.CharField(xpath="/Address/postcode")

  finders = {(id,): 'http://adresses/%s'}
```

This example would be used as follows:-

```
>>> address = Address.objects.get(id=2)
>>> print "address is %s, %s" % (address.number, address.street)
"22, Acacia Avenue"
```

CHAPTER 4

Heritage

This project is a fork of [Django REST Models](http://djangorestmodel.sourceforge.net/)

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

X