xmodels Documentation

Release 0.1.0

Bernd Meyer

November 02, 2014

Contents

CHAPTER 1

xmodels

For more information, please see our documentation: http://xmodels.readthedocs.org/en/latest/

CHAPTER 2

Overview

xmodels is a Python library to convert XML documents and Python dictionaries including collections.OrderedDict to and from instances of Python data structures and validate them against a schema. The conversions between XML and Python dict/collections.OrderedDict are performed by xmltodict. It supports XML specific schema features like ordered sets of element type **<xsd:sequence>**, selection from a set of element type **<xsd:choice>** and namespaces.

Contents:

2.1 Installation

At the command line:

```
$ easy_install xmodels
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv xmodels
$ pip install xmodels
```

2.2 Usage

To use Python xmodels in a project:

```
import xmodels
```

2.3 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

2.3.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/berndca/xmodels/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Python xmodels could always use more documentation, whether as part of the official Python xmodels docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/berndca/xmodels/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.3.2 Get Started!

Ready to contribute? Here's how to set up *xmodels* for local development.

1. Fork the *xmodels* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/xmodels.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv xmodels
   $ cd xmodels/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 xmodels tests
   $ python setup.py test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

2.3.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/berndca/xmodels/pull_requests and make sure that the tests pass for all supported Python versions.

2.4 Credits

2.4.1 Development Lead

- Bernd Meyer <berndca@gmail.com>

2.4.2 Contributors

None yet. Why not be the first?

2.5 History

2.5.1 0.1.0 (2014-11-02)

- First release on PyPI.

2.6 Fields

The fields module defines various field classes all of which are derived from BaseField.

2.6.1 Field Methods

BaseField.**validate**(*raw_data*, ***kwargs*)
The validate method validates raw_data against the field .

Parameters **raw_data** (*str or other valid formats*) – raw data for field

Returns validated_data

Raises ValidationException if self.required and raw_data is None

BaseField.**deserialize**(*raw_data*, ***kwargs*)

BaseField.**serialize**(*py_data*, ***kwargs*)

class xmodels.fields.**BaseField**(***kwargs*)
Base class for all field types.

The `source` parameter sets the key that will be retrieved from the source data. If `source` is not specified, the field instance will use its own name as the key to retrieve the value from the source data.

The `serial_format` parameter controls the serialization format, e.g. in DateTimeField etc.

A default value can be assigned through the `default` parameter.

Parameters

- **kwargs['required']** (*bool*) – indicates required field
- **kwargs['default']** (*str*) – default value, used when raw_data is None
- **kwargs['serial_format']** (*str*) – format string for serialization and deserialization
- **kwargs['source']** (*str*) – field name for serialized version

class xmodels.fields.**RegexField**(***kwargs*)
Field to represent unicode strings matching a regular expression. It raises ValidationException if there is no match. :param regex: regular expression to match.

class xmodels.fields.**RangeField**(***kwargs*)
Base class for IntegerField and FloatField. :param int/float kwargs['min']: indicates minimum allow value (inclusive). :param int/float kwargs['max']: indicates maximum allow value (inclusive).

2.6.2 Basic Fields

BooleanField

class xmodels.**BooleanField**(***kwargs*)
Field to represent a boolean. The string `'True'` (case insensitive) will be converted to `True`, as will any positive integers and the boolean value `True`.

```
>>> from xmodels import BooleanField
>>> BooleanField().validate('TRUE')
True
>>> BooleanField().validate('not true!')
False
>>> BooleanField().validate(42)
True
>>> BooleanField().validate(-3)
False
>>> BooleanField().validate(True)
True
```

CharField

class xmodels.**CharField**(***kwargs*)
Field to represent a simple Unicode string value.

```
>>> from xmodels import CharField
>>> char_field = CharField()
>>> char_field.validate(' valid unicode string!\n')
'valid unicode string!'
```

class xmodels.**Token**(***kwargs*)
CharField for xsd:token. Tokens are strings without leading and trailing whitespaces. All other whitespaces are collapsed.

class xmodels.**Name**(***kwargs*)
Field for xsd:name. Values of this type must start with a letter, underscore (_), or colon (:), and may contain only letters, digits, underscores (_), colons (:), hyphens (-), and periods (.). Colons should only be used to separate namespace prefixes from local names.

```
>>> from xmodels import Name
>>> name_field = Name()
>>> name_field.validate('valid_name')
'valid_name'
```

class xmodels.**NCName**(***kwargs*)
Field for xsd:ncname. The type NCName represents an XML non-colonized name, which is simply a name that does not contain colons. An NCName must start with either a letter or underscore (_) and may contain only letters, digits, underscores (_), hyphens (-), and periods (.). This is identical to the Name type, except that colons are not permitted.

class xmodels.**Language**(***kwargs*)
Field for xsd:language. The type language represents a natural language identifier, generally used to indicate the language of a document or a part of a document. Before creating a new attribute of type language, consider using the xml:lang attribute that is intended to indicate the natural language of the element and its content. Values of the language type conform to RFC 3066, Tags for the Identification of Languages, in version 1.0 and to RFC 4646, Tags for Identifying Languages, and RFC 4647, Matching of Language Tags, in version 1.1. The three most common formats are: For ISO-recognized languages, the format is a two- or three-letter (usually lowercase) language code that conforms to ISO 639, optionally followed by a hyphen and a two-letter, usually uppercase, country code that conforms to ISO 3166. For example, en or en-US. For languages registered by the Internet Assigned Numbers Authority (IANA), the format is i-langname, where langname is the registered name. For example, i-navajo. For unofficial languages, the format is x-langname, where langname is a name of up to eight characters agreed upon by the two parties sharing the document. For example, x-Newspeak. Any of these three formats may have additional parts, each preceded by a hyphen, which identify more countries or dialects. Schema processors will not verify that values of the language type conform to the above rules. They will simply deserialize them based on the pattern specified for this type, which says that it must consist of one or more parts of up to eight characters each, separated by hyphens.

class xmodels.**NMTOKEN**(***kwargs*)
Field for xsd:NMTOKEN. The type NMTOKEN represents a single string token. NMTOKEN values may consist of letters, digits, periods (.), hyphens (-), underscores (_), and colons (:). They may start with any of these characters. NMTOKEN has a whitespace facet value of collapse, so any leading or trailing whitespace will be removed. However, no whitespace may appear within the value itself.

class xmodels.**IntegerField**(***kwargs*)
Field to represent an integer value.

class xmodels.**NonNegativeInteger**(***kwargs*)
Field to represent a non negative integer value.

class xmodels.**PositiveInteger**(***kwargs*)
Field to represent a positive integer value.

class xmodels.**NegativeInteger**(***kwargs*)
Field to represent a negative integer value.

class xmodels.**FloatField**(***kwargs*)
Field to represent a floating point value. The serial_format uses the standard string format notation with the surrounding curly brackets.

class xmodels.**NonNegativeFloat**(***kwargs*)
Field to represent a non negative floating point value.

class xmodels.**EnumField**(***kwargs*)
Tests that the value is one of the members of a given list (options). There can be no empty strings in options. value has to be a string.

If matchLower is True it will also compare value.lower() with the lower case version of all strings in options.

2.6.3 Datetime Fields

class xmodels.**DateTimeField**(***kwargs*)
Field to represent a datetime

The `format` parameter dictates the format of the input strings, and is used in the construction of the `datetime.datetime` object.

The `serial_format` parameter is a strftime formatted string for serialization. If `serial_format` isn't specified, an ISO formatted string will be returned by `to_serial()`.

class xmodels.**DateField**(***kwargs*)
Field to represent a `datetime.date`

class xmodels.**TimeField**(***kwargs*)
Field to represent a `datetime.time`

2.6.4 Relationship Fields

class xmodels.**ModelField**(*wrapped_class*, ***kwargs*)
Field containing a model instance

Use this field when you wish to nest one object inside another. It takes a single required argument, which is the nested class. For example, given the following dictionary:

```
some_data = {
    'first_item': 'Some value',
    'second_item': {
        'nested_item': 'Some nested value',
    },
}
```

You could build the following classes (note that you have to define the inner nested models first):

```
class MyNestedModel(xmodels.Model):
    nested_item = xmodels.CharField()

class MyMainModel(xmodels.Model):
    first_item = xmodels.CharField()
    second_item = xmodels.ModelField(MyNestedModel)
```

class xmodels.**ModelCollectionField**(*wrapped_class*, ***kwargs*)
Field containing a list of model instances.

Use this field when your source data dictionary contains a list of dictionaries. It takes a single required argument, which is the name of the nested class that each item in the list should be converted to. For example:

```
some_data = {
    'list': [
        {'value': 'First value'},
        {'value': 'Second value'},
        {'value': 'Third value'},
    ]
}

class MyNestedModel(xmodels.Model):
    value = xmodels.CharField()

class MyMainModel(xmodels.Model):
    list = xmodels.ModelCollectionField(MyNestedModel)
```

class xmodels.**FieldCollectionField**(*field_instance*, ***kwargs*)
Field containing a list of the same type of fields.

The constructor takes an instance of the field.

Here are some examples:

```
data = {
            'legal_name': 'John Smith',
            'aliases': ['Larry', 'Mo', 'Curly']
}

class Person(Model):
    legal_name = CharField()
    aliases = FieldCollectionField(CharField())

p = Person(data)
```

And now a quick REPL session:: FIXME doctest

Here is a bit more complicated example involving args and kwargs:

```
data = {
            'name': 'San Andreas',
            'dates': ['1906-05-11', '1948-11-02', '1970-01-01']
}

class FaultLine(Model):
    name = CharField()
    earthquake_dates = FieldCollectionField(DateField('%Y-%m-%d',
                                                      serial_format='%m-%d-%Y'),
                                                      source='dates')

f = FaultLine(data)
```

Notice that source is passed to to the `FieldCollectionField`, not the `DateField`.

Let's check out the resulting `Model` instance with the

2.7 Models

class xmodels.models.**Model**

The Model is the main component of xmodels.models. It is the base class for AttributeModel and SequenceModel implementing common logic.

Usually one defines a number of fields as class variables. Fields may have default values. All fields can be assigned and read. A read to a field which has not been set previously return the default value if one exists or None. The validate method validates all fields defined as class variables.

Instance variables may be created in addition to the fields specified as class variables if Meta.allow_extra_elements is True. Otherwise the model validation fails. The validation results are stored in a logger instance.

class xmodels.models.**SequenceModel**

The SequenceModel is used to describe xml elements using xsd:sequence. The sequence is described with a list of SequenceElement in Meta.sequence. The validation method checks if all required attributes and min_occurs > 0 elements are not None.

The initial class variable is used for context initialization for identity constraints checking.

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

X