datatree Documentation

Release 0.2.alpha.1

Jason Webb

February 25, 2012

CONTENTS

CHAPTER

ONE

SUMMARY

DataTree is a library for creating structured documents in python. Inspired by ruby builder but with the goal of reducing the amount of line noise while remaining "pythonic". As an added bonus the tree can be output to to any structured format (with XML, JSON and YAML supported in the library).

Note: More documentation is coming soon but for now a very basic rough draft can be found at datatree.readthedocs.org.

CHAPTER

TWO

INSTALLATION

You can install via PyPi or direct from the github repository.

To install with pip:

```
$ pip install datatree
```

To install with easy_install:

```
$ easy_install datatree
```

CHAPTER

THREE

ABOUT THE 0.2.0 ALPHA

Version 0.2.0 is about 70% complete. The remaining tasks are:

1. Implement xml namespaces.
2. Finish updating documentation to the new API.
3. Make the `Node` callable the same as the `node` method.

3.1 Installing the Alpha

You can install the alpha with pip directly from github using the command:

```
$ pip install -e git://github.com/bigjason/datatree.git@v0.2.alpha.1#egg=datatree
```

CHAPTER

FOUR

EXAMPLE

A small example:

```
from datatree import Tree, Node

tree = Tree()
with tree.node("author") as author:
    author.node('name', 'Terry Pratchett')
    author.node('genre', 'Fantasy/Comedy')
    author.comment("Only 2 books listed")
    with author.node('novels', count=2) as novels:
        novels.node('novel', 'Small Gods', year=1992)
        novels.node('novel', 'The Fifth Elephant', year=1999)
        novels.node("novel", "Guards! Guards!", year=1989)

print tree(pretty=True)
```

Which produces the XML:

```
<author>
    <name>Terry Pratchett</name>
    <genre>Fantasy/Comedy</genre>
    <!-- Only 2 books listed -->
    <novels count="2">
        <novel year="1992">Small Gods</novel>
        <novel year="1999">The Fifth Elephant</novel>
        <novel year="1989">Guards! Guards!</novel>
    </novels>
</author>
```

Or the JSON:

```
{
    "author": {
        "genre": "Fantasy/Comedy",
        "name": "Terry Pratchett",
        "novels": [
            "Small Gods",
            "The Fifth Elephant",
            "Guards! Guards!"
        ]
    }
}
```

Or the YAML:

```
author:
  genre: Fantasy/Comedy
  name: Terry Pratchett
  novels: [Small Gods, The Fifth Elephant, Guards! Guards!]
```

4.1 License

4.2 Souce Code

The source code can be found on github.

4.3 Feedback

I welcome any and all constructive feedback. Feel free to contact me (Jason Webb) at www.bigjason.com or (preferably) on twitter @bigjasonwebb.

4.4 Contributing

Contributions are welcome. Just fork on github and I will try to be as responsive as possible.

CHAPTER

FIVE

CONTENTS:

5.1 Basic Usage

5.1.1 Building Data Trees

Working with datatree begins with the `Tree` class:

```
tree = Tree()
```

To add a new node to the `tree` you simply call a function with the name of the new node that you wish to add. So to add a `root` node to the tree you could:

```
tree.root()
```

All nodes are also context managers. This allows for very logical building of the tree like so:

```
with tree.root() as root:
    root.name("Bob Smith")
```

There are some various options when adding nodes, although most of them are only relevent if you are creating XML. When adding a node, any `**kwargs` argument passed in is converted to an attribute:

```
with tree.root() as root:
    root.name("Bob Smith", gender='Male')
```

5.1.2 Emitting Formatted Data

Renderers are used to output your datatree into a dataformat. These are simple classes that translate the `Tree` into a specific format. By default XML is used. However an alias can be used for a different format. Additionaly all of the `**kwargs` are used to pass specific options to the renderer. To output pretty XML:

```
tree = Tree()
with tree.root() as root:
    root.name("Bob Smith", full=True)
print tree('xml', pretty=True)
```

Outputs:

```
<root>
    <name full="True">Bob Smith</name>
</root>
```

The aliases for the formats are as follows:

Renderer	Alias(s)
XML	xml, [blank]
JSON	json, jsn
YAML	yaml, yml
Python dict	dict, dictionary

5.2 API

5.2.1 Tree

You can call (*almost*) any method name on the `Tree` to create a new Node.

class `datatree.`**`Tree`**(**args*, ***kwargs*)

Very top node in a datatree.

The Tree is the top node used to build a datatree.

`cdata`(*text*, ***attributes*)

Add a `CDataNode` to the current tree.

Note: CData tags are ignored by some of the renderers such as json and dict. Consult the documentation to find out the behaviour.

Parameters

- **text** – Text value of the CDATA tag.
- **attributes** – Additional attributes to be added to the tag.

Returns The created `CDataNode`.

`comment`(*text*)

Adds a comment to the node.

Note: Comments are ignored by some of the renderers such as json and dict. Consult the documentation to find out the behaviour.

Parameters **text** – Text content of the comment.

`declare`(*name*, **attributes*)

Add an xml declaration to the datatree.

Note: Declarations are ignored by some of the renderers such as json and dict. Consult the documentation to find out the behaviour.

Warning: This functionality is pretty limited for the time being, hopefully the API for this will become more clear with time.

Parameters

- **name** – Name of the declaration node.

- **attributes** – Extra attributes to be added. Strings will be added as quoted strings. Symbols will be added as unquoted strings. Import the ___ object and use it like this: `___.SomeValue` to add a symbol.

instruct (*name='xml'*, ***attributes*)
Add an xml processing instruction.

Note: Instructions are ignored by some of the renderers such as json and dict. Consult the documentation to find out the behaviour.

Parameters

- **name** – Name of the instruction node. A value of xml will create the instruction `<?xml ?>`.
- **attributes** – Any extra attributes for the instruction.

node (*name*, *value=None*, ***attributes*)
Creates and adds `Node` object to the current tree.

Parameters

- **name** – The name identifier for the node..
- **value** – The value of this node. Note that this will be converted to a string usually during rendering. Default is `None` which generally means it is ignored during rendering.
- **attributes** – The key value pairs that will be used as the attributes for the node.

Returns The created `Node`.

static **register_renderer** (*klass*)
Register a renderer class with the datatree rendering system.

Parameters **klass** – Either a string with the fully qualified name of the renderer class to register, or the actual class itself. The name will be read from the class.

render (*renderer='xml'*, *as_root=False*, ***options*)
Render the datatree using the provided renderer.

Parameters

- **renderer** – The name of the renderer to use. You may add more renderers by using the register_renderer method.
- **as_root** – If True, the tree will be rendered from this node down, otherwise rendering will happen from the tree root.
- **options** – Key value pairs of options that will be passed to the renderer.

to_string (*level=0*)
Create an ugly representation of the datatree from this node down.

Warning: This is included as a debug aid and is not good for much else. The output is messy and inconsistent.

5.2.2 Node

Node is not instantiated directly, but is created for every node added to the `Tree`.

class datatree.tree.**Node**(*node_name='root'*, *node_value=None*, ***attributes*)
A node is able to be instantiated directly and added to any Vertex.

cdata(*text*, ***attributes*)
Add a CDataNode to the current tree.

Note: CData tags are ignored by some of the renderers such as json and dict. Consult the documentation to find out the behaviour.

Parameters

- **text** – Text value of the CDATA tag.
- **attributes** – Additional attributes to be added to the tag.

Returns The created CDataNode.

comment(*text*)
Adds a comment to the node.

Note: Comments are ignored by some of the renderers such as json and dict. Consult the documentation to find out the behaviour.

Parameters **text** – Text content of the comment.

node(*name*, *value=None*, ***attributes*)
Creates and adds Node object to the current tree.

Parameters

- **name** – The name identifier for the node..
- **value** – The value of this node. Note that this will be converted to a string usually during rendering. Default is None which generally means it is ignored during rendering.
- **attributes** – The key value pairs that will be used as the attributes for the node.

Returns The created Node.

static register_renderer(*klass*)
Register a renderer class with the datatree rendering system.

Parameters **klass** – Either a string with the fully qualified name of the renderer class to register, or the actual class itself. The name will be read from the class.

render(*renderer='xml'*, *as_root=False*, ***options*)
Render the datatree using the provided renderer.

Parameters

- **renderer** – The name of the renderer to use. You may add more renderers by using the register_renderer method.
- **as_root** – If True, the tree will be rendered from this node down, otherwise rendering will happen from the tree root.
- **options** – Key value pairs of options that will be passed to the renderer.

to_string(*level=0*)
Create an ugly representation of the datatree from this node down.

> **Warning:** This is included as a debug aid and is not good for much else. The output is messy and inconsistent.

5.2.3 Renderers

The renderers are responsible for converting the datatree into a usable format. Usually this format is a string, but sometimes other formats are used.

The examples in this section use this datatree:

```
from datatree import Tree, Node

tree = Tree()
with tree.node("author") as author:
    author.node('name', 'Terry Pratchett')
    author.node('genre', 'Fantasy/Comedy')
    author.comment("Only 2 books listed")
    with author.node('novels', count=2) as novels:
        novels.node('novel', 'Small Gods', year=1992)
        novels.node('novel', 'The Fifth Elephant', year=1999)
        novels.node("novel", "Guards! Guards!", year=1989)
```

XmlRenderer

Outputs the tree as an xml string. It is available under the alias `'xml'`.

Options

Name	Description	Default
pretty	When True, Outputs the xml document with pretty formatting.	`False`
indent	Used with pretty formatting. It is the string that will be used to indent each level.	`' '`

Example Output

```
tree('xml', pretty=True)
```

Or even shorter:

```
tree(pretty=True)
```

```
<author>
    <name>Terry Pratchett</name>
    <genre>Fantasy/Comedy</genre>
    <!-- Only 2 books listed -->
    <novels count="2">
        <novel year="1992">Small Gods</novel>
        <novel year="1999">The Fifth Elephant</novel>
        <novel year="1989">Guards! Guards!</novel>
    </novels>
</author>
```

JsonRenderer

Outputs the tree as json string using the python json module. It is available under the alias `'json'`, `'jsn'` or `'js'`.

Options

Name	Description	Default
pretty	Outputs the json document with pretty formatting.	`False`
sort_keys	Sorts the keys in the json document.	`False`

Example Output

```
tree('json', pretty=True)
```

```
{
    "author": {
        "genre": "Fantasy/Comedy",
        "name": "Terry Pratchett",
        "novels": [
            "Small Gods",
            "The Fifth Elephant",
            "Guards! Guards!"
        ]
    }
}
```

YamlRenderer

Outputs the tree as yaml string using the PyYAML package (which must be installed). It is available under the alias `'yaml'` or `'yml'`.

Options

Name	Description	Default
None		

Example Output

```
tree('yaml')
```

```
author:
  genre: Fantasy/Comedy
  name: Terry Pratchett
  novels: [Small Gods, The Fifth Elephant, Guards! Guards!]
```

DictRenderer

Outputs the tree as python dict. It is available under the alias `'dict'` and `'dictionary'`.

Options

Name	Description	Default
pretty_string	When True, outputs the `dict` as a string with pretty formatting.	`False`
allow_node_loss	Determines if a duplicate node name will result in a node loss due to duplicate keys in the dict.	`False`

Example Output

```
tree('dict', pretty_string=True)

{'author': {'genre': 'Fantasy/Comedy',
            'name': 'Terry Pratchett',
            'novels': ['Small Gods', 'The Fifth Elephant', 'Guards! Guards!']}}
```

Duplicate Node Names

While xml handles duplicate nodes just fine, python dicts and json for that matter do not allow duplicates. To handle this the DictRenderer will attempt to group nodes with the same name into a sub dictionary. This is why in the above example there is only one key for "novels".

Implementing a Renderer

You can implement your own renderer. Just look at the source for one of the existing renderers and implement the same methods, and then register your plugin with the `register_renderer()` method.

5.3 Change Log

5.3.1 0.2.alpha.1

Release Date *TBD*

- Change of API removing *Magic* methods.
- Documentation updates.
- More tests and small bug fixes.

5.3.2 0.1.8.1

Release Date 06-26-2011

- Changed license to Apache License, Version 2.0.
- Added coverage.py to the test suite along with some testing updates.

5.3.3 0.1.8

Release Date 06-21-2011

- `JsonRenderer` pretty option now defaults to `False`.
- Documentation updates.
- Bug fixes.

5.3.4 0.1.7

Release Date 06-16-2011

- Any node in a data tree can now be used to render the entire tree.
- More documentation.
- Bug fixes.

5.3.5 0.1.6

Release Date 06-16-2011

- Fix xml pretty rendering bad xml.
- More documentation.

5.3.6 0.1.5

Release Date 06-15-2011

- Support for python 2.6.
- Bug fixes.

5.3.7 0.1.1

Release Date 06-15-2011

- Renamed S class to n and removed the SubNode class.
- Bug fixes.

5.3.8 0.1

Release Date 06-15-2011

- First stable release.

5.4 Planned Features

- Support for xml namespaces.
- Addition of an HTML renderer.
- Full unicode support.
- More operator overloads (jury still out on this one).
- A solution for including attributes in the DictRenderer, which will trickle down to JsonRenderer and YamlRenderer.
- A more unified and maintainable test suite.
- Python 3.2 support.

CHAPTER

SIX

INDICES AND TABLES

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

PYTHON MODULE INDEX

d