vtkxml-builder Documentation

Release 0.1.0

George Lesica

May 05, 2013

CONTENTS

CHAPTER

ONE

TUTORIAL

This is a quick run-through of how to use vtkxml-builder to create a set of VTU files from a set of Python lists.

So, say we have a set of particles that we want to visualize using ParaView or something similar to that. Each particle has a three-tuple coordinate, a radius and a temperature.

```
>>> x = [1.5, 1.0, 2.5]
>>> y = [0.5, 3.0, 2.0]
>>> z = [2.5, 1.5, 3.0]
>>> r = [1.0, 2.0, 1.5]
>>> t = [0.0, 0.5, 0.5]
```

So now we have three lists to represent the position. The particles are located at (1.5, 0.5, 2.5), (1.0, 3.0, 1.5), and (2.5, 2.0, 3.0). Their radii are 1.0, 2.0, and 1.5, and their temperatures are 0.0, 0.5, and 0.5.

Now we want to actually write some VTU files. We instantiate a `VtuWriter`, then call its `write_vtu_file()` method and pass in our data. We use the *scalars* parameter to pass in our scalar data series (radius and temperature) and the *vectors* parameter to pass in our vectors. We also specify a file name and, optionally, data types (if you don't specify types they will be guessed). Also, note that we need to provide the name of the data series to be treated as positions. This is a special category in VTK, so it needs to be included. The default is "positions", so if you use that name, you don't have to specify it.

```
>>> writer = VtuWriter()
>>> scalars = {'radius': r, 'temp': t}
>>> vectors = {'position': [x, y, z]}
>>> writer.write_vtu_file('vtu0.vtu', scalars, vectors, pname='position')
```

This will write a VTU file called "vtu0.vtu" to the current directory. Successive calls to `write_vtu_file()` can then be used to create additional files.

CHAPTER

TWO

MODULE DOCUMENTATION

2.1 vtkxml

This module provides tools for creating VTK XML files for use with ParaView and other similar data visualization tools. It is meant to be used within a separate post-processing script to turn raw data output into an XML file.

Todo

Add writer for other grid types

---

class vtkxml.**VtuWriter**(*bit64=False*)

Parameters **bit64** – If true, use 64 bit value strings.

Writer for VTK unstructured grid XML files. The primary public methods are `write_data_file()` and `write_pvd_file()`.

Usage example:

```
>>> import tempfile, os
>>> td = tempfile.gettempdir()
>>> w= VtuWriter()
>>> v = {'positions': [[1, 2.5, 3], [2, 1, 3], [3.5, 1, 2]]}
>>> s = {'temps': [1.0, 2.0, 1.5]}
>>> w.write_data_file(os.path.join(td, 'vtu0.vtu'), s, v)
```

Todo

Add support for cell and vert data in addition to point data

---

generate_scalar(*doc*, *name*, *data*, *datatype*)

Parameters

- **doc** – An *xml.dom.minidom.Document* object.
- **name** – Name of the data series.
- **data** – List of data series values.
- **datatype** – Data type to use for data series.

Generates a `<DataArray>` XML element that contains scalar data.

generate_vector(*doc*, *name*, *data*, *datatype*)

Parameters

- **doc** – An *xml.dom.minidom.Document* object.
- **name** – Name of the data series.
- **data** – List of data series values.
- **datatype** – Data type to use for data series.

Generates a `<DataArray>` XML element that contains vector data.

guess_scalar_type(*data*)

Parameters **data** – A list of scalar data values.

Guess the data type of the series. Assumes 32 bit values unless *bit64* was specified to the constructor. If a valid type cannot be found, a *ValueError* exception is raised.

```
>>> w32 = VtuWriter()
>>> w32.guess_scalar_type([1,2,3])
'UInt32'
>>> w32.guess_scalar_type([-1,2,3])
'Int32'
>>> w32.guess_scalar_type([1.0, 2.0, 3.0])
'Float32'
>>> w32.guess_scalar_type([1.0, 2, 3.0])
'Float32'
>>> w64 = VtuWriter(bit64=True)
>>> w64.guess_scalar_type([1,2,3])
'UInt64'
>>> w64.guess_scalar_type([-1,2,3])
'Int64'
>>> w64.guess_scalar_type([1.0, 2.0, 3.0])
'Float64'
>>> w64.guess_scalar_type([1.0, 2, 3.0])
'Float64'
```

guess_vector_type(*data*)

Parameters **data** – A list of vector element lists. Each list contains the data for a particular element of a vector.

Guess the data type of the vector series. Behaves the same as *guess_scalar_type*.

```
>>> w32 = VtuWriter()
>>> w32.guess_vector_type([[1,2,3], [1,2,3]])
'UInt32'
>>> w32.guess_vector_type([[-1,2,3], [1,2,3]])
'Int32'
>>> w32.guess_vector_type([[1.0, 2.0, 3.0], [1.0, 2.0, 3.0]])
'Float32'
>>> w32.guess_vector_type([[1.0, 2, 3.0], [1, 2, 3]])
'Float32'
>>> w64 = VtuWriter(bit64=True)
>>> w64.guess_vector_type([[1,2,3], [1,2,3]])
'UInt64'
>>> w64.guess_vector_type([[-1,2,3], [1,2,3]])
'Int64'
>>> w64.guess_vector_type([[1.0, 2.0, 3.0], [1.0, 2.0, 3.0]])
'Float64'
>>> w64.guess_vector_type([[1.0, 2, 3.0], [1, 2, 3]])
'Float64'
```

static vectors_to_string (**vectors*)

Parameters ***vectors** – A list of lists, each with an element of a vector.

Convert a list of one or more vectors into a string, where each line contains one element from each of the vectors. See the example below. Note that the order of the elements is preserved.

```
>>> print VtuWriter.vectors_to_string([1,2,3], [4,5,6])
1 4
2 5
3 6
>>> print VtuWriter.vectors_to_string([1,2,3])
1
2
3
```

write_data_file (*fn*, *scalars*, *vectors*, *types={}*, *pname='positions'*)

Parameters

- **fn** – Filename to save XML document to, including extension.
- **scalars** – Dictionary of scalar data series.
- **vectors** – Dictionary of vector data series.
- **types** – Dictionary of type strings.
- **pname** – Positions vector series name.

Create an XML file with filename *fn* that contains the data given. Data are passed in using the *scalars* and *vectors* parameters. These are dictionaries of the form

```
{<field_name>: [<elements_1>, <elements_2>,...,<elements_n>], ...}
```

for vectors and

```
{<field_name>: <elements>, ...}
```

for scalars, where <*elements*> and <*elements_i*> are lists of data values. So, for example, a bunch of particle positions might be passed in using a *vectors* dictionary such as

```
{'positions': [[1,2,3], [1,2,3], [1,2,3]]}
```

which would correspond to particles located at (1, 1, 1), (2, 2, 2), (3, 3, 3). Similarly, two scalar variables could be created using a *scalars* argument like the one given below.

```
{'radius': [3, 2, 5], 'mass': [5, 8, 3]}
```

In this case, the particle located at (1, 1, 1) in the *vectors* example would have radius *3* and mass *5*, etc.

Data types may optionally be provided for each data series using the *types* parameter. Vector and scalars types are provided in a single dictionary in which the keys are the field names from the *vector* and *scalar* arguments and the values are valid VTK data type strings such as *UInt32* or *Float32*. Any data series that do not have types provided will have their types auto-detected.

Finally, the *pname* parameter may be used to specify the name of a vector data series that should be treated as points in the unstructured grid. The default is *positions*. A points vector is required.

write_pvd_file (*fn*)

Parameters **fn** – Filename to save PVD file to.

Write a PVD file with filename *fn* that contains references to the data files previously created with this instance.

CHAPTER

THREE

TODO LIST

Todo

Add writer for other grid types

(The *original entry* is located in /var/build/user_builds/vtkxml-builder/checkouts/latest/vtkxml/vtkxml.py:docstring of vtkxml, line 8.)

Todo

Add support for cell and vert data in addition to point data

(The *original entry* is located in /var/build/user_builds/vtkxml-builder/checkouts/latest/vtkxml/vtkxml.py:docstring of vtkxml.VtuWriter, line 15.)

CHAPTER

FOUR

INDICES

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

PYTHON MODULE INDEX