PVGeo Documentation

Bane Sullivan

Aug 08, 2018

Contents

This website hosts the code documentation for the `PVGeo` python package found on GitHub and PyPI. This website strictly documents the code so that users have a convenient and familiar means of searching through the library to understand the backend of the features they are using. If you are searching for examples and demonstrations on how to use the `PVGeo` library, then head over to the full website where you can find tutorials with sample data sets and links to many other helpful resources.

CHAPTER 1

Requesting Features, Reporting Issues, and Contributing

Please feel free to post features you would like to see from this package on the issues page as a feature request. If you stumble across any bugs or crashes while using code distributed here, please report it in the issues page so we can promptly address it. For other questions please join the PVGeo community on Slack.

CHAPTER 2

About the Authors

The PVGeo code library is managed by Bane Sullivan, graduate student in the Hydrological Science and Engineering interdisciplinary program at the Colorado School of Mines under Whitney Trainor-Guitton. If you would like to contact us, please inquire with info@pvgeo.org.

It is important to note the project is open source and that many features in this repository were made possible by contributors volunteering their time. Please take a look at the Contributors Page to learn more about the developers of PVGeo.

CHAPTER 3

Getting Started

To begin using the PVGeo python package, create a new virtual environment and install PVGeo through pip.

```
$ conda create -n PVGeoEnv python=2.7

# Install VTK through conda as this is OS-independent
$ conda install -n PVGeoEnv vtk

$ source activate PVGeoEnv
(PVGeoEnv) $ pip install PVGeo

# Test the install on non-Windows OS
(PVGeoEnv) $ python -m PVGeo test
```

Now PVGeo is ready for use in your standard python environment. To use the *PVGeo* library as plugins in ParaView, please see the detailed explanation here.

CHAPTER 4

Current Statistics

macro	writer	filter	source	base	reader
3	5	21	4	7	9

4.1 Base Classes

4.1.1 Algorithm Base

class PVGeo.base.**AlgorithmBase**(*nInputPorts=1*, *inputType='vtkDataSet'*, *nOutputPorts=1*, *outputType='vtkTable'*)

Bases: vtk.util.vtkAlgorithm.VTKPythonAlgorithmBase

This is a base class to add convienace methods to the VTKPythonAlgorithmBase for all algorithms implemented in PVGeo. We implement our algorithms in this manner to harness all of the backend support that the VTKPythonAlgorithmBase class provides for integrating custom algorithms on a VTK pipeline. All of the pipeline methods for setting inputs, getting outputs, making requests are handled by the super classes. For more information on what functionality is available, check out the VTK Docs for the vtkAlgorithm and then check out the following blog posts:

- vtkPythonAlgorithm is great
- A VTK pipeline primer (part 1), (part 2), and (part 3)

Apply()
Update the algorithm and get the output data object

ErrorMessage()
A conveience method to print the error message.

ErrorOccurred()
A conveience method for handling errors on the VTK pipeline

Returns true if an error has ovvured since last checked

Return type bool

GetOutput(*port=0*)
A conveience method to get the output data object of this PVGeo algorithm.

4.1.2 Filter Preserve Type Base

class PVGeo.base.**FilterPreserveTypeBase**
Bases: PVGeo.base.FilterBase

A Base class for implementing filters that preserve the data type of their arbitrary input.

RequestDataObject(*request*, *inInfo*, *outInfo*)
There is no need to overwrite this. This method lets the pipeline know that the algorithm will dynamically decide the output data type based in the input data type.

4.1.3 Reader Base

class PVGeo.base.**ReaderBase**(*nOutputPorts=1*, *outputType='vtkTable'*, ***kwargs*)
Bases: *PVGeo.base.AlgorithmBase*

A base class for inherrited functionality common to all reader algorithms

AddFileName(*fname*)
Use to set the file names for the reader. Handles singlt string or list of strings. :param fname: The absolute file name with path to read. :type fname: str

Apply(*fname*)
Given a file name (or list of file names), perfrom the read

ClearFileNames()
Use to clear file names of the reader.

Note:

- This does not set the reader to need to read again as there are no files to read.

GetFileNames(*idx=None*)
Returns the list of file names or given and index returns a specified timestep's filename.

GetTimestepValues()
Use this in ParaView decorator to register timesteps on the pipeline.

Modified(*readAgain=True*)
Call modified if the files needs to be read again again

NeedToRead(*flag=None*)
Ask self if the reader needs to read the files again.

Parameters **flag** (*bool*) – Set the read status

Returns the status of the reader.

Return type bool

RequestInformation(*request*, *inInfo*, *outInfo*)
This is a conveience method that should be overwritten when needed. This will handle setting the timesteps appropriately based on the number of file names when the pipeline needs to know the time information.

SetTimeDelta(*dt*)
An advanced property to set the time step in seconds.

_GetFileContents(*idx=None*)

_GetRawData(*idx=0*)

_ReadUpFront()

_UpdateTimeSteps()
For internal use only: appropriately sets the timesteps.

4.1.4 Two File Reader Base

class PVGeo.base.**TwoFileReaderBase**(*nOutputPorts=1*, *outputType='vtkUnstructuredGrid'*, ***kwargs*)
Bases: *PVGeo.base.AlgorithmBase*

A base clase for readers that need to handle two input files. One meta-data file and a series of data files.

AddModelFileName(*fname*)
Use to set the file names for the reader. Handles single string or list of strings.

Parameters **fname** (*str or list(str)*) – the file name(s) to use for the model data.

Apply()
Perfrom the read with parameters/file names set during init or by setters

ClearMesh()
Use to clear mesh file name

ClearModels()
Use to clear data file names

GetMeshFileName()

GetModelFileNames(*idx=None*)
Returns the list of file names or given and index returns a specified timestep's filename.

GetTimestepValues()
Use this in ParaView decorator to register timesteps

static HasModels(*modelfiles*)
A convienance method to see if a list contatins models filenames.

Modified(*readAgainMesh=True*, *readAgainModels=True*)
Call modified if the files needs to be read again again

Parameters

- **readAgainMesh** (*bool*) – set the status of the reader for mesh files.
- **readAgainModels** (*bool*) – set the status of the reader for model files.

NeedToReadMesh(*flag=None*)
Ask self if the reader needs to read the mesh file again.

Parameters **flag** (*bool*) – set the status of the reader for mesh files.

NeedToReadModels(*flag=None*)
Ask self if the reader needs to read the model files again.

Parameters **flag** (*bool*) – set the status of the reader for model files.

RequestInformation(*request*, *inInfo*, *outInfo*)
Overwritten by subclass to provide meta-data to downstream pipeline.

SetMeshFileName(*fname*)
Set the mesh file name.

SetTimeDelta(*dt*)
An advanced property for the time step in seconds.

ThisHasModels()
Ask self if the reader has model filenames set.

_TwoFileReaderBase__UpdateTimeSteps()
For internal use only

4.1.5 Writer Base

class PVGeo.base.**WriterBase**(*nInputPorts=1*, *inputType='vtkPolyData'*, ***kwargs*)
Bases: *PVGeo.base.AlgorithmBase*

Apply(*inputDataObject*)
Update the algorithm and get the output data object

GetFileName()
Get the set filename.

RequestData(*request*, *inInfoVec*, *outInfoVec*)
OVERWRITE: This is executed by the pipeline and handles the write out

SetFileName(*fname*)
Specify the filename for the output. Writer can only handle a single output data object/time step.

Write(*inputDataObject=None*)
Perfrom the write out.

4.2 Test Runner

4.2.1 test

PVGeo.tester.**test**(*close=False*)
This is a convienance method to run all of the tests in PVGeo while in an active python environment.

Note: This can be executed from either the command line of within a standard Python environment.

Example

```
# From the command line
$ python -m PVGeo test
```

```
>>> # From an active Python environment
>>> import PVGeo
>>> PVGeo.test()
```

4.3 Version Verifier

4.3.1 checkNumpy

PVGeo.version.**checkNumpy**(*warn=True*)
A method to check the active environments version of NumPy for compatibility with PVGeo.

Parameters **war** (*bool*) – raise a RuntimeWarning if NumPy is not at satisfactory version.

4.4 Internal Helpers

4.4.1 arrays

addArray

PVGeo._helpers.arrays.**addArray**(*pdo*, *field*, *vtkArray*)
Adds an array to a vtkDataObject given its field association.

Parameters

- **pdo** (*vtkDataObject*) – the output data object
- **field** (*int*) – the field type id
- **vtkArray** (*vtkDataArray*) – the data array to add to the output

Returns the output data object with the data array added

Return type vtkDataObject

copyArraysToPointData

PVGeo._helpers.arrays.**copyArraysToPointData**(*pdi*, *pdo*, *field*)
Copys arrays from an input to an ouput's point data.

Parameters

- **pdi** (*vtkDataObject*) – The input data object to copy from
- **pdo** (*vtkDataObject*) – The output data object to copy over to
- **field** (*int*) – the field type id

Returns returns the output data object parameter

Return type vtkDataObject

getNumPyArray

PVGeo._helpers.arrays.**getNumPyArray**(*wpdi*, *field*, *name*)
Grabs an array from vtkDataObject given its name and field association.

Parameters

- **wpdi** (*wrapped vtkDataObject*) – the input data object wrapped using vtk dataset adapter

- **field** (*int*) – the field type id
- **name** (*str*) – the name of the input array for the given index

Returns a wrapped `vtkDataArray` for NumPy

Return type numpy.array

getSelectedArray

`PVGeo._helpers.arrays.`**`getSelectedArray`**(*algorithm*, *wpdi*, *idx*)

Gets selectected array at index idx wrapped for NumPy

Parameters

- **algorithm** (*vtkAlgorithm*) – A vtkAlgorithm class instantiation
- **wpdi** (*wrapped vtkDataObject*) – the input data object wrapped using vtk dataset adapter
- **idx** (*int*) – the input array index

Returns a wrapped `vtkDataArray` for NumPy

Return type numpy.array

getSelectedArrayField

`PVGeo._helpers.arrays.`**`getSelectedArrayField`**(*algorithm*, *idx*)

Gets the field of the input array for a given index on a VTK algorithm

Parameters

- **algorithm** (*vtkAlgorithm*) – A vtkAlgorithm class instantiation
- **idx** (*int*) – the input array index

Returns the field type of the input array for the given index

Return type int

getSelectedArrayName

`PVGeo._helpers.arrays.`**`getSelectedArrayName`**(*algorithm*, *idx*)

Gets the name of the input array for a given index on a VTK algorithm

Parameters

- **algorithm** (*vtkAlgorithm*) – A vtkAlgorithm class instantiation
- **idx** (*int*) – the input array index

Returns the name of the input array for the given index

Return type str

getVTKArray

PVGeo._helpers.arrays.**getVTKArray**(*pdi*, *field*, *name*)

Grabs an array from vtkDataObject given its name and field association.

Parameters

- **pdi** (*vtkDataObject*) – the input data object
- **field** (*int*) – the field type id
- **name** (*str*) – the name of the input array for the given index

Returns the array from input field and name

Return type vtkDataArray

numToVTK

PVGeo._helpers.arrays.**numToVTK**(*arr*, *name*)

Converts a 1D numpy array to a VTK data array given a nameself.

Parameters

- **arr** (*np.array*) – A 1D numpy array
- **name** (*str*) – the name of the data array for VTK

Returns a converted data array

Return type vtkDataArray

4.4.2 errors

ErrorObserver

class PVGeo._helpers.errors.**ErrorObserver**

A class for catching errors when processing on a VTK pipeline. The `AlgorithmBase` class handles setting up this observer on initialization.

Example

```
>>> import PVGeo
>>> # Only use this observer on sub classes of the AlgorithmBase:
>>> f = PVGeo.AlgorithmBase()
>>> f.Update()
>>> if f.ErrorOccurred():
>>>     print(f.ErrorMessage())
ERROR: ...
```

ErrorMessage(*etc=False*)

Get the last set error message

Returns the last set error message

Return type str

ErrorOccurred()
Ask self if an error has occured

MakeObserver(*algorithm*)
Make this an observer of an algorithm

PVGeoError

class PVGeo._helpers.errors.**PVGeoError**(*message*)
Bases: exceptions.Exception

This is a custom error class for handling errors when proccessing on the VTK pipeline. It makes the error messages easy to decipher in ParaView and cleans the messages when used in Python outside of ParaView. When on the VTK pipeline, errors aren't really raised but passed over and printed to the console. This class makes decipher the error streams a whole lot easier for human eyes.

static CleanMessage(*message*)

QUALIFIER_L = '@@@@PVGeoError ---> '

QUALIFIER_R = ' <--- PVGeoError@@@@'

SEARCHER = <_sre.SRE_Pattern object at 0x26d98a0>

4.4.3 readers

These are helpers specifically for the file readers for private use only. @author: Bane Sullivan

cleanDataNm

PVGeo._helpers.readers.**cleanDataNm**(*dataNm*, *FileName*)
A helper to clean a FileName to make a useful data array name

getVTKtype

PVGeo._helpers.readers.**getVTKtype**(*typ*)
This looks up the VTK type for a give python data type.

Returns the integer type id specified in vtkType.h

Return type int

getdTypes

PVGeo._helpers.readers.**getdTypes**(*dtype=''*, *endian=None*)
This converts char dtypes and an endian to a numpy and VTK data type.

Returns the numpy data type and the integer type id specified in vtkType.h for VTK data types

Return type tuple (numpy.dtype, int)

placeArrInTable

PVGeo._helpers.readers.**placeArrInTable**(*ndarr*, *titles*, *pdo*)
Takes a 1D/2D numpy array and makes a vtkTable of it

Parameters

- **ndarr** (*numpy.ndarray*) – The 1D/2D array to be converted to a table
- **titles** (*list or tuple*) – The titles for the arrays in the table. Must have same number of elements as columns in input ndarray
- **pdo** (*vtkTable*) – The output data object pointer

Returns returns the same input pdo table

Return type vtkTable

4.4.4 timeseries

GetInputTimeSteps

PVGeo._helpers.timeseries.**GetInputTimeSteps**(*algorithm*, *port=0*, *idx=0*)
Get the timestep values for the algorithm's input

Parameters

- **algorithm** (*vtkDataObject*) – The data object (Proxy) on the pipeline (pass *self* from algorithm subclasses)
- **port** (*int*) – the input port
- **idx** (*int*) – optional : the connection index on the input port

Returns the time step values of the input

Return type list

GetRequestedTime

PVGeo._helpers.timeseries.**GetRequestedTime**(*algorithm*, *outInfoVec*, *idx=0*)
Handles setting up the timesteps on on the pipeline for a file series reader.

Parameters

- **algorithm** (*vtkDataObject*) – The data object (Proxy) on the pipeline (pass *self* from algorithm subclasses)
- **outInfoVec** (*vtkInformationVector*) – The output information for the algorithm
- **idx** (*int*) – the index for the output port

Returns the index of the requested time

Return type int

Example

```
>>> # Get requested time index
>>> i = _helpers.GetRequestedTime(self, outInfoVec)
```

UpdateTimeSteps

PVGeo._helpers.timeseries.**UpdateTimeSteps**(*algorithm*, *nt*, *dt*)
Handles setting up the timesteps on on the pipeline for a file series reader.

Parameters

- **algorithm** (*vtkDataObject*) – The data object (Proxy) on the pipeline (pass *self* from algorithm subclasses)
- **nt** (*int or list*) – Number of timesteps (Pass a list to use length of that list)
- **dt** (*float*) – The discrete value in seconds for the time step.

Returns Returns the timesteps as an array

Return type numpy.array

4.4.5 xml

_helpArraysXml

PVGeo._helpers.xml.**_helpArraysXml**(*idx*, *inputName=None*, *label=None*)
Internal helper

getDropDownXml

PVGeo._helpers.xml.**getDropDownXml**(*name*, *command*, *labels*, *help=''*, *values=None*)
Get the XML content for a drop down menu when making a ParaView plugin.

getFileReaderXml

PVGeo._helpers.xml.**getFileReaderXml**(*extensions*, *readerDescription=''*, *command='AddFileName'*)
Get the XML for a selectectable file for a reader when building a ParaView plugin

---

Note:

- Thanks: Daan van Vugt and for his work here
- Modified by Bane Sullivan

---

getInputArrayXml

PVGeo._helpers.xml.**getInputArrayXml**(*labels=None*, *nInputPorts=1*, *numArrays=1*, *inputNames='Input'*)
Get the XML content for an array selection drop down menu when making a ParaView plugin.

getPropertyXml

PVGeo._helpers.xml.**getPropertyXml**(*name*, *command*, *default_values*, *visibility='default'*, *help=''*)
Get the XML content for a property of a parameter for a python data object when making a ParaView plugin.

getPythonPathProperty

PVGeo._helpers.xml.**getPythonPathProperty**()
Get the XML content for setting the Python path when making a ParaView plugin.

getReaderTimeStepValues

PVGeo._helpers.xml.**getReaderTimeStepValues**(*extensions*, *readerDescription*)
Get the XML content for reader time step values the Python path when making a ParaView plugin.

getVTKTypeMap

PVGeo._helpers.xml.**getVTKTypeMap**()
Get the the VTK Type Map as specified in `vtkType.h`

4.5 General Filters

4.5.1 poly

Add Cell Connectivity to Points

class PVGeo.filters.poly.**AddCellConnToPoints**(**kwargs*)
Bases: PVGeo.base.FilterBase

This filter will add linear cell connectivity between scattered points. You have the option to add `VTK_Line` or `VTK_PolyLine` connectivity. `VTK_Line` connectivity makes a straight line between the points in order (either in the order by index or using a nearest neighbor calculation). The `VTK_PolyLine` adds a poly line connectivity between all points as one spline (either in the order by index or using a nearest neighbor calculation). Type map is specified in *vtkCellType.h*.

Cell Connectivity Types:

- 4: Poly Line
- 3: Line

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output data object

SetCellType(*cellType*)
Set the cell typ by the integer id as specified in *vtkCellType.h*

SetUseNearestNbr(*flag*)
Set a flag on whether to use SciPy's `cKDTree` nearest neighbor algorithms to sort the points to before adding linear connectivity.

_ConnectCells(*pdi*, *pdo*, *logTime=False*)
Internal helper to perfrom the connection

Array Math

class PVGeo.filters.poly.**ArrayMath**(**kwargs*)
Bases: *PVGeo.base.FilterPreserveTypeBase*

This filter allows the user to select two input data arrays on which to perfrom math operations. The input arrays are used in their order of selection for the operations.

Available Math Operations:

- *add*: This adds the two data arrays together
- *subtract*: This subtracts input array 2 from input array 1
- *multiply*: Multiplies the two data arrays together
- *divide*: Divide input array 1 by input array 2 (arr1/arr2)
- *correlate*: Use *np.correlate(arr1, arr2, mode='same')*

GetMultiplier()
Return the set multiplier/scalar

GetNewArrayName()

static GetOperation(*idx*)
Gets a math operation based on an index in the keys

Returns the math operation method

Return type callable

static GetOperationNames()
Gets a list of the math operation keys

Returns the keys for getting the math operations

Return type list(str)

static GetOperations()
Returns the math operation methods as callable objects in a dictionary

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to perfrom operation and generate output

SetInputArrayToProcess(*idx*, *port*, *connection*, *field*, *name*)
Used by pipeline/paraview GUI wrappings to set the input arrays

SetMultiplier(*val*)
This is a static shifter/scale factor across the array after normalization.

SetNewArrayName(*name*)
Give the new array a meaningful name.

SetOperation(*op*)
Set the math operation to perform

Parameters **op** (*str, int, or callable*) – The operation as a string key, int index, or callable method

---

Note: This can accept a callable method to set a custom operation as long as its signature is: `<callable>(arr1, arr2)`

_MathUp(*pdi*, *pdo*)
Make sure to pass array names and integer associated fields. Use helpers to get these properties.

_SetInputArray1(*field*, *name*)

_SetInputArray2(*field*, *name*)

static _add(*arr1*, *arr2*)

static _correlate(*arr1*, *arr2*)
Use `np.correlate()` on `mode='same'` on two selected arrays from one input.

static _divide(*arr1*, *arr2*)

static _multiply(*arr1*, *arr2*)

static _subtract(*arr1*, *arr2*)

Normalize Array

class PVGeo.filters.poly.**NormalizeArray**(***kwargs*)
Bases: *PVGeo.base.FilterPreserveTypeBase*

This filter allows the user to select an array from the input data set to be normalized. The filter will append another array to that data set for the output. The user can specify how they want to rename the array, can choose a multiplier, and can choose from several types of common normalizations (more functionality added as requested).

Normalization Types:

- *feature_scale*: Feature Scale
- *standard_score*: tandard Score
- *log10*: Natural Log
- *natural_log*: Log Base 10
- *just_multiply*: Only Multiply by Multiplier

static GetArrayRange(*pdi*, *field*, *name*)
Returns a tuple of the range for a `vtkDataArray` on a `vtkDataObject`

GetMultiplier()
Return the set multiplier/scalar

GetNewArrayName()

static GetNormalization(*idx*)
Gets a normalization based on an index in the keys

Returns the normalization method

Return type callable

static GetNormalizationNames()
Gets a list of the normalization keys

Returns the keys for getting the normalizations

Return type list(str)

static GetNormalizations()
All Available normalizations

Returns dictionary of callable methods for normalizing an array

Return type dict

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output

SetInputArrayToProcess(*idx*, *port*, *connection*, *field*, *name*)
Used by pipeline/paraview GUI wrappings to set the input arrays

SetMultiplier(*val*)
This is a static shifter/scale factor across the array after normalization.

SetNewArrayName(*name*)
Give the new array a meaningful name.

SetNormalization(*norm*)
Set the normalization operation to perform

Parameters **norm** (*str, int, or callable*) – The operation as a string key, int index, or callable method

---

Note: This can accept a callable method to set a custom operation as long as its signature is: `<callable>(arr)`

SetTakeAbsoluteValue(*flag*)
This will take the absolute value of the array before normalization.

_Normalize(*pdi*, *pdo*)
Perform normalize on a data array for any given VTK data object.

static _featureScale(*arr*)

static _log10(*arr*)

static _logNat(*arr*)

static _passArray(*arr*)

static _standardScore(*arr*)

Percent Threshold

class PVGeo.filters.poly.**PercentThreshold**(***kwargs*)
Bases: PVGeo.base.FilterBase

Allows user to select a percent of the data range to threshold. This will find the data range of the selected input array and remove the bottom percent. This can be reversed using the invert property.

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline for execution

SetInputArrayToProcess(*idx*, *port*, *connection*, *field*, *name*)
Used by pipeline/paraview GUI wrappings to set the input array to threshold

SetInvert(*flag*)
Use to invert the threshold filter

SetPercent(*percent*)
Set the percent for the threshold in range (0, 100). Any values falling beneath the set percent of the total data range will be removed.

SetUseContinuousCellRange(*flag*)
If this is on (default is off), we will use the continuous interval [minimum cell scalar, maxmimum cell scalar] to intersect the threshold bound , rather than the set of discrete scalar values from the vertices

Points to Tube

class PVGeo.filters.poly.**PointsToTube**(***kwargs*)
Bases: *PVGeo.filters.poly.AddCellConnToPoints*

Takes points from a vtkPolyData object and constructs a line of those points then builds a polygonal tube around that line with some specified radius and number of sides.

SetNumberOfSides(*num*)
Set the number of sides (resolution) for the tube

SetRadius(*radius*)
Set the radius of the tube

_ConnectCells(*pdi*, *pdo*, *logTime=False*)
This uses the parent's `_ConnectCells()` to build a tub around

4.5.2 slicing

Many Slices Along Axis

class PVGeo.filters.slicing.**ManySlicesAlongAxis**(*numSlices=5*, *axis=0*, *rng=None*, *outputType='vtkUnstructuredGrid'*)
Bases: `PVGeo.filters.slicing._SliceBase`

Slices a `vtkDataSet` along a given axis many times. This produces a specified number of slices at once each with a normal vector oriented along the axis of choice and spaced uniformly through the range of the dataset on the chosen axis.

GetAxis()
Get the set axis to slice upon as int index (0,1,2)

GetInputBounds(*pdi*)
Gets the bounds of the input data set on the set slicing axis.

GetInputCenter(*pdi*)
Gets the center of the input data set

Returns the XYZ coordinates of the center of the data set.

Return type tuple

GetNormal()
Get the normal of the slicing plane

GetRange()
Get the slicing range for the set axis

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output

SetAxis(*axis*)
Set the axis on which to slice

Parameters **axis** (*int*) – the axial index (0, 1, 2) = (x, y, z)

_GetOrigin(*pdi*, *idx*)
Internal helper to get plane origin

_SetAxialRange(*pdi*)
Internal helper to set the slicing range along the set axis

_UpdateNumOutputs(*num*)
for internal use only

Many Slices Along Points

class PVGeo.filters.slicing.**ManySlicesAlongPoints**(*numSlices=5*, *nearestNbr=True*)
Bases: PVGeo.filters.slicing._SliceBase

Takes a series of points and a data source to be sliced. The points are used to construct a path through the data source and a slice is added at intervals of that path along the vector of that path at that point. This constructs many slices through the input dataset as an appended output vtkUnstructuredGrid.

Note:

- Make sure the input data source is slice-able.
- The SciPy module is required for this filter.

Apply(*points*, *data*)
Update the algorithm and get the output data object

FillInputPortInformation(*port*, *info*)
This simply makes sure the user selects the correct inputs

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output

SetUseNearestNbr(*flag*)
Set a flag on whether to use SciPy's nearest neighbor approximation when generating the slicing path

_ManySlicesAlongPoints(*pdipts*, *pdidata*, *pdo*)
Internal helper to perfrom the filter

Slice Through Time

class PVGeo.filters.slicing.**SliceThroughTime**(*numSlices=5*, *dt=1.0*, *axis=0*, *rng=None*)
Bases: *PVGeo.filters.slicing.ManySlicesAlongAxis*

Takes a sliceable vtkDataSet and progresses a slice of it along a given axis. The macro requires that the clip already exist in the pipeline. This is especially useful if you have many clips linked together as all will move through the seen as a result of this macro.

GetTimestepValues()
Use this in ParaView decorator to register timesteps

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output

RequestInformation(*request*, *inInfoVec*, *outInfoVec*)
Used by pipeline to set the time information

SetNumberOfSlices(*num*)
Set the number of slices/timesteps to generate

SetTimeDelta(*dt*)
Set the time step interval in seconds

_UpdateTimeSteps()
For internal use only

4.5.3 tables

Combine Tables

class PVGeo.filters.tables.**CombineTables**
Bases: PVGeo.base.FilterBase

Takes two tables and combines them if they have the same number of rows.

Apply(*table0*, *table1*)
Update the algorithm and get the output data object

FillInputPortInformation(*port*, *info*)
Used by pipeline. Necessary when dealing with multiple input ports

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output

Extract Array

class PVGeo.filters.tables.**ExtractArray**
Bases: PVGeo.base.FilterBase

Extract an array from a vtkDataSet and make a vtkTable of it.

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output

SetInputArrayToProcess(*idx*, *port*, *connection*, *field*, *name*)
Used by pipeline/paraview GUI wrappings to set the input arrays

Reshape Table

class PVGeo.filters.tables.**ReshapeTable**(***kwargs*)
Bases: PVGeo.base.FilterBase

This filter will take a vtkTable object and reshape it. This filter essentially treats vtkTable``s as 2D matrices and reshapes them using ``numpy.reshape in a C contiguous manner. Unfortunately, data fields will be renamed arbitrarily because VTK data arrays require a name.

AddName(*name*)
Use to append a name to the list of data array names for the output table

GetNames()

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline

SetNames(*names*)
Set names using a semicolon (;) seperated string or a list of strings

Parameters **names** (*string*) – a string of data array names for the reshaped table using a semicolon (;) to spearate

SetNumberOfColumns(*ncols*)
Set the number of columns for the output vtkTable

SetNumberOfRows(*nrows*)
Set the number of rows for the output vtkTable

SetOrder(*order*)
Set the reshape order (`'C'` of `'F'`)

_Reshape(*pdi*, *pdo*)
Internal helper to perfrom the reshape

4.5.4 voxelize

Voxelize Points

class PVGeo.filters.voxelize.**VoxelizePoints**(***kwargs*)
Bases: PVGeo.base.FilterBase

This makes a `vtkUnstructuredGrid` of scattered points given voxel sizes as input arrays. This assumes that the data is at least 2-Dimensional on the XY Plane.

static AddCellData(*grid*, *arr*, *name*)
Add a NumPy array as cell data to the given grid input

AddFieldData(*grid*)
An internal helper to add the recovered information as field data

EstimateUniformSpacing(*x*, *y*, *z*)
This assumes that the input points make up some sort of uniformly spaced grid on at least an XY Plane

PointsToGrid(*xo*, *yo*, *zo*, *dx*, *dy*, *dz*, *grid=None*)
Convert XYZ points to a `vtkUnstructuredGrid`.

RequestData(*request*, *inInfoVec*, *outInfoVec*)
Used by pipeline to generate output

SetDeltaX(*dx*)
Set the X cells spacing

Parameters **dx** (*float or np.array(floats)*) – the spacing(s) for the cells in the X-direction

SetDeltaY(*dy*)
Set the Y cells spacing

Parameters **dy** (*float or np.array(floats)*) – the spacing(s) for the cells in the Y-direction

SetDeltaZ(*dz*)
Set the Z cells spacing

Parameters **dz** (*float or np.array(floats)*) – the spacing(s) for the cells in the Z-direction

SetDeltas(*dx*, *dy*, *dz*)
Set the cell spacings for each axial direction

Parameters

- **dx** (*float or np.array(floats)*) – the spacing(s) for the cells in the X-direction
- **dy** (*float or np.array(floats)*) – the spacing(s) for the cells in the Y-direction
- **dz** (*float or np.array(floats)*) – the spacing(s) for the cells in the Z-direction

SetEstimateGrid(*flag*)
Set a flag on whether or not to estimate the grid spacing/rotation

SetSafeSize(*safe*)
A voxel size to use if a spacing cannot be determined for an axis

_CopyArrays(*pdi*, *pdo*)
internal helper to copy arrays from point data to cell data in the voxels.

4.5.5 xyz

Extract Points

class PVGeo.filters.xyz.**ExtractPoints**
Bases: PVGeo.base.FilterBase

Extracts XYZ coordinates and point/cell data from an input vtkDataSet

RequestData(*request*, *inInfo*, *outInfo*)
Overwritten by subclass to execute the algorithm.

PointsToPolyData

PVGeo.filters.xyz.**PointsToPolyData**(*points*)
Create vtkPolyData from a numpy array of XYZ points

Returns points with point-vertex cells

Return type vtkPolyData

Rotate Points

class PVGeo.filters.xyz.**RotatePoints**(*angle=45.0, origin=[0.0, 0.0], useCorner=True*)
Bases: PVGeo.base.FilterBase

Rotates XYZ coordinates in *vtkPolyData* around an origin at a given angle on the XY plane.

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output.

SetOrigin(*xo*, *yo*)
Sets the origin to perform the rotate around.

SetRotationDegrees(*theta*)
Sets the rotational angle in degrees.

SetUseCorner(*flag*)
A flag to use a corner of the input data set as the rotational origin.

Rotation Tool

class PVGeo.filters.xyz.**RotationTool**(*decimals=6*)
Bases: object

A class that holds a set of methods/tools for performing and estimating coordinate rotations.

static CosBetween(*pts*)

static DistanceBetween(*pts*)
Gets the distance between two points

EstimateAndRotate(*x*, *y*, *z*)
A method to estimate the rotation of a set of points and correct that rotation on the XY plane

static Rotate(*pts*, *theta*)
Rotate points around (0,0,0) given an anlge on the XY plane

static RotateAround(*pts*, *theta*, *origin*)
Rotate points around an origins given an anlge on the XY plane

static RotationMatrix(*vector_orig*, *vector_fin*)
Calculate the rotation matrix required to rotate from one vector to another. For the rotation of one vector to another, there are an infinit series of rotation matrices possible. Due to axially symmetry, the rotation axis can be any vector lying in the symmetry plane between the two vectors. Hence the axis-angle convention will be used to construct the matrix with the rotation axis defined as the cross product of the two vectors. The rotation angle is the arccosine of the dot product of the two unit vectors. Given a unit vector parallel to the rotation axis, w = [x, y, z] and the rotation angle a, the rotation matrix R is:

```
      |  1 + (1-cos(a))*(x*x-1)   -z*sin(a)+(1-cos(a))*x*y   y*sin(a)+(1-
↪cos(a))*x*z |
R  =  |  z*sin(a)+(1-cos(a))*x*y   1 + (1-cos(a))*(y*y-1)   -x*sin(a)+(1-
↪cos(a))*y*z |
      | -y*sin(a)+(1-cos(a))*x*z   x*sin(a)+(1-cos(a))*y*z   1 + (1-
↪cos(a))*(z*z-1)  |
```

Parameters

- **vector_orig** (*umpy array, len 3*) – The unrotated vector defined in the reference frame.
- **vector_fin** (*numpy array, len 3*) – The rotated vector defined in the reference frame.

Note: This code was adopted from printipi under the MIT license.

static SinBetween(*pts*)

_ConvergeAngle(*pt1*, *pt2*)
Internal use only: pts should only be a two neighboring points.

_EstimateAngleAndSpacing(*pts*, *sample=0.1*)
internal use only

static _GetRotationMatrix(*theta*)

latLonTableToCartesian

PVGeo.filters.xyz.**latLonTableToCartesian**(*pdi*, *arrlat*, *arrlon*, *arralt*, *radius=6371.0*, *pdo=None*)

WORK IN PROGRESS

4.6 Grid Tools

4.6.1 extract_topo

Extract Topography

class PVGeo.grids.extract_topo.**ExtractTopography**
Bases: PVGeo.base.FilterBase

This filter takes two inputs: a gridded data set and a set of points for a Topography source. This will add a boolean data array to the cell data of the input grid on whether that cell should be active (under topographic layer).

Apply(*data*, *points*)
Update the algorithm and get the output data object

FillInputPortInformation(*port*, *info*)
This simply makes sure the user selects the correct inputs

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output

RequestDataObject(*request*, *inInfo*, *outInfo*)
Constructs the output data object based on the input data object

static _ExtractTopography__GetVoxelCenter(*voxel*)
Returns tuple for center of Voxel

Note: The Z-coordinate is at the top of the cell

4.6.2 reverse_axii

Reverse Image Data Axii

class PVGeo.grids.reverse_axii.**ReverseImageDataAxii**(*axes=[True, True, True]*)
Bases: PVGeo.base.FilterBase

This filter will flip vtkImageData on any of the three cartesian axii. A checkbox is provided for each axis on which you may desire to flip the data.

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output.

SetFlipX(*flag*)
Set the filter to flip th input data along the X-axis

SetFlipY(*flag*)
Set the filter to flip th input data along the Y-axis

SetFlipZ(*flag*)
Set the filter to flip th input data along the Z-axis

_ReverseGridAxii(*idi*, *ido*)

4.6.3 surfer

This module contains general grid readers and writers for programs like Surfer.

Surfer Grid Reader

class PVGeo.grids.surfer.**SurferGridReader**(*outputType='vtkImageData'*, ***kwargs*)
Bases: *PVGeo.readers.delimited.DelimitedTextReader*

Read 2D ASCII Surfer grid files

GetDataName()

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to get data for current timestep and populate the output data object.

RequestInformation(*request*, *inInfo*, *outInfo*)
Used by pipeline to set grid extents.

SetDataName(*dataName*)

_ExtractHeader(*content*)
Override this. Removes header from single file's content.

_FileContentsToDataArray(*contents*)
Puts Surfer file contents (Z-values) into a 1D array

_GetRawData(*idx=0*)
This will return the proper data for the given timestep. This method handles Surfer's NaN data values and checkes the value range

Write `vtkImageData` to Surfer Format

class PVGeo.grids.surfer.**WriteImageDataToSurfer**
Bases: *PVGeo.base.WriterBase*

Write a 2D `vtkImageData` object to the Surfer grid format

RequestData(*request*, *inInfoVec*, *outInfoVec*)
OVERWRITE: This is executed by the pipeline and handles the write out

SetInputArrayToProcess(*idx*, *port*, *connection*, *field*, *name*)
Used by pipeline/paraview GUI wrappings to set the input arrays. The inpput array is the data value (z-value) to write for the Surfer format

4.6.4 table2grid

Table To Grid

class PVGeo.grids.table2grid.**TableToGrid**(*extent=[10, 10, 10]*, *order='C'*, *spacing=[1.0, 1.0, 1.0]*, *origin=[0.0, 0.0, 0.0]*, *seplib=False*, *swapXY=False*)
Bases: `PVGeo.base.FilterBase`

This filter takes a `vtkTable` object with columns that represent data to be translated (reshaped) into a 3D grid (2D also works, just set the third dimensions extent to 1). The grid will be a `n1` by `n2` by `n3` `vtkImageData` structure and an origin (south-west bottom corner) can be set at any xyz point. Each column of the `vtkTable` will represent a data attribute of the `vtkImageData` formed (essentially a uniform mesh). The SEPlib option allows you to unfold data that was packed in the SEPlib format where the most important dimension is z and thus the z data is d1 (`d1=z`, `d2=y`, `d3=x`). When using SEPlib, specify `n1` as the number of elements in the Z-direction, `n2` as the number of elements in the X-direction, and `n3` as the number of elements in the Y-direction (and so on for other parameters).

Warning: Work in progress

static RefoldIdx(*SEPlib=True*, *swapXY=False*)
Theses are indexing corrections to set the spacings and origin witht the correct axes after refolding.

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output

RequestInformation(*request*, *inInfo*, *outInfo*)
Used by pipeline to set whole output extent.

SetExtent(*nx*, *ny*, *nz*)
Set the extent of the output grid

SetOrder(*order*)
Set the reshape order (*'C'* or *'F'*)

SetOrigin(*x0*, *y0*, *z0*)
Set the origin of the output *vtkImageData*

SetSEPlib(*flag*)
Set a flag to swap the axial order for the refold of the table

SetSpacing(*dx*, *dy*, *dz*)
Set the spacing for the points along each axial direction

SetSwapXY(*flag*)
Set a flag to swap the X and Y axii

_TableToGrid(*pdi*, *ido*)
Converts a table of data arrays to vtkImageData given an extent to reshape that table. Each column in the table will be treated as seperate data arrays for the described data space.

static _rearangeSEPlib(*arr*, *extent*)
This is a helper method to swap axes when using SEPlib axial conventions.

static _refold(*arr*, *extent*, *SEPlib=True*, *order='F'*, *swapXY=False*)
This is a helper method to handle grabbing a data array and make sure it is ready for VTK/Fortran ordering in `vtkImageData`.

static _transposeXY(*arr*, *extent*, *SEPlib=False*)
Transposes X and Y axes. Needed for PoroTomo project.

static _unpack(*arr*, *extent*, *order='C'*)
This is a helper method that handles the initial unpacking of a data array. ParaView and VTK use Fortran packing so this is convert data saved in C packing to Fortran packing.

4.6.5 trans_origin

Translate Grid Origin

class PVGeo.grids.trans_origin.**TranslateGridOrigin**(*corner=1*)
Bases: `PVGeo.base.FilterBase`

This filter will translate the origin of *vtkImageData* to any specified Corner of the data set assuming it is currently in the South West Bottom Corner (will not work if Corner was moved prior).

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output.

SetCorner(*corner*)
Set the corner to use

Parameters **corner** (*int*) – corner location; see note.

Note:

- 1: South East Bottom
- 2: North West Bottom
- 3: North East Bottom
- 4: South West Top
- 5: South East Top
- 6: North West Top
- 7: North East Top

_Translate(*pdi*, *pdo*)

4.6.6 write

4.7 GSLib & SGeMS

4.7.1 gslib

GSLib Table Reader

class PVGeo.gslib.gslib.**GSLibReader**(*outputType='vtkTable'*, ***kwargs*)
Bases: *PVGeo.readers.delimited.DelimitedTextReader*

Reads a GSLIB file format to a `vtkTable`. The GSLIB file format has headers lines followed by the data as a space delimited ASCI file (this filter is set up to allow you to choose any single character delimiter). The first header line is the title and will be printed to the console. This line may have the dimensions for a grid to be made of the data. The second line is the number (n) of columns of data. The next n lines are the variable names for the data in each column. You are allowed up to ten characters for the variable name. The data follow with a space between each field (column).

GetFileHeader()
Returns the file header. If file hasn't been read, returns `None`

_ExtractHeader(*content*)
Override this. Removes header from single file's content.

4.7.2 sgems

SGeMS Grid Reader

class PVGeo.gslib.sgems.**SGeMSGridReader**(*origin=(0.0, 0.0, 0.0)*, *spacing=(1.0, 1.0, 1.0)*, ***kwargs*)
Bases: *PVGeo.gslib.gslib.GSLibReader*

Generates `vtkImageData` from the uniform grid defined in the inout file in the SGeMS grid format. This format is simply the GSLIB format where the header line defines the dimensions of the uniform grid.

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to get output data object for given time step. Constructs the `vtkImageData`

RequestInformation(*request*, *inInfo*, *outInfo*)
Used by pipeline to set grid extents.

SetOrigin(*ox*, *oy*, *oz*)
Set the origin corner of the grid

SetSpacing(*dx*, *dy*, *dz*)
Set the spacing for each axial direction

_ExtractHeader(*content*)
Override this. Removes header from single file's content.

_ReadExtent()
Reads the input file for the SGeMS format to get output extents. Computationally inexpensive method to discover whole output extent.

Returns This returns a tuple of the whole extent for the uniform grid to be made of the input file (0,n1-1, 0,n2-1, 0,n3-1). This output should be directly passed to set the whole output extent.

Return type tuple

4.7.3 write

Write `vtkImageData` To SGeMS Grid Format

class PVGeo.gslib.write.**WriteImageDataToSGeMS**(*inputType='vtkImageData'*)
Bases: *PVGeo.base.WriterBase*

Writes a `vtkImageData` object to the SGeMS uniform grid format. This writer can only handle point data.

RequestData(*request*, *inInfoVec*, *outInfoVec*)
OVERWRITE: This is executed by the pipeline and handles the write out

Write `vtkTable` To GSLib Format

class PVGeo.gslib.write.**WriteTableToGSLib**(*inputType='vtkTable'*)
Bases: *PVGeo.base.WriterBase*

Write the row data in a `vtkTable` to the GSLib Format

RequestData(*request*, *inInfoVec*, *outInfoVec*)
OVERWRITE: This is executed by the pipeline and handles the write out

SetHeader(*header*)
Set the file header string

4.8 Model Builder

4.8.1 earth

GSLib Table Reader

class PVGeo.model_build.earth.**EarthSource**(*radius=6371.0*)

Bases: *PVGeo.base.AlgorithmBase*

A simple data source to produce a vtkEarthSource

RequestData(*request*, *inInfo*, *outInfo*)

Overwritten by subclass to execute the algorithm.

SetRadius(*radius*)

4.8.2 evenModel

Create Even Rectilinear Grid

class PVGeo.model_build.evenModel.**CreateEvenRectilinearGrid**(*extent=[10, 10, 10], xrng=[-1.0, 1.0], yrng=[-1.0, 1.0], zrng=[-1.0, 1.0]*)

Bases: *PVGeo.base.AlgorithmBase*

This creates a vtkRectilinearGrid where the discretization along a given axis is uniformly distributed.

RequestData(*request*, *inInfo*, *outInfo*)

Overwritten by subclass to execute the algorithm.

RequestInformation(*request*, *inInfo*, *outInfo*)

Overwritten by subclass to provide meta-data to downstream pipeline.

SetExtent(*nx*, *ny*, *nz*)

Set the extent of the output grid.

SetXRange(*start*, *stop*)

Set range (min, max) for the grid in the X-direction.

SetYRange(*start*, *stop*)

Set range (min, max) for the grid in the Y-direction

SetZRange(*start*, *stop*)

Set range (min, max) for the grid in the Z-direction

Create Uniform Grid

class PVGeo.model_build.evenModel.**CreateUniformGrid**(*extent=[10, 10, 10], spacing=[1.0, 1.0, 1.0], origin=[0.0, 0.0, 0.0]*)

Bases: *PVGeo.base.AlgorithmBase*

Create uniform grid (vtkImageData)

RequestData(*request*, *inInfo*, *outInfo*)

Overwritten by subclass to execute the algorithm.

RequestInformation(*request*, *inInfo*, *outInfo*)
Overwritten by subclass to provide meta-data to downstream pipeline.

SetExtent(*nx*, *ny*, *nz*)
Set the extent of the output grid.

SetOrigin(*x0*, *y0*, *z0*)
Set the origin of the output grid.

SetSpacing(*dx*, *dy*, *dz*)
Set the spacing for the points along each axial direction.

4.8.3 oddModel

Create Tensor Mesh

class PVGeo.model_build.oddModel.**CreateTensorMesh**(*origin=[-350.0, -400.0, 0.0], dataname='Data'*)

Bases: *PVGeo.base.AlgorithmBase*

This creates a vtkRectilinearGrid where the discretization along a given axis is uniformly distributed.

GetExtent()

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to generate output data object

RequestInformation(*request*, *inInfo*, *outInfo*)
Used by pipeline to set output whole extent

SetOrigin(*x0*, *y0*, *z0*)
Set the origin of the output

SetXCells(*xcells*)
Set the spacings for the cells in the X direction

Parameters **xcells** (*list or np.array(floats)*) – the spacings along the X-axis

SetXCellsStr(*xcellstr*)
Set the spacings for the cells in the X direction

Parameters **xcellstr** (*str*) – the spacings along the X-axis in the UBC style

SetYCells(*ycells*)
Set the spacings for the cells in the Y direction

Parameters **ycells** (*list or np.array(floats)*) – the spacings along the Y-axis

SetYCellsStr(*ycellstr*)
Set the spacings for the cells in the Y direction

Parameters **ycellstr** (*str*) – the spacings along the Y-axis in the UBC style

SetZCells(*zcells*)
Set the spacings for the cells in the Z direction

Parameters **zcells** (*list or np.array(floats)*) – the spacings along the Z-axis

SetZCellsStr(*zcellstr*)
Set the spacings for the cells in the Z direction

Parameters **zcellstr** (*str*) – the spacings along the Z-axis in the UBC style

_AddModelData(*pdo*, *data*)

_MakeModel(*pdo*)

static _ReadCellLine(*line*)
Read cell sizes for each line in the UBC mesh line strings

4.9 General Readers

4.9.1 binaries

Madagascar SSRSF Reader

class PVGeo.readers.binaries.**MadagascarReader**(***kwargs*)
Bases: *PVGeo.readers.binaries.PackedBinariesReader*

This reads in float or double data that is packed into a Madagascar binary file format with a leader header. The reader ignores all of the ascii header details by searching for the sequence of three special characters: EOL EOL EOT and it will treat the followng binary packed data as one long array and make a `vtkTable` with one column of that data. The reader uses defaults to import as floats with native endianness. Use the Table to Uniform Grid or the Reshape Table filters to give more meaning to the data. We will later implement the ability to create a gridded volume from the header info. This reader is a quick fix for Samir. We chose to use a `vtkTable` object as the output of this reader because it gives us more flexibility in the filters we can apply to this data down the pipeline and keeps thing simple when using filters in this repository. Details Here.

_ReadRawFile(*fileName*)

Packed Binaries Reader

class PVGeo.readers.binaries.**PackedBinariesReader**(***kwargs*)
Bases: *PVGeo.base.ReaderBase*

This reads in float or double data that is packed into a binary file format. It will treat the data as one long array and make a `vtkTable` with one column of that data. The reader uses defaults to import as floats with native endianness. Use the Table to Uniform Grid or the Reshape Table filters to give more meaning to the data. We chose to use a `vtkTable` object as the output of this reader because it gives us more flexibility in the filters we can apply to this data down the pipeline and keeps thing simple when using filters in this repository.

ConvertArray(*arr*)
Converts the numpy array to a vtkDataArray

GetDataName()

GetDataTypes()

GetEndian()

RequestData(*request*, *inInfo*, *outInfo*)
Used by pipeline to request data for current timestep

SetDataName(*dataName*)
The string name of the data array generated from the inut file.

SetDataType(*dtype*)
The data type of the binary file: *double='d'*, *float='f'*, *int='i'*

SetEndian(*endian*)
The endianness of the data file.

Parameters **endian** (*int or char*) – no preference = '' or 0, Little = 1 or < or Big = 2 >.

_GetFileContents(*idx=None*)

_GetRawData(*idx=0*)
This will return the proper data for the given timestep

_ReadRawFile(*fileName*)

_ReadUpFront()
Should not need to be overridden

4.9.2 delimited

Delimited Text Reader

class PVGeo.readers.delimited.**DelimitedTextReader**(*nOutputPorts=1*, *outputType='vtkTable'*, ***kwargs*)

Bases: *PVGeo.base.ReaderBase*

This reader will take in any delimited text file and make a `vtkTable` from it. This is not much different than the default .txt or .csv reader in ParaView, however it gives us room to use our own extensions and a little more flexibility in the structure of the files we import.

GetSkipRows()

GetTitles()

HasTitles()

RequestData(*request*, *inInfo*, *outInfo*)
Used b pipeline to get data for current timestep and populate the output data object.

SetComments(*identifier*)
The character identifier for comments within the file.

SetDelimiter(*deli*)
The input file's delimiter. To use a tab delimiter please use `SetUseTab()`

Parameters **deli** (*str*) – a string delimiter/seperator

SetHasTitles(*flag*)
A boolean for if the delimited file has header titles for the data arrays.

SetSkipRows(*skip*)
The integer number of rows to skip at the top of the file.

SetSplitOnWhiteSpace(*flag*)
Set a boolean flag to override the `SetDelimiter()` and use any white space as a delimiter.

SetUseTab(*flag*)
Deprecated

_ExtractHeader(*content*)
Override this. Removes header from single file's content.

_ExtractHeaders(*contents*)
Should NOT be overriden.

_FileContentsToDataArray(*contents*)
Should NOT need to be overriden

_GetDeli()
For itenral use

_GetFileContents(*idx=None*)

_GetRawData(*idx=0*)
This will return the proper data for the given timestep.

_ReadUpFront()
Should not need to be overridden.

XYZ Text Reader

class PVGeo.readers.delimited.**XYZTextReader**(***kwargs*)
Bases: *PVGeo.readers.delimited.DelimitedTextReader*

A makeshift reader for XYZ files where titles have comma delimiter and data has space delimiter.

_ExtractHeader(*content*)
Override this. Removes header from single file's content.

4.10 UBC Mesh Tools

4.10.1 octree

UBC OcTree Mesh Appender

class PVGeo.ubc.octree.**OcTreeAppender**(***kwargs*)
Bases: *PVGeo.ubc.two_file_base.ModelAppenderBase*

This filter reads a timeseries of models and appends it to an input `vtkUnstructuredGrid`

_PlaceOnMesh(*output*, *idx=0*)

_ReadUpFront()

UBC OcTree Mesh Reader

class PVGeo.ubc.octree.**OcTreeReader**(*nOutputPorts=1*, *outputType='vtkUnstructuredGrid'*, ***kwargs*)
Bases: *PVGeo.ubc.two_file_base.ubcMeshReaderBase*

This class reads a UBC OcTree Mesh file and builds a `vtkUnstructuredGrid` of the data in the file. Model File is optional. Reader will still construct `vtkUnstructuredGrid` safely.

ClearMesh()
Use to clean/rebuild the mesh.

ClearModels()
Use to clean the models and reread the data

static PlaceModelOnOcTreeMesh(*mesh*, *model*, *dataNm='Data'*)
Places model data onto a mesh. This is for the UBC Grid data reaers to associate model data with the mesh grid.

Parameters

- **mesh** (*vtkUnstructuredGrid*) – The `vtkUnstructuredGrid` that is the mesh to place the model data upon. Needs to have been read in by ubcOcTree

- **model** (*np.ndarray*) – A NumPy float array that holds all of the data to place inside of the mesh's cells.
- **dataNm** (*str*) – The name of the model data array once placed on the vtkUnstructuredGrid.

Returns Returns the input vtkUnstructuredGrid with model data appended.

Return type vtkUnstructuredGrid

RequestData(*request*, *inInfo*, *outInfo*)
Overwritten by subclass to execute the algorithm.

RequestInformation(*request*, *inInfo*, *outInfo*)
Pipeline method for handling requests about the grid extents and time step values

_OcTreeReader__ubcOcTree(*FileName_Mesh*, *FileName_Models*, *output*)
Wrapper to Read UBC GIF OcTree mesh and model file pairs. UBC OcTree models are defined using a 2-file format. The "mesh" file describes how the data is descritized. The "model" file lists the physical property values for all cells in a mesh. A model file is meaningless without an associated mesh file. This only handles OcTree formats

Parameters **FileName_Mesh** (*str*) – The OcTree Mesh filename as an absolute path for the input mesh file in UBC OcTree Mesh Format

FileName_Models (list(str)): The model filenames as absolute paths for the input model timesteps in UBC OcTree M
output (vtkUnstructuredGrid): The output data object

Returns Returns a vtkUnstructuredGrid generated from the UBC 2D/3D Mesh grid. Mesh is defined by the input mesh file. Cell data is defined by the input model file.

Return type vtkUnstructuredGrid

static ubcOcTreeMesh(*FileName*, *pdo=None*)
This method reads a UBC OcTree Mesh file and builds a vtkUnstructuredGrid of the data in the file. This method generates the vtkUnstructuredGrid without any data attributes.

Parameters

- **FileName** (*str*) – The mesh filename as an absolute path for the input mesh file in UBC OcTree format.
- **pdo** (*vtkUnstructuredGrid*) – A pointer to the output data object.

Returns Returns a vtkUnstructuredGrid generated from the UBCMesh grid. Mesh is defined by the input mesh file. No data attributes here, simply an empty mesh. Use the PlaceModelOnOcTreeMesh() method to associate with model data.

Return type vtkUnstructuredGrid

4.10.2 tensor_mesh

UBC Tensor Mesh Appender

class PVGeo.ubc.tensor_mesh.**TensorMeshAppender**(***kwargs*)
Bases: *PVGeo.ubc.two_file_base.ModelAppenderBase*

This filter reads a timeseries of models and appends it to an input vtkRectilinearGrid

_PlaceOnMesh(*output*, *idx=0*)

_ReadUpFront()

UBC Tensor Mesh Reader

class PVGeo.ubc.tensor_mesh.**TensorMeshReader**(*nOutputPorts=1*, *outputType='vtkRectilinearGrid'*, ***kwargs*)

Bases: *PVGeo.ubc.two_file_base.ubcMeshReaderBase*

UBC Mesh 2D/3D models are defined using a 2-file format. The "mesh" file describes how the data is discretized. The "model" file lists the physical property values for all cells in a mesh. A model file is meaningless without an associated mesh file. The reader will automatically detect if the mesh is 2D or 3D and read the remainder of the data with that dimensionality assumption. If the mesh file is 2D, then then model file must also be in the 2D format (same for 3D).

Note: Model File is optional. Reader will still construct `vtkRectilinearGrid` safely.

ClearMesh()
Use to clean/rebuild the mesh

ClearModels()
Use to clean the models and reread

static PlaceModelOnMesh(*mesh*, *model*, *dataNm='Data'*)
Places model data onto a mesh. This is for the UBC Grid data reaers to associate model data with the mesh grid.

Parameters

- **mesh** (*vtkRectilinearGrid*) – The `vtkRectilinearGrid` that is the mesh to place the model data upon.
- **model** (*np.array*) – A NumPy float array that holds all of the data to place inside of the mesh's cells.
- **dataNm** (*str*) – The name of the model data array once placed on the `vtkRectilinearGrid`.

Returns Returns the input `vtkRectilinearGrid` with model data appended.

Return type vtkRectilinearGrid

RequestData(*request*, *inInfo*, *outInfo*)
Handles data request by the pipeline.

RequestInformation(*request*, *inInfo*, *outInfo*)
Handles info request by pipeline about timesteps and grid extents.

_TensorMeshReader__ubcMeshData2D(*FileName_Mesh*, *FileName_Models*, *output*)
Helper method to read a 2D mesh

_TensorMeshReader__ubcMeshData3D(*FileName_Mesh*, *FileName_Models*, *output*)
Helper method to read a 3D mesh

_TensorMeshReader__ubcTensorMesh(*FileName_Mesh*, *FileName_Models*, *output*)
Wrapper to Read UBC GIF 2D and 3D meshes. UBC Mesh 2D/3D models are defined using a 2-file format. The "mesh" file describes how the data is descritized. The "model" file lists the physical property values for all cells in a mesh. A model file is meaningless without an associated mesh file. If the mesh file is 2D, then then model file must also be in the 2D format (same for 3D).

Parameters

- **FileName_Mesh** (*str*) – The mesh filename as an absolute path for the input mesh file in UBC 2D/3D Mesh Format
- **FileName_Models** (*str or list(str)*) – The model filename(s) as an absolute path for the input model file in UBC 2D/3D Model Format.
- **output** (*vtkRectilinearGrid*) – The output data object

Returns Returns a `vtkRectilinearGrid` generated from the UBC 2D/3D Mesh grid. Mesh is defined by the input mesh file. Cell data is defined by the input model file.

Return type vtkRectilinearGrid

static ubcMesh2D(*FileName*, *output*)

This method reads a UBC 2D Mesh file and builds an empty `vtkRectilinearGrid` for data to be inserted into. Format Specs.

Parameters

- **FileName** (*str*) – The mesh filename as an absolute path for the input mesh file in UBC 3D Mesh Format.
- **output** (*vtkRectilinearGrid*) – The output data object

Returns a `vtkRectilinearGrid` generated from the UBC 3D Mesh grid. Mesh is defined by the input mesh file. No data attributes here, simply an empty mesh. Use the `PlaceModelOnMesh()` method to associate with model data.

Return type vtkRectilinearGrid

static ubcMesh3D(*FileName*, *output*)

This method reads a UBC 3D Mesh file and builds an empty `vtkRectilinearGrid` for data to be inserted into.

Parameters

- **FileName** (*str*) – The mesh filename as an absolute path for the input mesh file in UBC 3D Mesh Format.
- **output** (*vtkRectilinearGrid*) – The output data object

Returns a `vtkRectilinearGrid` generated from the UBC 3D Mesh grid. Mesh is defined by the input mesh file. No data attributes here, simply an empty mesh. Use the `PlaceModelOnMesh()` method to associate with model data.

Return type vtkRectilinearGrid

static ubcModel2D(*FileName*)

Reads a 2D model file and returns a 1D NumPy float array. Use the `PlaceModelOnMesh()` method to associate with a grid.

Parameters **FileName** (*str*) – The model filename as an absolute path for the input model file in UBCMesh Model Format. Also accepts a list of string file names.

Returns a NumPy float array that holds the model data read from the file. Use the `PlaceModelOnMesh()` method to associate with a grid. If a list of file names is given then it will return a dictionary of NumPy float array with keys as the basenames of the files.

Return type np.array

4.10.3 two_file_base

Model Appender Base

class PVGeo.ubc.two_file_base.**ModelAppenderBase**(*inputType='vtkRectilinearGrid'*, *outputType='vtkRectilinearGrid'*, ***kwargs*)

Bases: *PVGeo.base.AlgorithmBase*

A base class for create mesh-model appenders on the UBC Mesh formats

AddModelFileName(*fname*)
Use to set the file names for the reader. Handles singlt string or list of strings.

ClearModels()
Use to clear data file names.

GetModelFileNames(*idx=None*)
Returns the list of file names or given and index returns a specified timestep's filename.

GetTimestepValues()
Use this in ParaView decorator to register timesteps.

HasModels()

Modified(*readAgain=True*)
Call modified if the files needs to be read again again.

NeedToRead(*flag=None*)
Ask self if the reader needs to read the files again

Parameters **flag** (*bool*) – if the flag is set then this method will set the read status

Returns The status of the reader aspect of the filter.

Return type bool

RequestData(*request*, *inInfo*, *outInfo*)
DO NOT OVERRIDE

RequestInformation(*request*, *inInfo*, *outInfo*)
DO NOT OVERRIDE

SetDataName(*name*)

_ModelAppenderBase__SetInputTimesteps()

_ModelAppenderBase__UpdateTimeSteps()
For internal use only: appropriately sets the timesteps.

_PlaceOnMesh(*output*, *idx=0*)

_ReadUpFront()

UBC Mesh Reader Base

class PVGeo.ubc.two_file_base.**ubcMeshReaderBase**(*nOutputPorts=1*, *outputType='vtkUnstructuredGrid'*, ***kwargs*)

Bases: *PVGeo.base.TwoFileReaderBase*

A base class for the UBC mesh readers

GetDataName()

Is2D()

Is3D()

SetDataName(*name*)
Set te data array name for the model data on the output grid.

_ReadExtent()
Reads the mesh file for the UBC 2D/3D Mesh or OcTree format to get output extents. Computationally inexpensive method to discover whole output extent.

Returns This returns a tuple of the whole extent for the grid to be made of the input mesh file (0,n1-1, 0,n2-1, 0,n3-1). This output should be directly passed to set the whole output extent.

Return type tuple

static _ubcMesh2D_part(*FileName*)

static ubcModel3D(*FileName*)
Reads the 3D model file and returns a 1D NumPy float array. Use the PlaceModelOnMesh() method to associate with a grid.

Parameters **FileName** (*str*) – The model file name(s) as an absolute path for the input model file in UBC 3D Model Model Format. Also accepts a *list* of string file names.

Returns Returns a NumPy float array that holds the model data read from the file. Use the `PlaceModelOnMesh()` method to associate with a grid. If a list of file names is given then it will return a dictionary of NumPy float array with keys as the basenames of the files.

Return type np.array

4.10.4 write

Write `vtkImageData` to UBC Tensor Mesh

class PVGeo.ubc.write.**WriteImageDataToUBC**
Bases: `PVGeo.ubc.write.ubcTensorMeshWriterBase`

Writes a `vtkImageData` (uniform grid) data object to the UBC Tensor Mesh format. This file reader currently only handles 3D data.

RequestData(*request*, *inInfoVec*, *outInfoVec*)
OVERWRITE: This is executed by the pipeline and handles the write out

Write `vtkRectilinearGrid` to UBC Tensor Mesh

class PVGeo.ubc.write.**WriteRectilinearGridToUBC**
Bases: `PVGeo.ubc.write.ubcTensorMeshWriterBase`

Writes a `vtkRectilinearGrid` data object to the UBC Tensor Mesh format. This file reader currently only handles 3D data.

RequestData(*request*, *inInfoVec*, *outInfoVec*)
OVERWRITE: This is executed by the pipeline and handles the write out

4.11 Pipeline

4.11.1 Delete Downstream Filters

pvmacros.pipeline.**deleteDownstream**(*input=None*)
Delete downstream filters for a given input. If no input provided, all filters on the pipeline will be deleted.

Parameters **input** (*str*) – The name of the object on the pipeline to preserve.

4.12 Export

4.13 Visualization

4.13.1 axes

Custom Axis Ticks

pvmacros.vis.axes.**customAxisTicks**(*rng*, *axis=0*, *uniform=False*)
Use to set custm axis ticks in the render view

4.13.2 objs

Camera

class pvmacros.vis.objs.**camera**(*cam=None*)
An object to store a single camera location/view. You can make a list/dict of these objects to save interesting views for your project. This object saves just a few parameters about the camera so that it can easily be reconstructed.

_getFocalPoint()

_getOrientation()

_getPosition()

_getViewUp()

static loadViews(*path='/home/docs'*)
Load a file containg a serialized camera objects. Dafault loads from home directory if relative path

Parameters

- **filename** (*str*) – The file basename for the serialized file (defualt is default for output def)
- **path** (*str*) – The directory from which you wish to load the views. Defaults to user home directory for relative paths.

static saveViews(*filename='views'*, *path='/home/docs'*)
Save a serialized dictionaty/list/whatever of views out to a file. Dafault saves to user's home directory

Parameters

- **lib** (*dict or list*) – some iterable object containg multiple *camera* objects

- **filename** (*str*) – The file basename for the serialized file
- **path** (*str*) – The directory you wish to save the views. Defaults to user home directory

screenShot(*cam=None*, *path='/home/docs'*, *basenm='view'*)
Save a screenshot of a single camera view

Parameters

- **cam** (*vtkRenderingOpenGL2Python.vtkOpenGLCamera*) – The camera you wish to view then save a screenshot
- **path** (*str*) – The directory you wish to save the screenshot. Defaults to user home directory
- **basenm** (*str*) – The file basename for the screenshot

static screenShotViews(*cam=None*, *path='/home/docs'*, *basenm='view'*)
Save screenshots of many views/cameras

Parameters

- **d** (*view*) – some iterable object containg multiple *camera* objects
- **cam** (*vtkRenderingOpenGL2Python.vtkOpenGLCamera*) – The camera you wish to view then save a screenshot
- **path** (*str*) – The directory you wish to save the screenshot. Defaults to user home directory
- **basenm** (*str*) – The file basename for the screenshot

update(*cam=None*)
Updates the camera location to that which is in the currently activated view unless a vtkOpenGLCamera is specified.

Parameters **cam** (*vtkRenderingOpenGL2Python.vtkOpenGLCamera*) – The camera you wish to update this object to. Totally optional

view(*cam=None*)
Use this method to update the camera to the saved location

Parameters **cam** (*vtkRenderingOpenGL2Python.vtkOpenGLCamera*) – The camera you wish to view/update in the current render view

Python Module Index

p

Index

Symbols

A

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X