



ParaView Scripting

David E DeMarle
Kitware Inc.



Why?

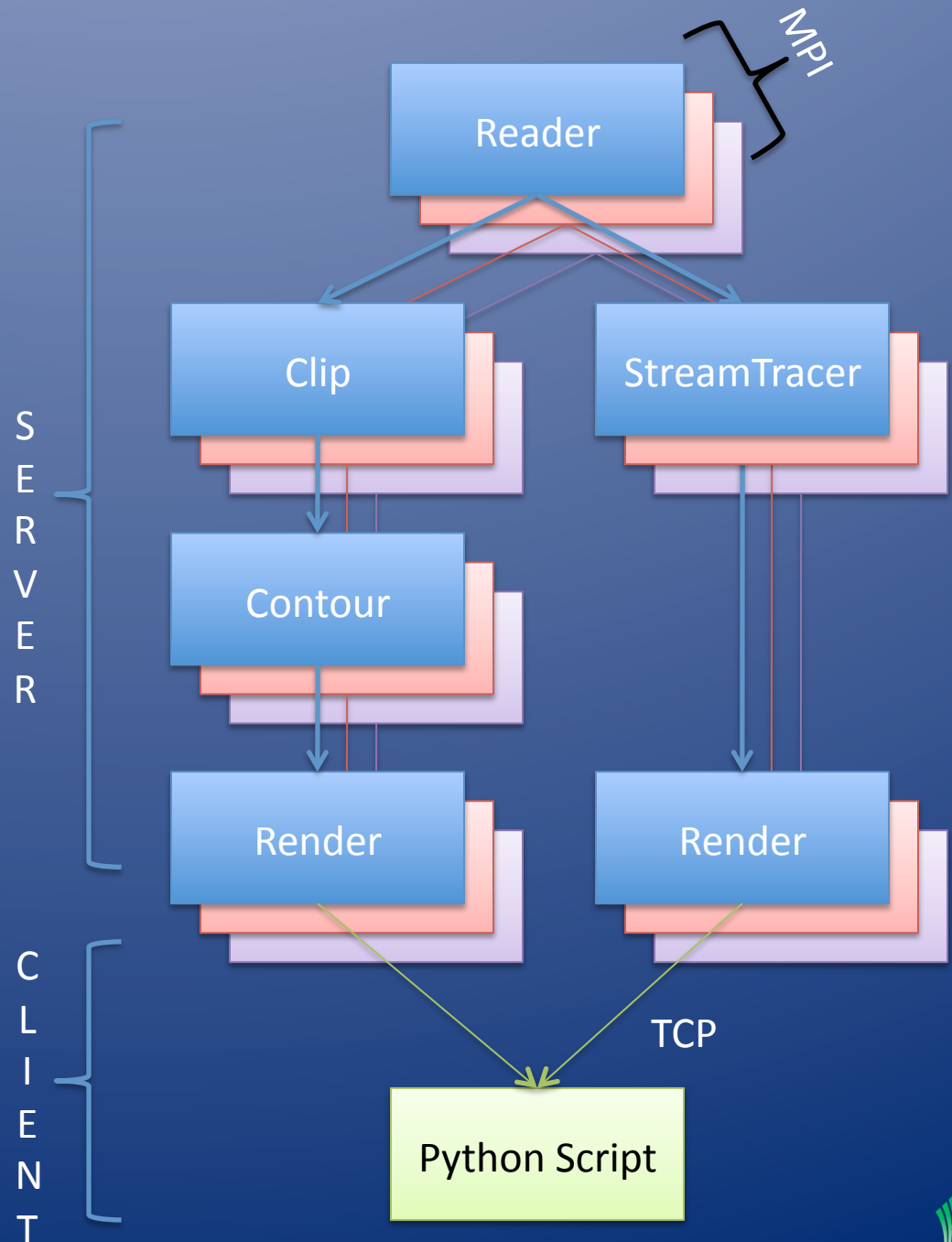
- Run in Batch mode
 - Set up Vis task on small representative dataset locally
 - Repeat with real data on supercomputer
- Script arbitrary parallel processing tasks
 - Not just visualization
 - A parallel interpreted programming environment
 - Examine, change and act upon individual data values in huge data sets
- To interface to use ParaView with other tools
- Script alongside the GUI



ParaView Scripting

VTK Pipeline,
in parallel,
on remote server(s),

controlled by and feeds
into python script.

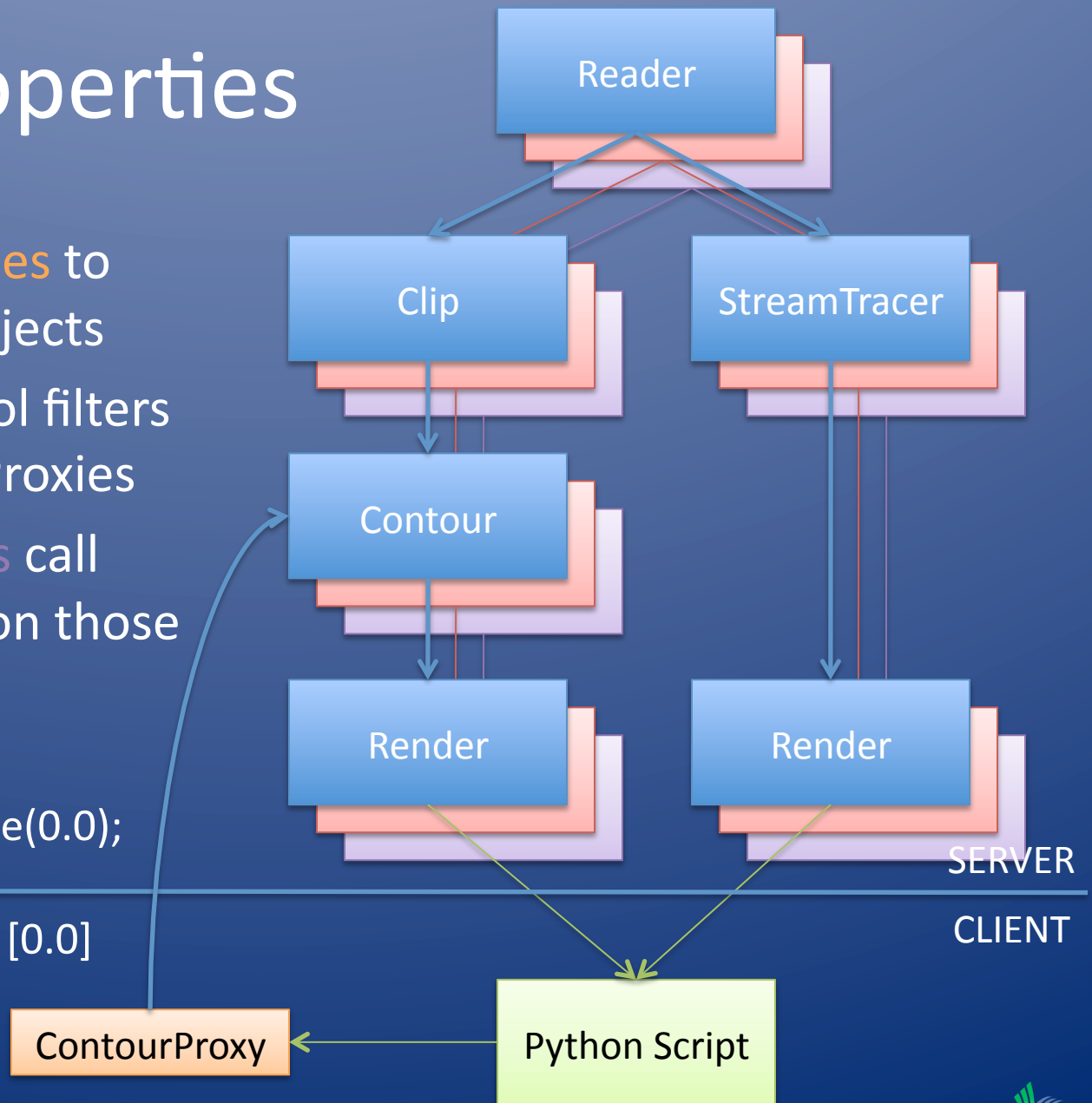


Proxies, Properties

- Client makes **Proxies** to control remote objects
- Proxies that control filters are called SourceProxies
- Proxies' **Properties** call specific methods on those objects

```
vtkContourFilter::SetValue(0.0);
```

```
Contour.ContourValues = [0.0]
```



How it Works

- Python wrapping of VTK
 - All public methods* of concrete classes callable
 - Unfortunately, can only call them on objects that live on client
- Python wrapping of ServerManager (SM)
 - vtkServerManager library is what allows client to control remote VTKSMSOURCEPROXY, VTKSMPROPERTY, etc
 - At this level you can control things on server
- Layered modules on top of wrapping simplify life
 - >>> import paraview (deprecated, PV <= 3.2)
 - >>> from paraview import servermanager (deprecated, PV <= 3.4)
 - >>> from paraview.simple import * (New and Improved! PV >= 3.6)

* That do not take pointer arguments, are not within //BTX ... //ETX, and are not in manually excluded files



How to Use it

- Shell within GUI
 - Tools->Python Shell
 - Fixed to same server that GUI is connected to
- Any python interpreter
 - Set PATHS to include ParaView libraries (bin and Utilities/VTKPythonWrapping)
- pvpython
 - python interpreter that comes with ParaView
 - Paths are set automatically
- pvbatch
 - MPI pvpython
 - Made to run on supercomputer
 - Can not interact with it, must give it filename of a script to run
 - Can not change server (no TCP) it actually runs inside the server
- All: Start script with “>>> from paraview.simple import *”

WARNING! None have event loop -> No mouse, just good old command line



Using External Interpreter

- Mac/Linux

```
% set PVBUILD=/Builds/ParaView/devel/build
```

```
% export PATH=${PATH}:${PVBUILD}/bin
```

```
% export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${PVBUILD}/bin
```

```
% export PYTHONPATH=${PVBUILD}/bin:
```

```
% export PYTHONPATH=${PYTHONPATH}:${PVBUILD}/Utilities/VTKPythonWrapping
```

- Windows

Start->Control Panel->Performance and Maintenance ->

System->Advanced->Environment Variables

Add new user variable PVBUILD C:\Builds\ParaView\devel\build

Add/Edit user variable PATH %PVBUILD%\Debug\bin

Add/Edit user variable PYTHONPATH %PVBUILD%\Debug\bin;

Edit user variable PYTHONPATH %PVBUILD%\Utilities\VTKPythonWrapping



Where to Start?

- build a pipeline by creating SourceProxies*
- Use properties to inspect and change the filters settings
- Properties are often assigned at instantiation

```
>>> myCone = Cone()
```

```
>>> print myCone.Center
```

```
>>> myCone.Center = [10,10,10]
```

```
>>> aDuplicateCone = Cone(Center=[10,10,10])
```

* SourceProxy – ParaView term for any reader, procedural generator, filter, or writer



Getting Help

- `help(paraview.simple)`
lists all functions that `paraview.simple` gives you including all `SourceProxies`
- `help(Cone)`
gives top level information about `Cones` (the class)
- `help(myCone)`
gives more details (ex properties you can access) when you give it a particular `Cone` (the object)
- `dir(myCone)` compact and sometimes more complete alternative
- `print(myCone)` sometimes gives more details about member values



About Properties

- Properties are python-esque

```
>>> myCone.Center = [0,0,0]
>>> myCone.Center[0] = [1]
>>> myCone.Center[2:3] = [2,3]
```

- VTK and ParaView are lazily evaluated. You don't see results until you tell Pipeline to run

```
>>> myCone.Radius = 2.0
>>> Show(myCone)
>>> Render()
>>> myCone.Radius = 0.1
>>> #!? Why no change?
>>> Render()
```



Building a Pipeline

- Like in GUI, build on top of the "Active" source

```
>>>aReader =  
XMLStructuredGridReader(  
    FileName="multicomb_0.vts")
```



- Set Properties as you go, like editing Property Tab in GUI's ObjectInspector

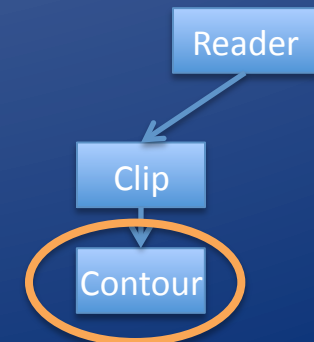
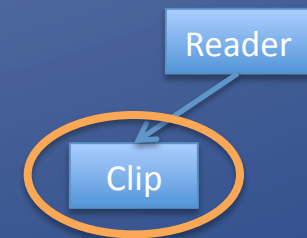
```
>>> aClip = Clip()
```

```
>>> aClip.ClipType.Normal =  
[0,-1,0]
```

```
>>> aContour = Contour()
```

```
>>> aContour.ContourBy =  
"Density"
```

```
>>> aContour.Isosurfaces = [0.5]
```



Building a Pipeline

- Branch by changing the active source, like choosing in GUI's PipelineBrowser

```
>>> SetActiveSource(aReader)
```

```
>>> aST = StreamTracer()
```

```
>>> aST.Vectors = "Momentum"
```

```
>>> aST.SeedType.Center = [3,2,28]
```

```
>>> aST.SeedType.Radius = 2
```

```
>>> aST.SeedType.NumberOfPoints  
= 100
```

- Unlike in GUI, displays are not automatically made or refreshed

```
>>> Show(aContour)
```

```
>>> Show(aStreamTracer)
```

```
>>> Render()
```



Navigating the Pipeline

- Don't have to use active source to branch, can assign at creation

```
>>> aST = StreamTracer(Input=aReader)
```

- Can change after the fact

```
>>> aST = StreamTracer()
```

```
>>> aST.Input = aReader
```

```
>>> aST.Input = aClip
```

- Can inspect ActiveSource

```
>>> aSource = GetActiveSource()
```

- Can get a hold of all or any particular SourceProxy

```
>>> GetSources()
```

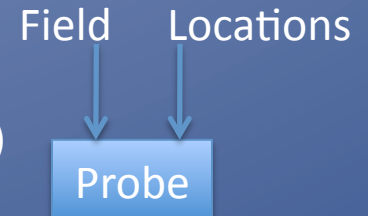
```
>>> someSource = FindSource("Contour1")
```



Merging and Multiplicity

- Some SourceProxies require multiple inputs, usually named “Input” and “Source”, but not always

```
>>> probe = ProbeLocation()  
>>> probe.Input = Mandebrot()  
>>> probe.ProbeType = Sphere()
```



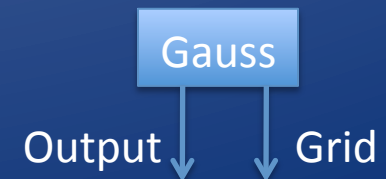
- Some SourceProxies have inputs that are repeatable, use array notation to assign them

```
>>> append = AppendGeometry()  
>>> append.Input = [poly1, poly2]
```



- A few SourceProxies produce multiple outputs, use array notation to retrieve them

```
>>> reader =  
    GaussianCubeReader(filename="my.cube")  
>>> shrink=Shrink()  
>>> shrink.Input=reader[1]
```



Displaying Results

- Can show output of any SourceProxy
- Parallel Flexible Display Pipeline complexity encapsulated by “Representations” in “Views”

Representation – visual qualities of an output

≈ Mapper + Actor + parallel transport

Show() returns a Representation

View - Visual qualities of a window

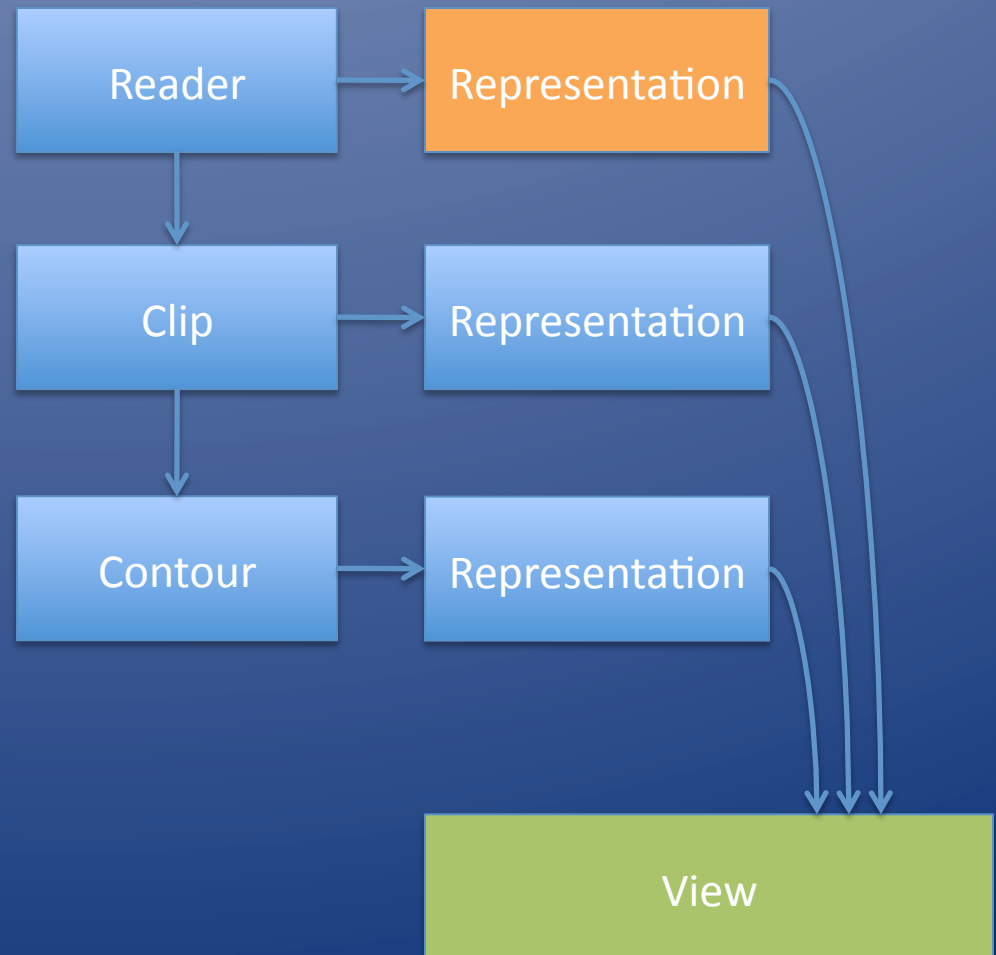
≈ Renderer + Camera + Lights + RenderWindow

Render() returns a View

- To make it easier to build, commands default to working with the

Active Representation

Active View



Controlling Display

- Change Properties of View and Representation Proxies to affect display
 - >>> aView = GetActiveView()
 - >>> help(aView)
 - >>> aView.Background = [0.0,0.0,0.0]
- Don't forget lazy evaluation
 - >>> Render()
- Visibility is particularly important, since all pipeline stages can be shown
 - >>> aRep = GetRepresentation()
 - >>> help(aRep)
 - >>> aRep.Visibility = 0
 - >>> aRep.Visibility = 1Show() and Hide() shortcuts set Visibility property



Controlling Display

- Many methods take `ActiveRepresentation` and `ActiveView` as default arguments (very Python-esque)

```
>>> activeSourcesRep = GetDisplayProperties()
>>> clipFiltersRep = GetDisplayProperties(aClip)
>>> clipFiltersRepInMyView =
    GetDisplayProperties(aClip, myView)
```

- But can get hold of and then control any particular `View` and `Representation`

```
>>> GetRenderViews()
>>> view0 = GetRenderViews()[0]
>>> allReps = view0.Representations()
>>> rep0_0 = allReps[0]
```



Camera

- RenderViews (not PlotViews etc) have Cameras
- View has properties to manipulate them
- Camera is actually a local VTK object that you can control with standard python wrapped VTK commands

```
>>> view0.CameraPosition = [16,0,51]
```

```
>>> cam = view0.GetActiveCamera()
```

```
>>> #or usually
```

```
>>> cam = GetActiveCamera()
```

```
>>> cam.GetPosition()
```

```
>>> cam.SetPosition(-16,0,51)
```

```
>>> print(cam)
```

```
>>> Render()
```



Rendering Modes

- A Representation's *Representation** property controls rendering mode:

Bounding Box

Points

Wireframe

Polygons (surface)

Volume Render

etc

```
>>> aRep = Show(aClip)
>>> aRep.GetProperty("Representation").
    Available
>>> aRep.Representation = 'Outline'
>>> Render()
>>> aRep.Representation = 'Points'
>>> Render()
>>> aRep.Representation = 'Wireframe'
>>> Render()
>>> aRep.Representation = 'Surface With Edges'
>>> Render()
>>> aRep.Representation = 'Surface'
>>> Render()
```

* Yes it is confusing, so top level is often called "DisplayProperty" or "Display Pipeline"



ColorMapping

- Representations have LookupTables that assign colors to values

```
>>> aRep.ColorArrayName = 'Density'  
>>> aRep.LookupTable=  
      MakeBlueToRedLT(0,1)  
>>> Render()
```
- MakeBlueToRedLT(min,max) is a convenient way to make one
- You can design your own if you need to:
 - Pick an array to color with
 - Pick the value ranges
 - Pick the colors

```
>>> aRep.ColorAttributeType='POINT_DATA'  
>>> aRep.ColorArrayName="Density"  
>>> lut =  
      servermanager.rendering.PVLookupTable()  
>>> aRep.LookupTable = lut  
>>> #           value,      R,G,B  
>>> lut.RGBPoints = [ 0.0,    0, 0, 1,  
                      0.1,    0.5,0,0.5  
                      1.0,    1, 0, 0]
```



Getting information

- ParaView has a client server architecture, and is lazily evaluated (designed for large data)
- You have to ask ParaView politely if you want results back from server (other than display)
- Three ways ways to get quantitative results back
 - Information properties
 - DataInformation
 - Fetch



Information Properties

- **Properties**

Most VTK methods on server
set parameters

SETFILENAME(), SETCONTOURS()

```
>>> aReader.FileName = "multicomb_0.vts"  
>>> print aReader.FileName  
foo.ex2  
>>> #does not ask server, just remembers  
>>> #what we set
```

- **Information Properties**

Some VTK methods return
simple results

GETFILENAME(),
GETNUMBEROFPOINTS()

```
>>> aReader.UpdatePropertyInformation()  
>>> print aReader.TimestepValues
```

Information Properties let the
client read these results

```
>>> aST2 = StreamTracer(Input=aReader)  
>>> aST2.UpdatePipeline()  
>>> aST2.UpdatePropertyInformation()  
>>> aST2.GetProperty("NumberOfPoints")
```

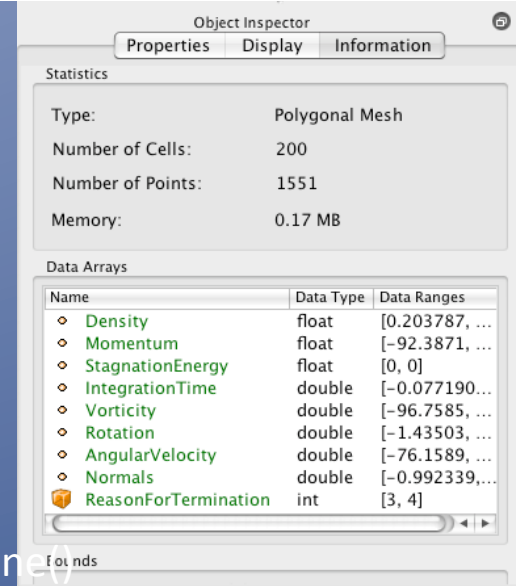


Data Information

- Data Information

- What GUI shows in Pipeline Browser's Information Tab
- What GUI uses to assign filter default settings
- Meta-Information about output of a SourceProxy
 - CLASSNAME,
 - MEMORYSIZE,
 - EXTENT,
 - NUMCELLS, NUMPOINTS,
 - ARRAYS, ARRAY NAMES, ARRAY RANGES
- Can't get individual values out

```
>>> aReader.UpdatePipeline()
>>> aReader.UpdatePipelineInformation()
>>> dInfo = aReader.GetDataInformation()
>>> dInfo.GetDataClassName()
>>> pdInfo = dInfo.GetPointDataArrayInformation()
>>> pdInfo.GetNumberOfArrays()
>>> ai0 = pdInfo.GetArrayInformation(0)
>>> ai0.GetName()
>>> ai0.GetNumberOfComponents()
>>> ai0.GetComponentRange(0)
>>> ai0.GetNumberOfTuples()
```



Fetch

- Copies entire DataSet from server to Client

```
>>> output = servermanager.Fetch(aClip)
```
- Once local, you can manipulate the data with python Wrapped VTK API and access individual data values

```
>>> print(output)
```
- Since data is large, don't often want whole data set on client

```
>>> processor1sOutput =  
servermanager.Fetch(aClip,1)
```
- Can also do some simple aggregation of attribute values
Just specify an aggregator function to apply on the way

```
>>> mm = MinMax()  
>>> mm.Operation = "MIN"  
>>> minResult = servermanager.Fetch(elev, mm, mm)  
>>> a0 = minResult.GetPointData().GetArray(1)  
>>> a0.GetName()  
>>> a0.GetValue(0)
```



Now that you know...

- Choosing a server

Disconnect from one server (destroying pipeline there) and connect to another.

```
>>> Connect(host, portnum)
```

```
>>> Help(Connect)
```

- Writers

Save output of any SourceProxy on server's file system

```
>>> writer = XMLUnstructuredGridWriter()
```

```
>>> writer.FileName = "foo.pvtk"
```

```
>>> writer.Input = aClip
```

```
>>> writer.UpdatePipeline()
```



Features I'm skipping

- Screen Shots

```
>>> WriteImage(filename, view==ActiveView, Magnification==0.0)
```

- Animation

Create key frames in tracks and automatically animate through them. Like GUI's Animation View

```
>>> scene = servermanager.animation.AnimationScene()
```

```
>>> track1 = servermanager.animation.KeyFrameAnimationCue()
```

```
>>> keyframe1 = servermanager.animation.CompositeKeyFrame()
```

```
>>> track1.KeyFrames = [keyframe1, keyframe2]
```

```
>>> scene.Cues = [track1]
```

- Movies

Save as series of screenshots or into a movie file*

```
>>> AnimateReader(reader, view, "myMovie.png")
```

* Assuming your ParaView has a codec, otherwise limited to numbered screenshots



Even More Features

- State

Save state in GUI, load it in python (and vice-versa)

```
>>> servermanager.LoadState("myteststate.pvsm")
```

```
>>> SetActiveView(GetRenderView())
```

```
>>> Render()
```

- Python Programmable Filter

A white box filter

Arbitrary scripted parallel processing

Numerous examples on wiki



Python Programmable Filter

runs inside a filter's RequestData() on server
python wrapped VTK API

Get hold of input and output DataSet(s)
examine geometry, topology and attributes
Do some arbitrary calculation

```
inDS = self.GetInput()
outDS = self.GetOutput()
inPtData = inDS.GetPointData()
outPtData = outDS.GetPointData()
outPtData.ShallowCopy(inPtData)

inPTArray = inPointData.GetArray("Name")
array = vtk.vtkDoubleArray()
array.SetName("Another Name")
outPtData.AddArray(array)
```

```
>>> pfilter = ProgrammableFilter()
>>> pfilter.Script = """
pdi = self.GetPolyDataInput()
pdo = self.GetPolyDataOutput()
newPoints = vtk.vtkPoints()
numPoints = pdi.GetNumberOfPoints()
for i in range(0, numPoints):
    coord = pdi.GetPoint(i)
    x, y, z = coord[:3]
    x = x * 1
    y = y * 1
    z = 1 + z*0.3
    newPoints.InsertPoint(i, x, y, z)
pdo.SetPoints(newPoints)
"""
```



Getting More Help

- **Wiki Page**
 - <http://www.paraview.org/Wiki/ParaView>
- **Source Code Documentation**
 - <http://www.paraview.org/ParaQ/Doc/Nightly/html/annotated.html>
- **Mailing List**
 - Sign up->
<http://public.kitware.com/mailman/listinfo/paraview>
 - Search -><http://markmail.org/search/?q=list:paraview>
- **Bug Tracker**
 - http://www.paraview.org/Bug/my_view_page.php
 - Project:-> ParaView3

