



12th Summer
School on
SCIENTIFIC
VISUALIZATION

Paraview scripting

Raffaele Ponzini - [r.ponzini@cineca.it](mailto:r.ponzini@ Cineca.it)
SuperComputing Applications and Innovation Department





OUTLINE

- Why scripting
- pvbatch and pvpython
- Macros
- Scripting using a tracefile
- Journaling in Paraview GUI
- Hands-on Python Scripting in ParaView



WHY SCRIPTING

- Automate repetitive tasks (several similar datasets)
- GUI is not useful or unavailable (batch execution)
- In-situ visualization or co-processing (computing+post-processing)



pvbatch - pvpython

- Paraview contents 2 command lines able to run a python script:
 - pvbatch;
 - pvpython;

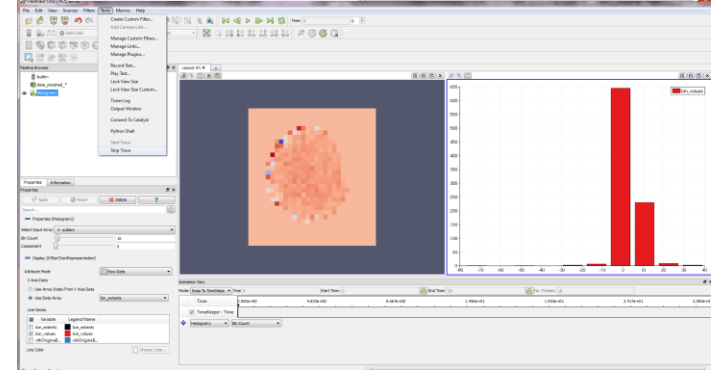
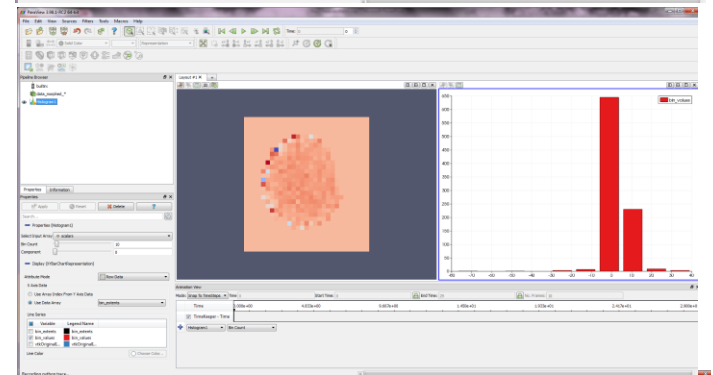
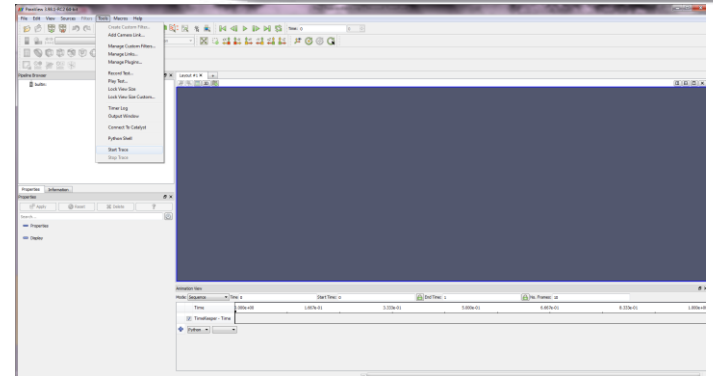
The differences is that:

- pvpython is equivalent to the paraview client where the python interpreter plays the role of the GUI (interactive, serial);
- pvbatch: is equivalent to the paraview server where the commands are coming from the python script instead then from a socket plugged to a paraview client (parallel)



BUILD A PYTHON SCRIPT IN THE PARAVIEW GUI

1. Start tracing
2. Build your pipeline as usual
3. Stop tracing and save the trace file with a name

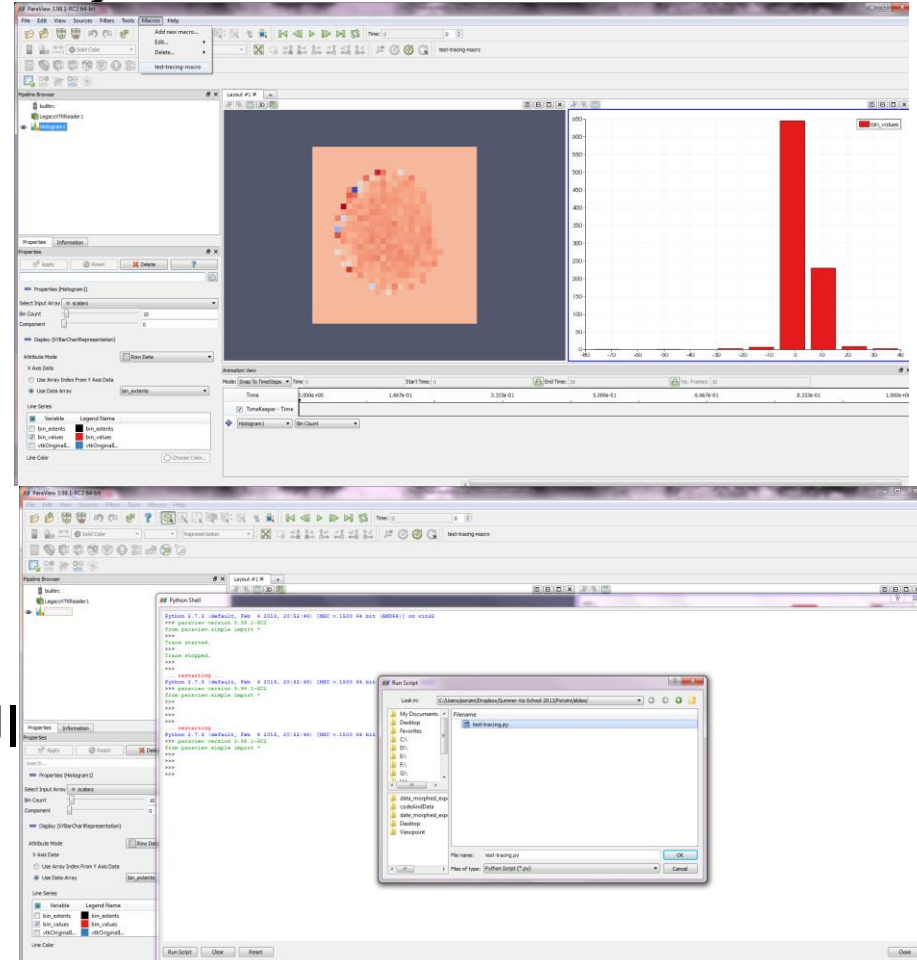




Use the trace file

You can use the traced file in two ways:

- As a macro
- As a python script using the GUI
- As a python script in batch





The `paraview.simple` module

- The first thing any ParaView Python script must do is load the `paraview.simple` module.

```
from paraview.simple import *
```

Or in a safe way using the try/except construct

```
try: paraview.simple
```

```
except: from paraview.simple import *
```

- This command is automatically invoked in the ParaView GUI tracing facility, but it must add when you are writing a script to be used in batch either by `pvpython` or by `pvbatch`.
- The `paraview.simple` module contains a function for every source, reader, filter, and writer with off-course the same name as shown in the GUI menus but with spaces and special characters removed.
- Each function creates a pipeline object, which will show up in the pipeline browser (with the exception of writers), and returns an object that is a proxy that can be used to query and manipulate the properties of that pipeline object.



BATCH SCRIPTING IN PYTHON

```
AnimationScene1 = GetAnimationScene()
AnimationScene1.EndTime = 29.0
AnimationScene1.PlayMode = 'Snap To TimeSteps'

RenderView1 = GetRenderView()
a1_scalars_PVLookupTable = GetLookupTableForArray( "scalars", 1, RGBPoints=[-77.78269958496094, 0.23, 0.299, 0.754,
39.99039840698242, 0.706, 0.016, 0.15], VectorMode='Magnitude', NanColor=[0.25, 0.0, 0.0], ColorSpace='Diverging',
ScalarRangeInitialized=1.0, AllowDuplicateScalars=1 )

a1_scalars_PiecewiseFunction = CreatePiecewiseFunction( Points=[0.0, 0.0, 0.5, 0.0, 1.0, 1.0, 0.5, 0.0] )

DataRepresentation1 = Show()
DataRepresentation1.EdgeColor = [0.0, 0.0, 0.5000076295109483]
DataRepresentation1.SelectionPointFieldDataArrayName = 'scalars'
DataRepresentation1.ScalarOpacityFunction = a1_scalars_PiecewiseFunction
DataRepresentation1.ColorArrayName = 'scalars'
DataRepresentation1.ScalarOpacityUnitDistance = 4.349219049453166
DataRepresentation1.LookupTable = a1_scalars_PVLookupTable
DataRepresentation1.Representation = 'Slice'
DataRepresentation1.ScaleFactor = 3.0

RenderView1.CameraFocalPoint = [14.0, 15.0, 0.0]
RenderView1.CameraPosition = [14.0, 15.0, 10000.0]
RenderView1.InteractionMode = '2D'
RenderView1.CenterOfRotation = [14.0, 15.0, 0.0]
```



BATCH SCRIPTING IN PYTHON

```
a1_scalars_PVLookupTable.ScalarOpacityFunction = a1_scalars_PiecewiseFunction

ScalarBarWidgetRepresentation1 = CreateScalarBar( Title='scalars', LabelFontSize=12, Enabled=1, LookupTable=a1_scalars_PVLookupTable,
TitleFontSize=12 )
GetRenderView().Representations.append(ScalarBarWidgetRepresentation1)

RenderView1.CameraPosition = [14.0, 15.0, 79.27656374961008]
RenderView1.CameraClippingRange = [78.48379811211397, 80.46571220585423]
RenderView1.CameraParallelScale = 20.518284528683193

a1_scalars_PVLookupTable.RGBPoints = [-100.0, 0.23, 0.299, 0.754, 100.0, 0.706, 0.016, 0.15]
a1_scalars_PVLookupTable.LockScalarRange = 1

a1_scalars_PiecewiseFunction.Points = [-100.0, 0.0, 0.5, 0.0, 100.0, 1.0, 0.5, 0.0]

WriteAnimation('C:/Users/ponzini/Dropbox/Summer-Viz-School-2013/Ponzini/tutorial/animation.avi', Magnification=1, Quality=2,
FrameRate=1.000000)

Render()
```

BATCH SCRIPTING IN PYTHON

The screenshot shows the ParaView 3.98.1-RC2 64-bit interface. The main window displays a pipeline with a 'Sphere' source. A 'Python Shell' window is open, showing the following code and output:

```

Python 2.7.3 (default, Feb  4 2013, 20:52:48) [MSC v.1500 64 bit (AMD64)] on win32
>>> paraview version 3.98.1-RC2
from paraview.simple import *
>>> help(Sphere)
Help on function CreateObject in module paraview.simple:

CreateObject(*input, **params)
    The Sphere source can be used to add a polygonal sphere to the 3D scene. The output of the Sphere source is polygonal data with point normals defined.
    This function creates a new proxy. For pipeline objects that accept inputs, all non-keyword arguments are assumed to be inputs. All keyword arguments are assumed to be property.value pairs and are passed to the new proxy.

>>> help(Show)
Help on function Show in module paraview.simple:

Show(proxy=None, view=None, **params)
    Turns the visibility of a given pipeline object on in the given view.
    If pipeline object and/or view are not specified, active objects are used.

>>> help(Render)
Help on function Render in module paraview.simple:

Render(view=None)
    Renders the given view (default value is active view)

>>> help(Hide)
Help on function Hide in module paraview.simple:

Hide(proxy=None, view=None)
    Turns the visibility of a given pipeline object off in the given view.
    If pipeline object and/or view are not specified, active objects are used.

>>> mysphere=Sphere()
>>> dir(mysphere)
['CellData', 'Center', 'EndPhi', 'EndTheta', 'FieldData', 'FileNameChanged', 'GetCellDataInformation', 'GetDataInformation', 'GetFieldDataInformation', 'GetPointDataInformation', 'GetProperty', 'GetPropertyValue', 'Initialize',
 'InitializeFromProxy', 'ListProperties', 'Observed', 'ObserverTag', 'PhiResolution', 'PointData', 'Port', 'Radius', 'SMProxy', 'SetPropertyWithName', 'StartPhi', 'StartTheta', 'ThetaResolution', 'UpdatePipeline',
 'UpdatePipelineInformation', '_Proxy_ConvertArgumentsAndCall', '_Proxy_GetActiveCamera', '_Proxy_LastAttributeName', '_Proxy_Properties', '__class__', '__del__', '__delattr__', '__dict__', '__doc__', '__eq__', '__format__',
 '__getattr__', '__getattribute__', '__getitem__', '__hash__', '__init__', '__iter__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
 '__weakref__', 'add_attribute']
>>> print mysphere.Radius
0.5
>>> print mysphere.PointData
<paraview.servermanager.FieldDataInformation object at 0x00000000178BAC60>
>>> print mysphere.PointData()
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: 'FieldDataInformation' object is not callable
>>> print mysphere.s

```

A context menu is visible over the code, with the following options:

- SMProxy
- SetPropertyWithName
- StartPhi
- StartTheta

The 'Run Script' button is highlighted in the Python Shell window.



BATCH SCRIPTING IN PYTHON

Python 2.7.3 (default, Feb 4 2013, 20:52:48) [MSC v.1500 64 bit (AMD64)] on win32

```
>>> paraview version 3.98.1-RC2
```

```
from paraview.simple import *
```

```
>>> help(Sphere)
```

Help on function CreateObject in module paraview.simple:

```
CreateObject(*input, **params)
```

The Sphere source can be used to add a polygonal sphere to the 3D scene. The output of the Sphere source is polygonal data with point normals defined.

This function creates a new proxy. For pipeline objects that accept inputs, all non-keyword arguments are assumed to be inputs. All keyword arguments are assumed to be property,value pairs and are passed to the new proxy.

```
>>> help(Show)
```

Help on function Show in module paraview.simple:

```
Show(proxy=None, view=None, **params)
```

Turns the visibility of a given pipeline object on in the given view.

If pipeline object and/or view are not specified, active objects are used.

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

Help on function Render in module paraview.simple:

```
Render(view=None)
```

Renders the given view (default value is active view)

```
>>> help(Hide)
```

Help on function Hide in module paraview.simple:

```
Hide(proxy=None, view=None)
```

Turns the visibility of a given pipeline object off in the given view.

If pipeline object and/or view are not specified, active objects are used.



BATCH SCRIPTING IN PYTHON

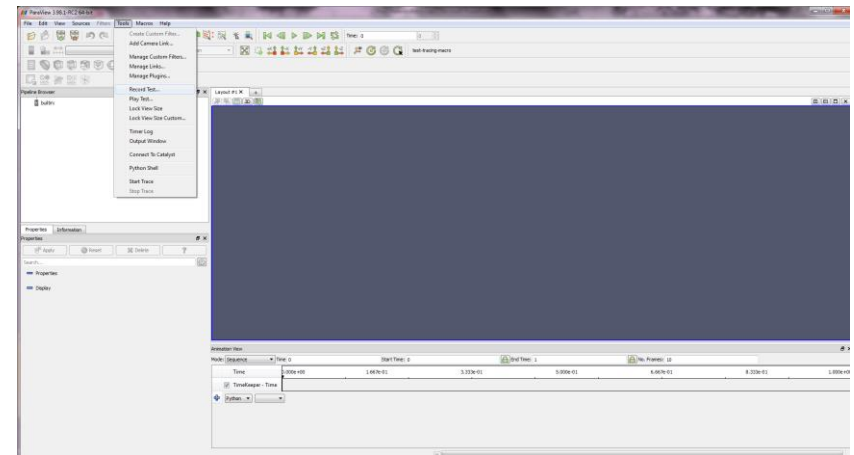
```
>>> mysphere=Sphere()
>>> dir(mysphere)
['CellData', 'Center', 'EndPhi', 'EndTheta', 'FieldData', 'FileNameChanged', 'GetCellDataInformation', 'GetDataInformation',
'GetFieldDataInformation', 'GetPointDataInformation', 'GetProperty', 'GetPropertyValue', 'Initialize', 'InitializeFromProxy', 'ListProperties',
'Observed', 'ObserverTag', 'PhiResolution', 'PointData', 'Port', 'Radius', 'SMPProxy', 'SetPropertyWithName', 'StartPhi', 'StartTheta',
'ThetaResolution', 'UpdatePipeline', 'UpdatePipelineInformation', '_Proxy__ConvertArgumentsAndCall', '_Proxy__GetActiveCamera',
'_Proxy__LastAttrName', '_Proxy__Properties', '__class__', '__del__', '__delattr__', '__dict__', '__doc__', '__eq__', '__format__', '__getattr__',
'__getattribute__', '__getitem__', '__hash__', '__init__', '__iter__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'add_attribute']
>>> print mysphere.Radius
0.5
>>> print mysphere.PointData
<paraview.servermanager.FieldDataInformation object at 0x0000000017BBAC50>
>>> Render()
<paraview.servermanager.RenderView object at 0x0000000018044FD0>
>>> Show()
<paraview.servermanager.GeometryRepresentation object at 0x0000000017A50748>
>>>
```



Recording a Paraview pipeline

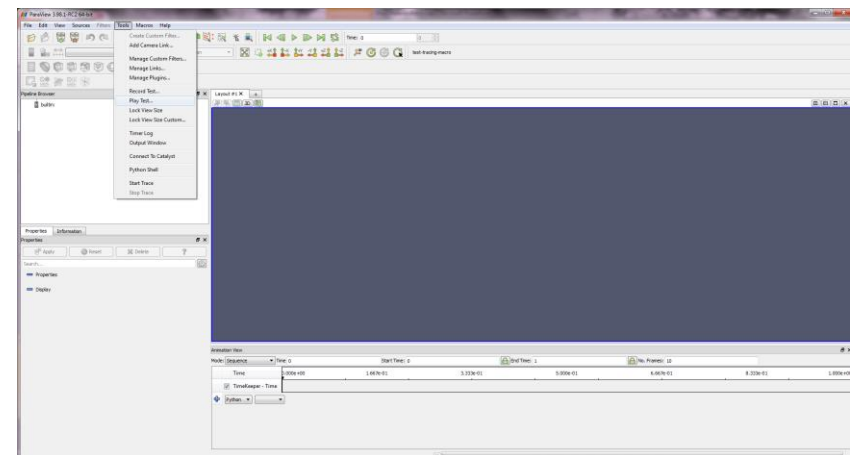
A similar (but different) solution to launch a pipeline in an automated way is to:

- record a test



and then to:

- Play a test





WORKOUT

- Build a script to process several VTK dataset according to the pipeline obtained during the tutorial of day-3
- Launch the script over the dataset to archive different animations
- Record a test and play it for the above mentioned operations