PyNode Visualisation

- A way of visualising sets of nodes, unlike the result plotter which can only visualise single nodes
- Possible by using:
  - A new plotter plugin in the meqbrowser
  - A base PyNode which attaches plotting information to the result object to be used by the plotter
- You create new visualisations by creating new PyNodes which define the data to plot, and how to plot it

What can be Plotted? (for now)

- Set of nodes (with multiple vellsets) against the node number
- Set of nodes (ideally one vellset) against each other
- Set of nodes with multiple vellsets within an argand plot (useful for cohaerency matrices for example)
- Set of nodes against a set of user-defined values
- A few nodes with vellsets within an history plot

Installing Things

- Make sure you have your Waterhole working
- Update Waterhole
- Open meqbrowser.py with a text editor so that we can use the new plotter:
  - ../Timba/install/symlinked-release/bin/meqbrowser.py
- Add the following import statement in import plugin section of the script:
  - import Timba.Contrib.AxM.pyvis.pynode_plotter

Plotting Examples

- Let's make sure that everything is working fine
- Start the meqbrowser and load the script:
  - Waterhole/contrib/AxM/pyvis/PyPlottableExamples.py
- Choose the first plotter in the compile options
- Load the only bookmark for this script
- Execute the script
- You should be able to see a curve

Plotting Examples

- There are five plotters in total available in the examples script
- These plotters (which can be cannabilised at will) provide an excellent demonstration of how to go about plotting whatever you require
- Some simple example trees are also provided
- After everyone is convinced that these things do actually look pretty cool, we can start creating our first PyPlotter

Creating Your First PyPlotter [1/5]

- A more detailed document can be found in Waterhole/contrib/AxM/pyvis
- The first steps:
  - Create a new class which inherits from PyBasePlottable
  - Constructor not required
  - Override the get_result method
- When you override any method, the first statement must be the parent classes' method call

Creating You First PyPlotter [2/5]

```
from Timba.Contrib.AxM.pyvis.PyBasePlottable import *

class PyMyPlotter(PyPlottableBase):

    def get_result(self, request, *children):

        super(PyMyPlotter, self).get_result(request, children)
```

- Now we need to create a ResultVector object, which is a helper class that encapsulates the list of pynode child results.

```
rv = ResultVector(children, labels = [str(i) for i in

                                      range(len(children))])
```

Creating Your First PyPlotter [3/5]

- Next we create the MeqResult object

```
vells = meq.vells(meq.shape(request.cells))

result = meq.result(meq.vellset(vells), request.cells)
```

- Now we define what to plot, on which axis
  - In this plotter we will plot the children's vellset means agains the child's index.
  - We need to define axis dictionaries which tells the plotter what results to use

```
y_axis = define_axis(expr = ‘means’)

x_axis = None
```

Creating Your First PyPlotter [4/5]

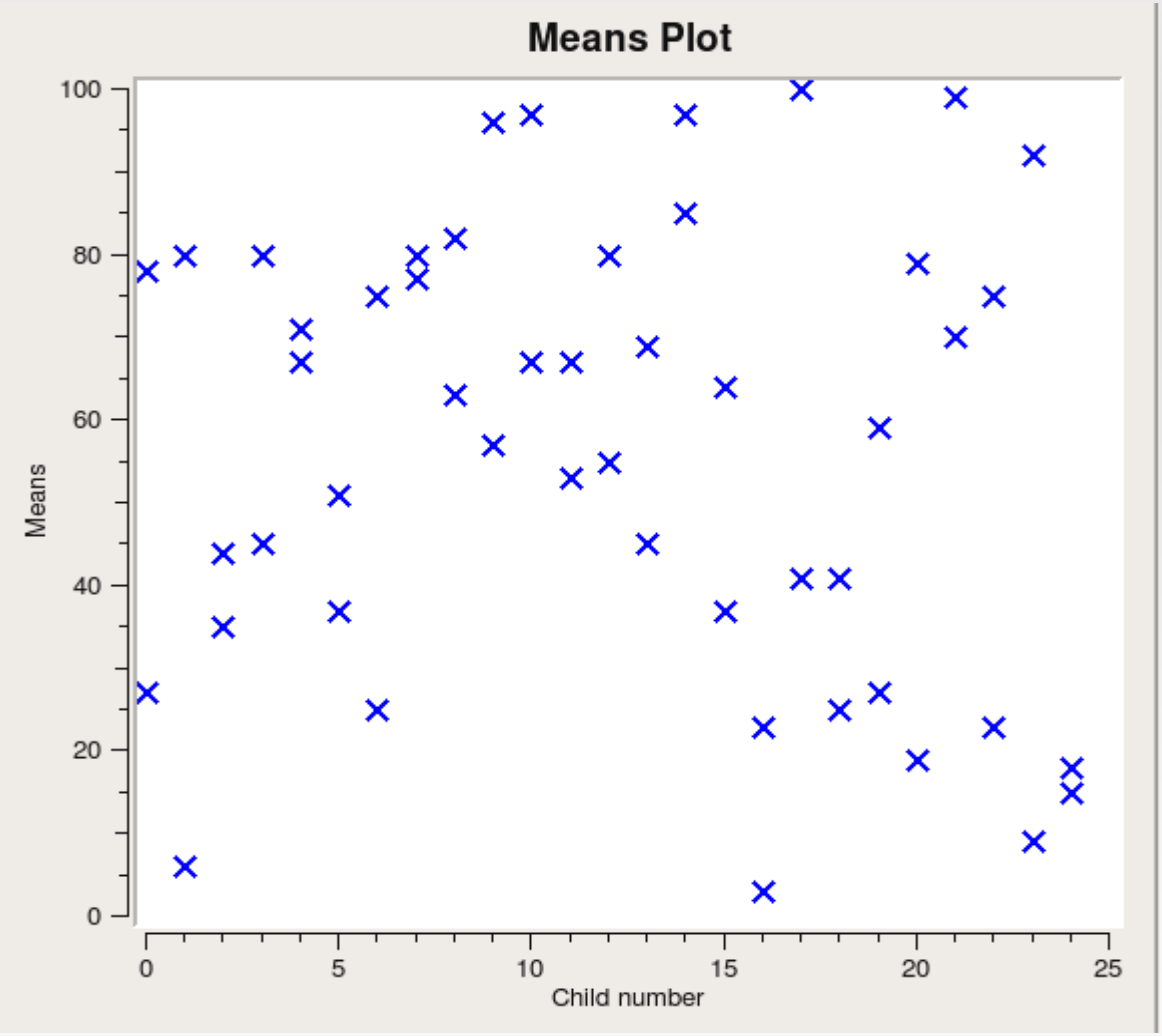
- Next up we can define the styles which will be applied to the curve (lines style, symbols styles, colours etc..). Here we will just to a scatter plot

```
curve = CurveProperties(curve_style = CurveStyle['none'],
        symbol = create_symbol(symbol = Symbols['xCross']))
```

- The penultimate step involves defining plot properties, such as plot/axis titles

```
plot = PlotProperties(axis = [ create_axis(
    axis_id =  AxisId['xBottom'], title = 'Child number'),
   create_axis(axis_id = AxisId['yLeft'], title = 'Means') ],
   title = ‘Scatter Means Plot')
```

Creating Your First PyPlotter [5/5]



And finally, we call the attach_result method which will append the required information to the result object

```
return self.attach_pyresult(result, rv, y_axis, plot = plot,                curve = cur
```

Using the PyPlotter [1/2]

- Now that we can create infinitely complex plotters, we need to actually use them
- All nodes that need to be plotted must be attached to the plotter as its children.
- As a demonstration of how this can be done, we will edit the example_sim.py script in Siamese within the Cattery
- The plotter will take the form of an inspector, which checks the status of its children for each request

Using the PyPlotter [2/2]

- To attach the plotter to the script, we need to:
  - Get hold of the nodes will become the plotter's children. This can be done with the nodesearch facility

```
nodes = ns.Search(name='\(uvw.%d*\)', class_name='MeqSpigot')
```

  - Add PyPlotter as a new root node, providing it with the nodes and specifying which plotter class to use

```
ns.pynode << Meq.PyNode(children = nodes,
  class_name="Timba.Contrib.AxM.pyvis.PyPlottableExamples.PyScatter
  Plotter")
```

  - It is a good idea to add a bookmark for the plotter

You Are Now Plotting Experts

- You can now apply these new skills to plot various combination of nodes which wasn't possible before
- The best plots by the end of the day will be **generously awarded**

Ready, Steady...

GO

(faster)