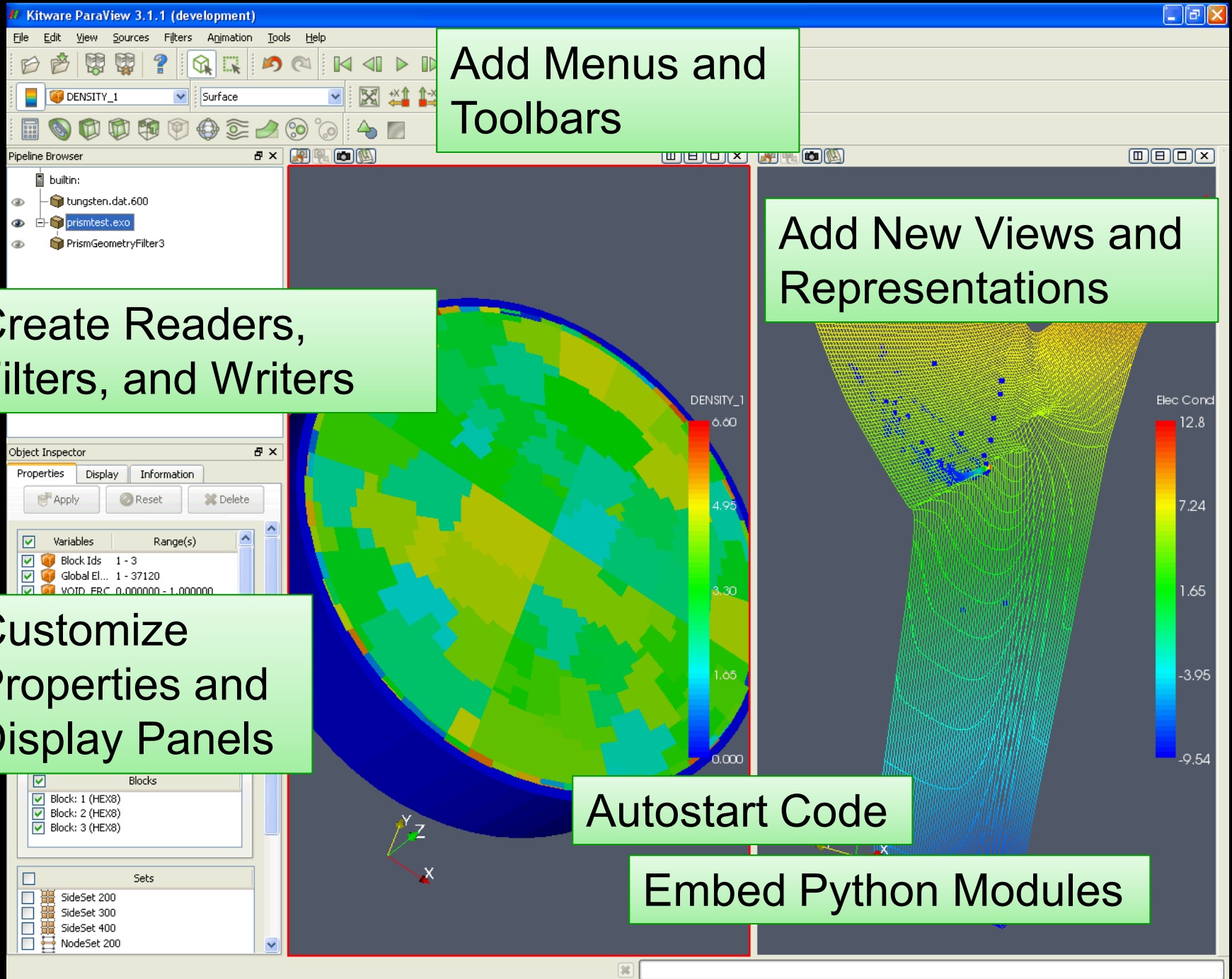


Customizing ParaView with Plugins

IEEE Vis ParaView Tutorial

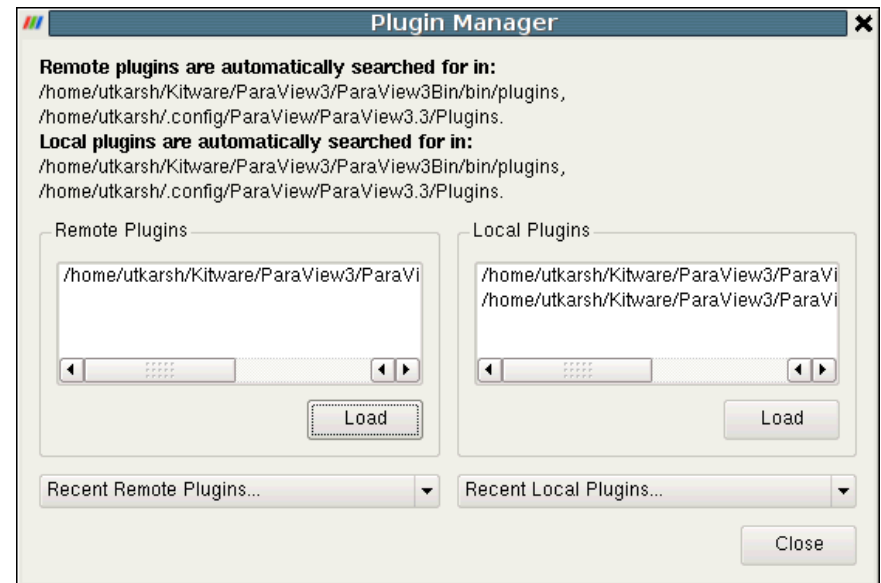
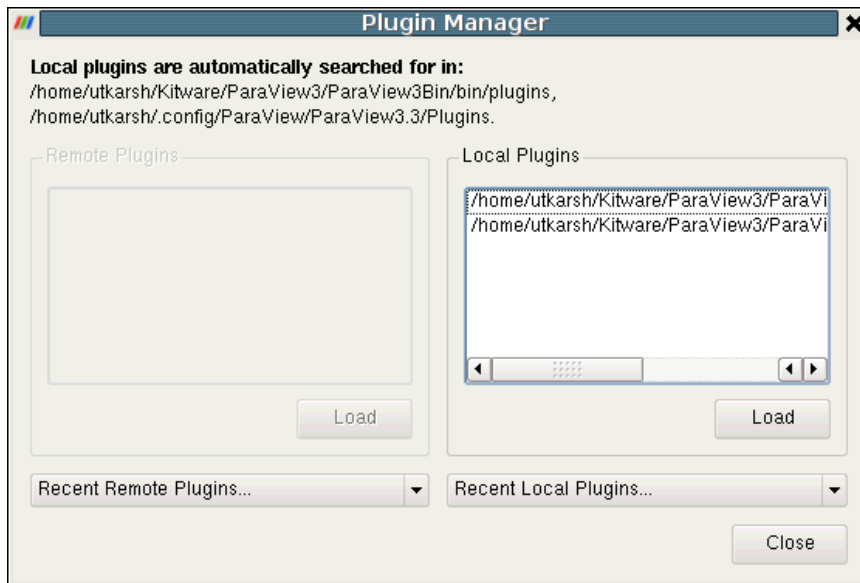
October 2009

**Kenneth Moreland
Sandia National Laboratories**



Loading Plugins

- **GUI Plugin Manager (Tools→Manage Plugins/Extensions).**



- **PV_PLUGIN_PATH** environment variable.
- **Recognized locations (see top of Plugin Manager).**

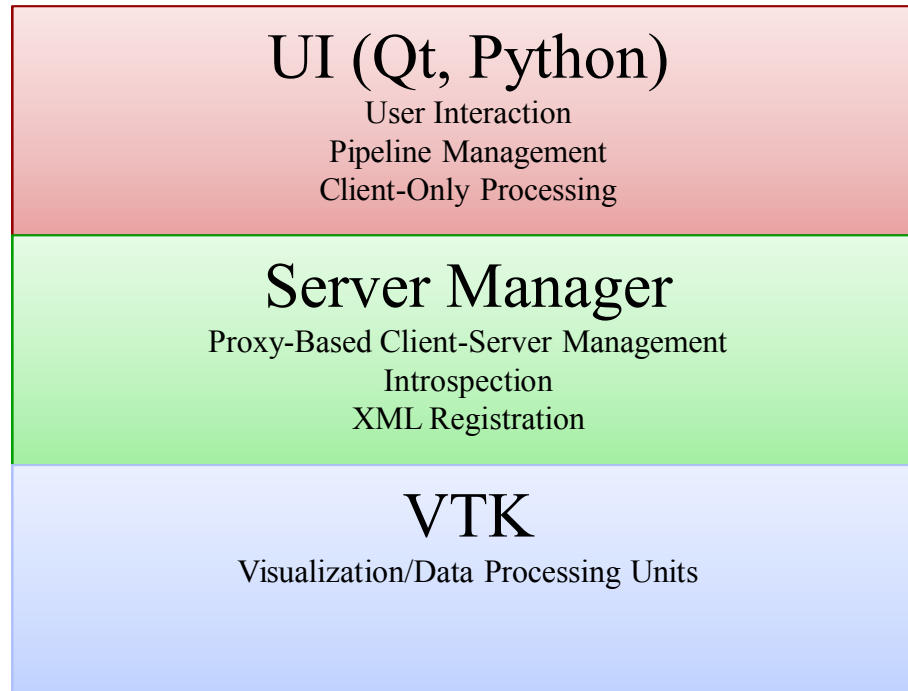


Where to Go for Help

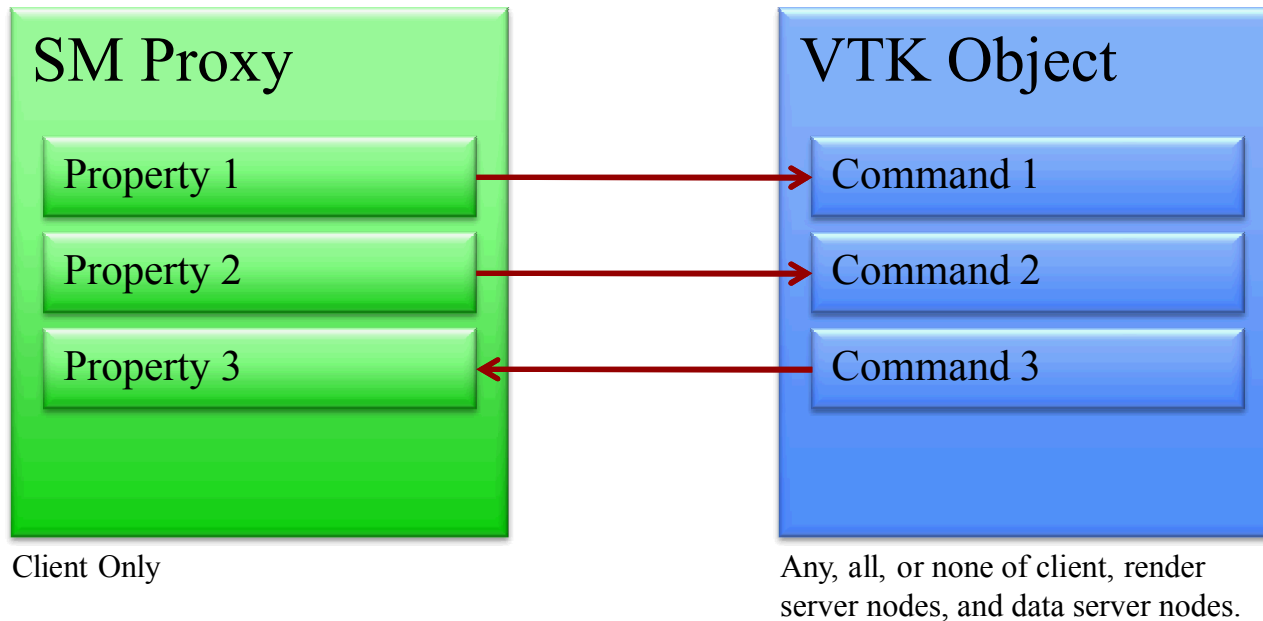
- Documentation on the Wiki
 - http://www.paraview.org/Wiki/Plugin_HowTo
 - www.paraview.org → Help → Wiki → Plugins
- MIRARCO plugin wizard
 - <http://pluginwizard.mirarco.org/>
- Examples
 - ParaView3/Examples/Plugins
 - ParaView3/Plugins



ParaView Application Layers



Server Manager Proxy Objects





Root Tag

Named Group

Filter Proxy Definition

Property tags inside
proxy tag.

```
<ServerManagerConfiguration>
  <ProxyGroup name="filters">
    <SourceProxy name="OBBDicer" class="vtkOBBDicer" label="OBB Dicer">
      <Documentation
        short_help="Break dataset into pieces."
        long_help="Define pieces for a dataset and annotate in a field.">
        This filter uses a tree of oriented bounding boxes to partition the
        space in which a data set is defined. A field is then added to the
        dataset that defines which partition each point is contained in.
      </Documentation>

      <InputProperty name="Input" ...

      <IntVectorProperty name="DiceMode" ...

      <IntVectorProperty name="NumberOfPointsPerPiece" ...

      <IntVectorProperty name="NumberOfPieces" ...

      <IntVectorProperty name="MemoryLimit" ...

    </SourceProxy> <!-- OBBDicer -->
  </ProxyGroup>
</ServerManagerConfiguration>
```



```
<ServerManagerConfiguration>  
  <ProxyGroup name="filters">
```

```
    <SourceProxy name="OBBDicer" ...
```

vtkAlgorithmMethod

Groups from
which inputs
can come.

```
      <InputProperty name="Input" command="SetInputConnection">  
        <ProxyGroupDomain name="groups">  
          <Group name="sources" />  
          <Group name="filters" />  
        </ProxyGroupDomain>  
        <DataTypeDomain name="input_type">  
          <DataType value="vtkDataSet" />  
        </DataTypeDomain>  
      </InputProperty>
```

Data Types the
Input can have.

```
    <IntVectorProperty name="DiceMode" ...
```

```
    <IntVectorProperty name="NumberOfPointsPerPiece" ...
```

```
    <IntVectorProperty name="NumberOfPieces" ...
```

```
    <IntVectorProperty name="MemoryLimit" ...
```




```
<ServerManagerConfiguration>  
  <ProxyGroup name="filters">
```

```
    <SourceProxy name="OBBDicer" ...
```

```
      <InputProperty name="Input" ...
```

Numerical Properties are
Vectors (scalars are size 1)

```
    <IntVectorProperty name="DiceMode" command="SetDiceMode"  
      number_of_elements="1"  
      default_values="0">
```

```
      <EnumerationDomain name="enum">
```

```
        <Entry value="0" text="Number of Points" />
```

```
        <Entry value="1" text="Specified Number" />
```

```
        <Entry value="2" text="Memory Limit" />
```

```
      </EnumerationDomain>
```

```
    <Documentation>
```

Specify the method to determine how many pieces the data should
be broken into.

```
    </Documentation>
```

```
  </IntVectorProperty>
```

```
  <IntVectorProperty name="NumberOfPointsPerPiece"  
    command="SetNumberOfPointsPerPiece"  
    number_of_elements="1"  
    default_values="5000">
```

```
    <IntRangeDomain name="range" min="1000" />
```

```
  <Documentation>
```

The maximum number of points to have in each piece. Only valid
if the mode is set to Number of Points.

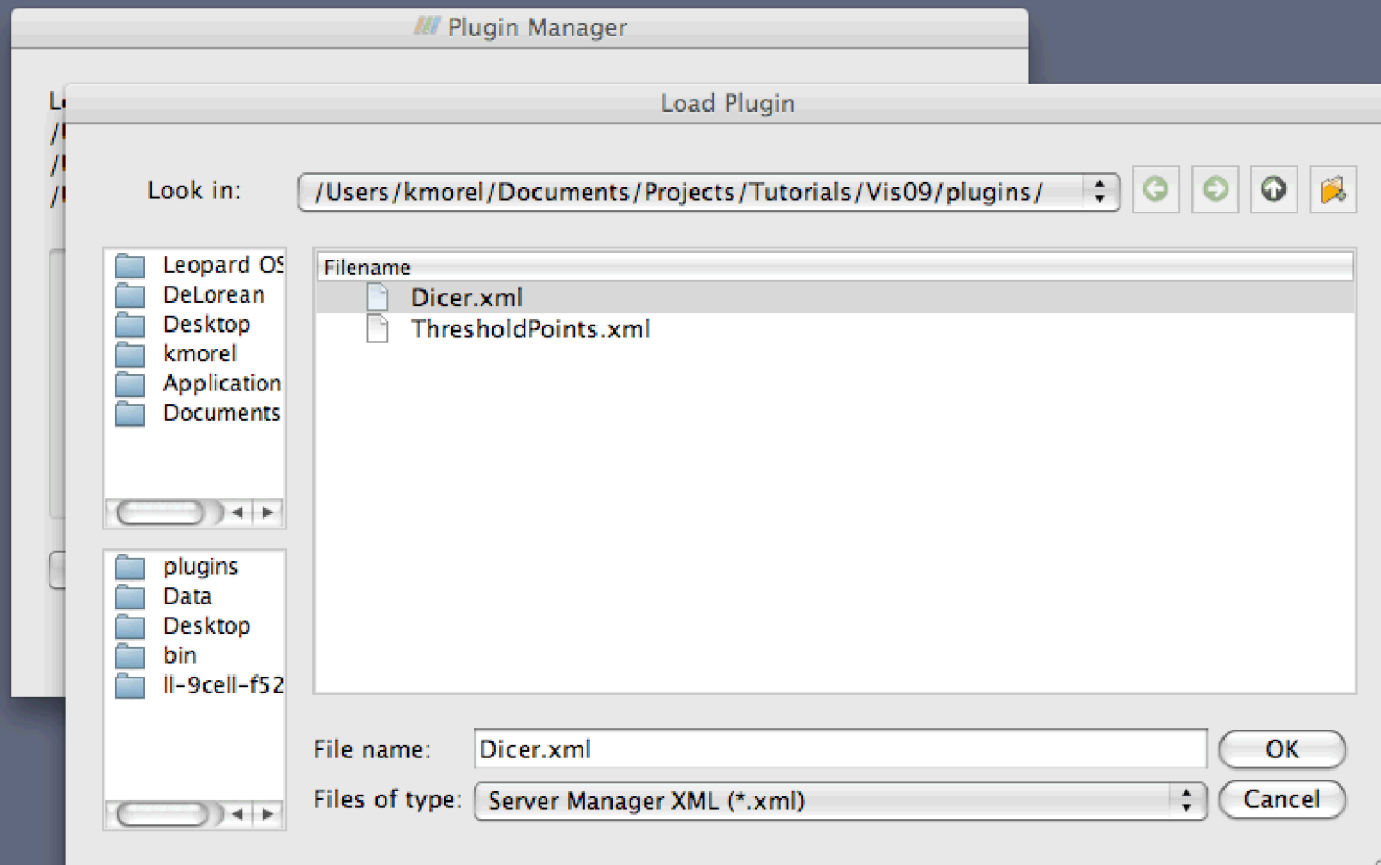
```
  </Documentation>
```

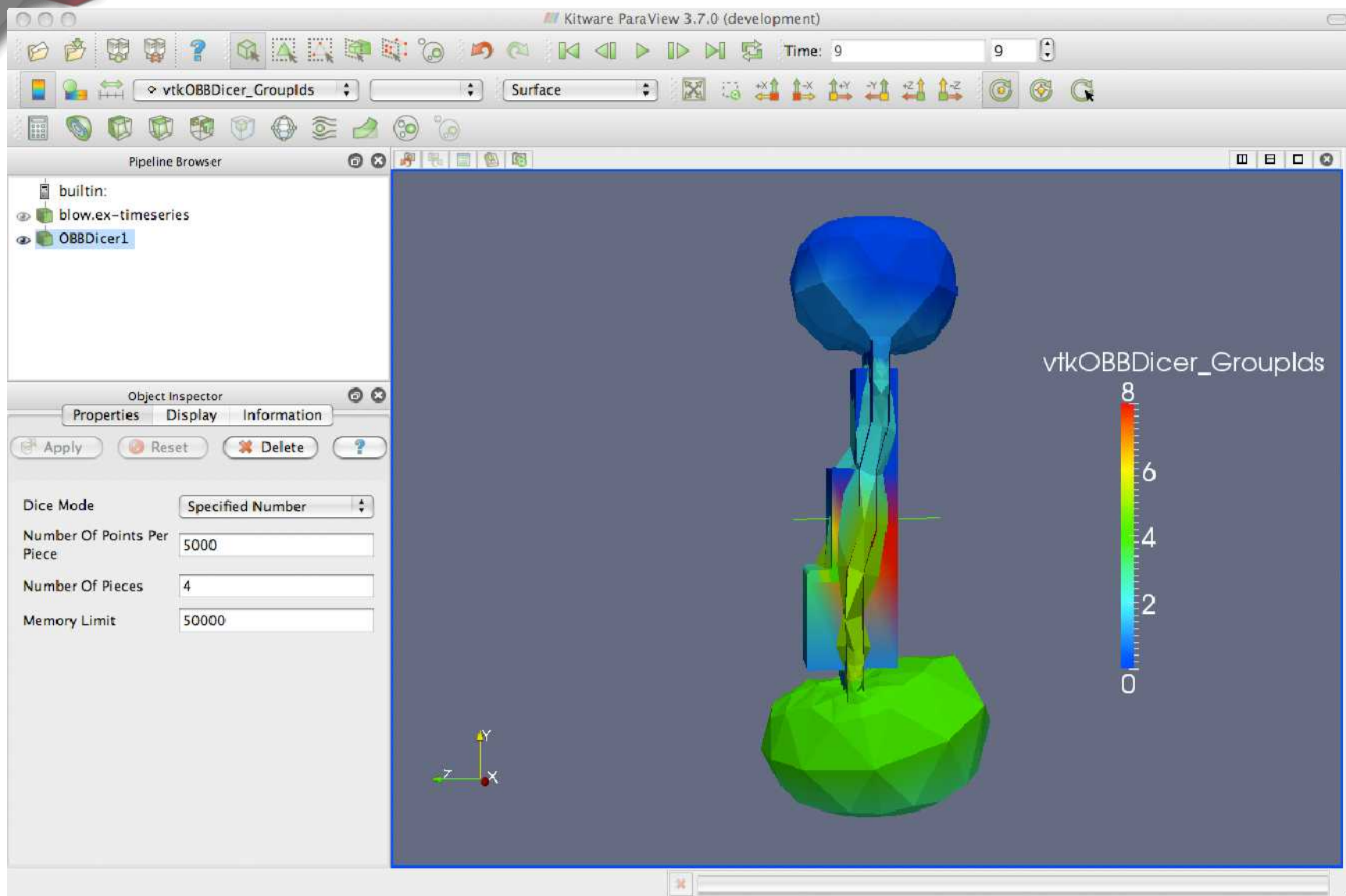
```
  </IntVectorProperty>
```

```
  <IntVectorProperty name="NumberOfPieces" ...
```

```
  <IntVectorProperty name="MemoryLimit" ...
```

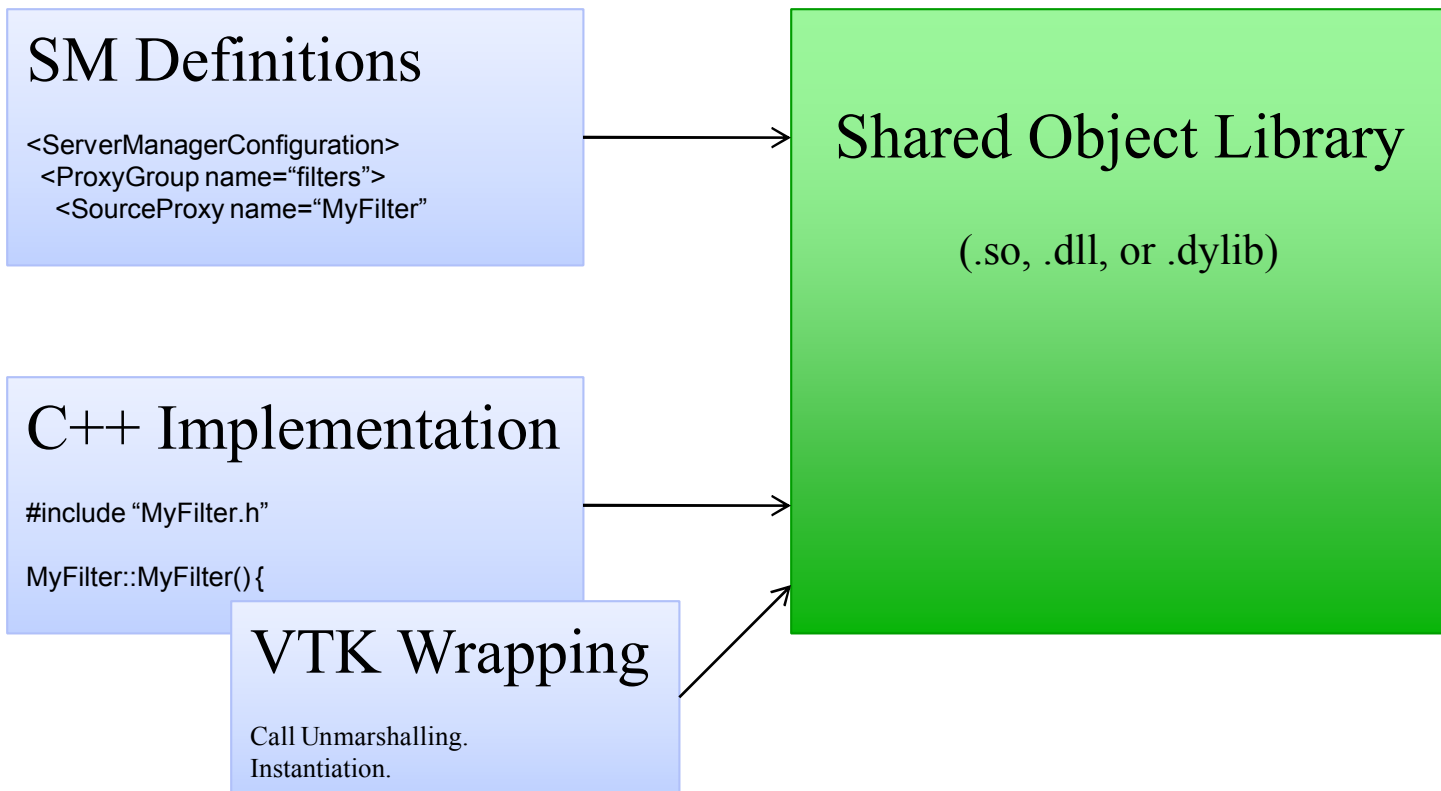
Domains limit values and
help GUI build widgets







Bundling SM XML with Implementation





Bundling Plugins with CMake

```
PROJECT(MyFilter)
```

```
FIND_PACKAGE(ParaView REQUIRED)
```

```
INCLUDE(${PARAVIEW_USE_FILE})
```

```
ADD_PARAVIEW_PLUGIN(MyFilter "1.0"  
  SERVER_MANAGER_XML MyFilter.xml  
  SERVER_MANAGER_SOURCES vtkMyFilter.cxx  
)
```

Using MIRARCO's Plugin Wizard

- <http://pluginwizard.mirarco.org/>



Create a vtkTransformImage Filter

ParaView Plugin Wizard

Select the desired type of plugin.

Please choose which kind of plugin you wish to develop from the list below:

- ☒ Filter
- ☐ Reader
- ☐ Source
- ☐ Writer



[Go Back](#) [Continue](#)

ParaView Plugin Wizard

VTK Filter Class: Basic information (Step 1/5)

Please fill in the pertinent information below.

Class name (must begin with 'vtk'):

vtkTransformImage

Header comment (/* */ delimiter will automatically be added):

Allows you to translate and scale an image by adjusting the origin and spacing.

List any header files that you would like imported (i.e. stdlib.h):

Supply the constructor's parameters (if any) here, comma-separated:

Number of Input Ports: 1

Number of Output Ports: 1

☒ Add destructor



[Go Back](#) [Continue](#)

ParaView Plugin Wizard

VTK Filter Class: Method override and parent class (Step 2/5)

Choose any methods to override, and define the parent class.

Select any methods that will be overridden:

- ☐ int FillInputPortInformation(...)
- ☐ vtkCompositeDataSet
- ☐ int FillOutputPortInformation(...)
- ☐ vtkCompositeDataSet
- ☒ int RequestData(...)
- ☐ int RequestInformation(...)
- ☒ void PrintSelf(...)

Select the parent class, or define a custom parent:

- ☐ vtkPolyDataAlgorithm
- ☐ vtkUnstructuredGridAlgorithm
- ☒ vtkImageAlgorithm
- ☐ vtkMultiBlockDataSetAlgorithm
- ☐ Custom parent class (the parent class name must begin with 'vtk')

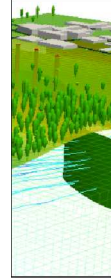
vtk

Input array details:

☐ Add Input Array

Type:

- ☒ Cell
- ☐ Point
- ☐ Scalars
- ☐ Optional



[Go Back](#) [Continue](#)



Implementation: Add Properties to SM XML

```
<SourceProxy name="TransformImage" class="vtkTransformImage">
```

```
  <InputProperty
    name="Input"
    command="SetInputConnection">
    <ProxyGroupDomain name="groups">
      <Group name="sources"/>
      <Group name="filters"/>
    </ProxyGroupDomain>
    <DataTypeDomain name="input_type">
      <DataType value="vtkImageData"/>
    </DataTypeDomain>
  </InputProperty>
```

```
    <DoubleVectorProperty name="Translate" command="SetTranslate"
      number_of_elements="3" default_values="0 0 0">
    </DoubleVectorProperty>
    <DoubleVectorProperty name="Scale" command="SetScale"
      number_of_elements="3" default_values="1 1 1">
    </DoubleVectorProperty>
```

```
</SourceProxy>
```




Implementation: Add Properties to Header

public:

```
static vtkTransformImage *New();  
vtkTypeRevisionMacro(vtkTransformImage, vtkImageAlgorithm);  
void PrintSelf(ostream& os, vtkIndent indent);
```

```
vtkGetVector3Macro(Translate, double);  
vtkSetVector3Macro(Translate, double);
```

```
vtkGetVector3Macro(Scale, double);  
vtkSetVector3Macro(Scale, double);
```

protected:

```
vtkTransformImage();  
~vtkTransformImage();
```

```
double Translate[3];  
double Scale[3];
```

```
int RequestData(vtkInformation *, vtkInformationVector **,  
                vtkInformationVector *);
```



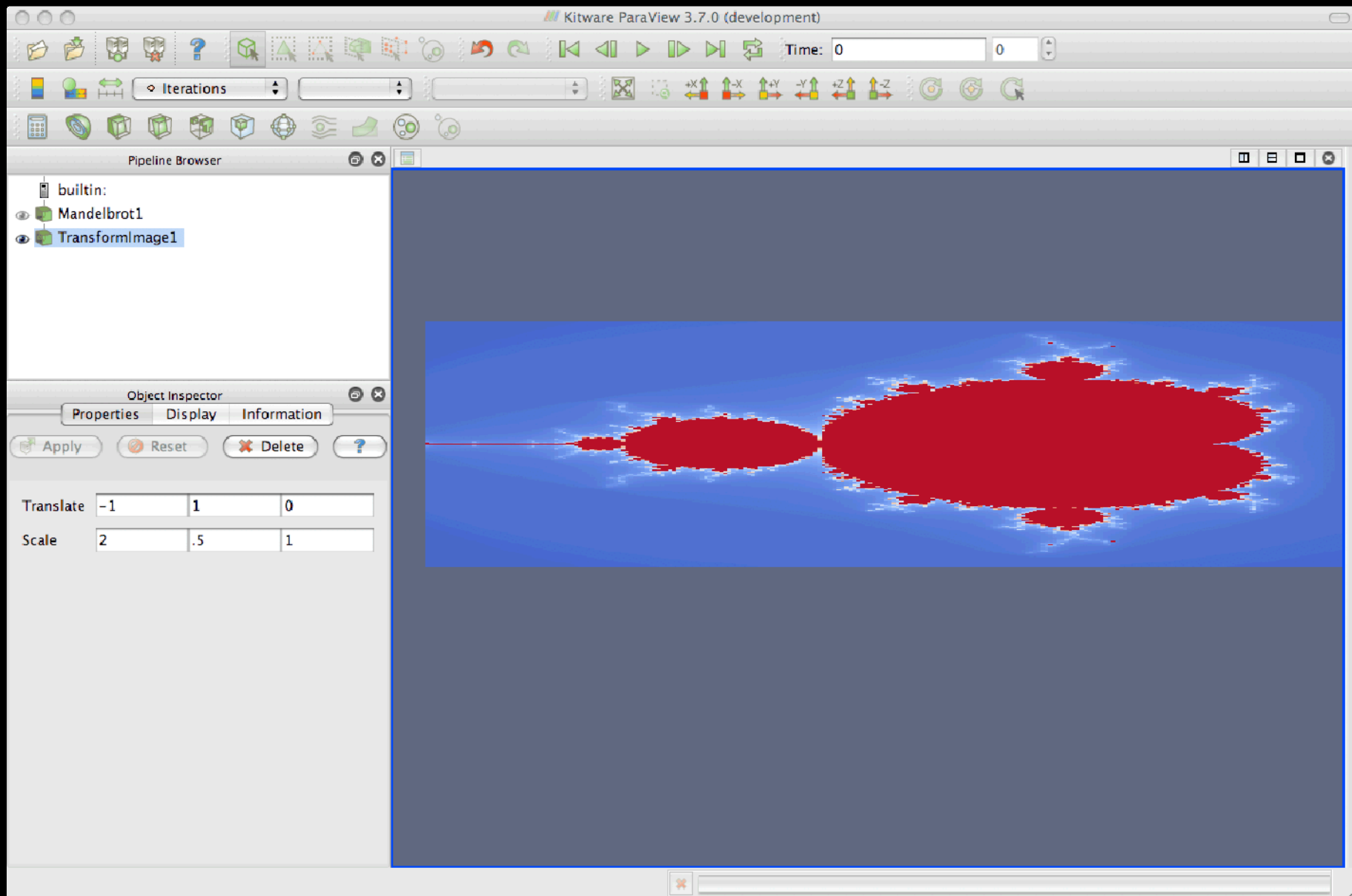
Implementation: Computation

```
int vtkTransformImage::RequestData(vtkInformation *request,
                                   vtkInformationVector **inputVector,
                                   vtkInformationVector *outputVector)
{
    vtkImageData *input = vtkImageData::GetData(inputVector[0]);
    vtkImageData *output = vtkImageData::GetData(outputVector);

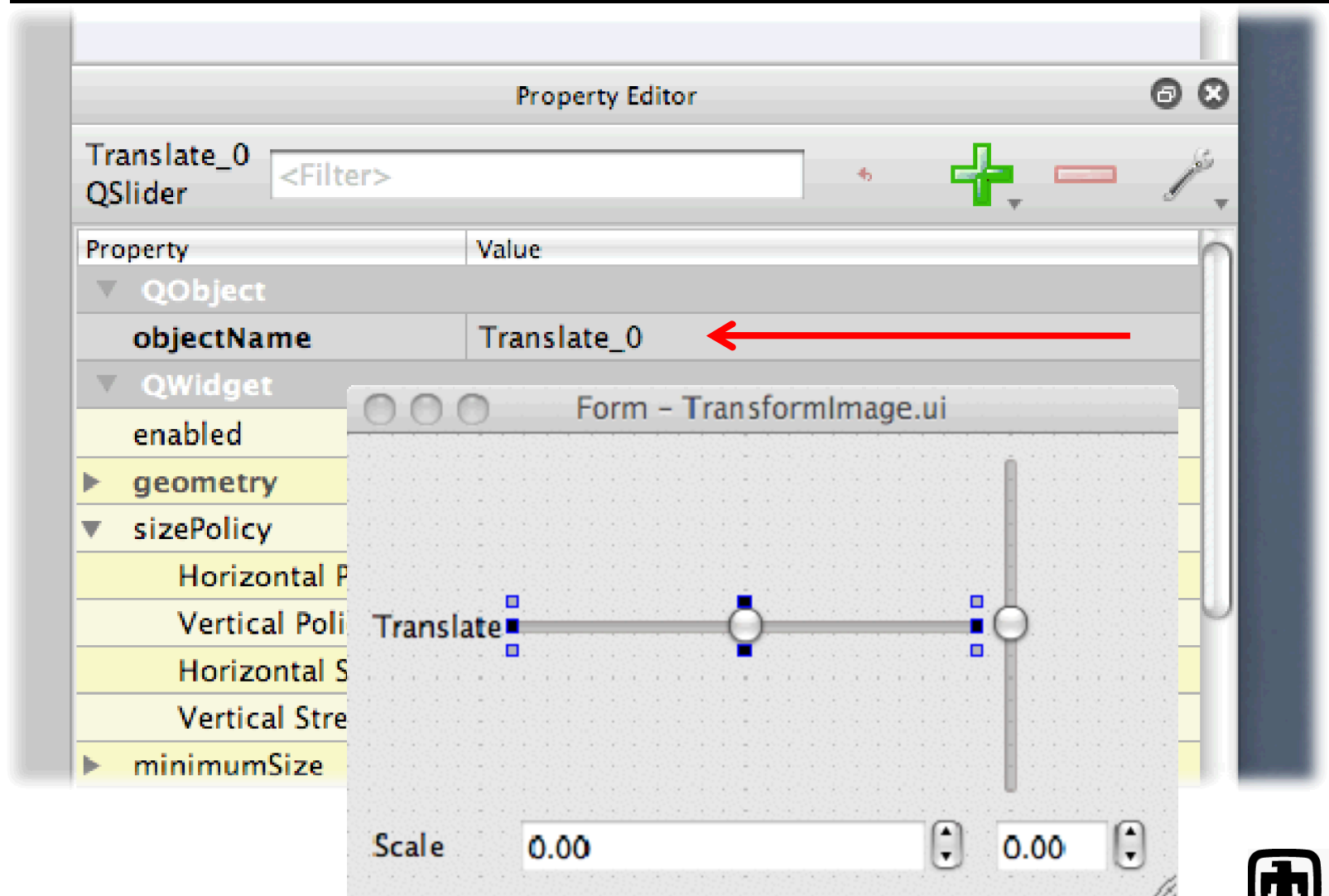
    output->CopyStructure(input);
    output->GetPointData()->PassData(input->GetPointData());
    output->GetCellData()->PassData(input->GetCellData());

    double origin[3], spacing[3];
    output->GetOrigin(origin);
    output->GetSpacing(spacing);
    for (int i = 0; i < 3; i++)
    {
        origin[i] += this->Translate[i];
        spacing[i] *= this->Scale[i];
    }
    output->SetOrigin(origin);
    output->SetSpacing(spacing);

    return 1;
}
```

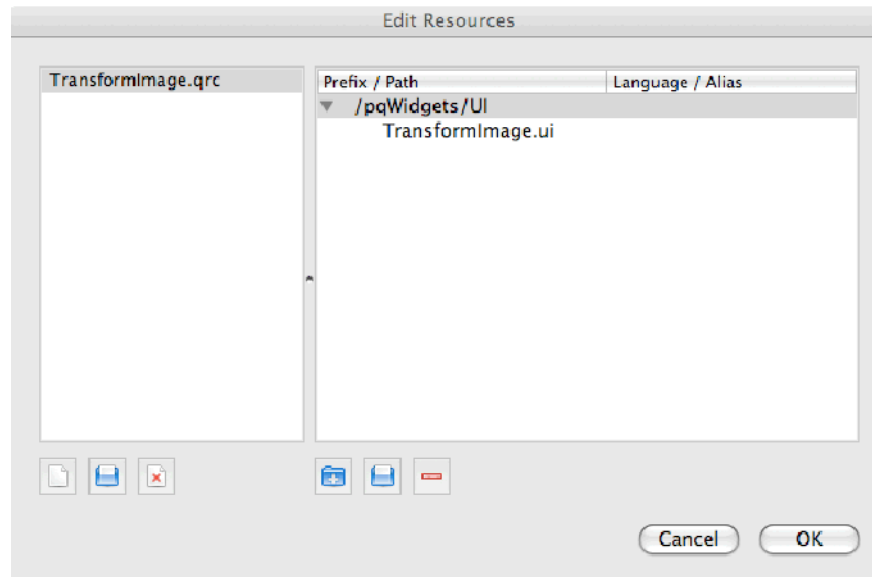


Creating a Custom Object Panel in Qt Designer



Linking the Custom Object Panel

- ParaView will look for a Qt resource file with path `:/pqWidgets/UI/proxyname.ui`



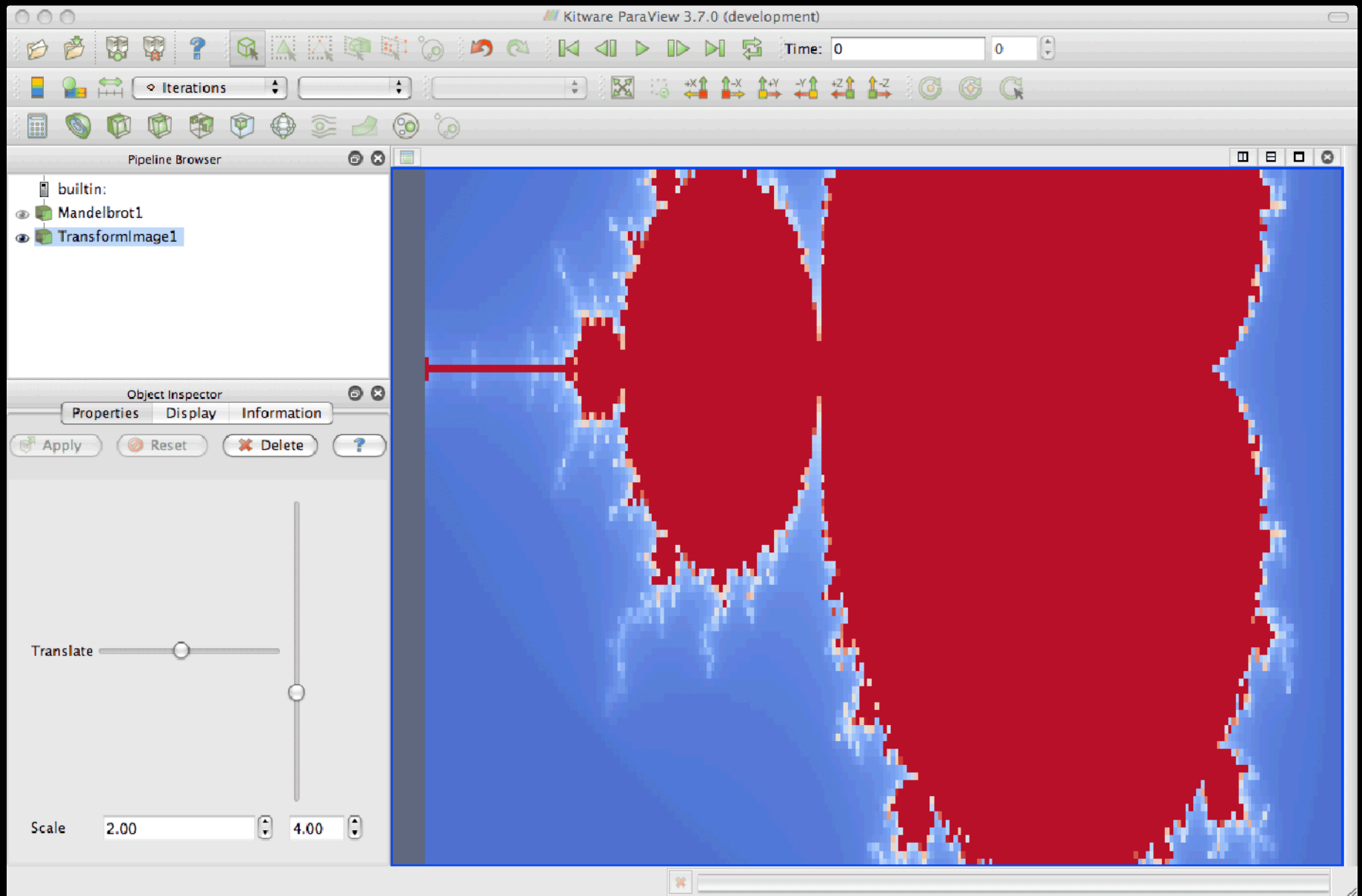
```
<RCC>
  <qresource prefix="/pqWidgets/UI" >
    <file>TransformImage.ui</file>
  </qresource>
</RCC>
```



Linking the Custom Object Panel

- Add the resource file to the **ADD_PARAVIEW_PLUGIN** command.

```
ADD_PARAVIEW_PLUGIN(TransformImageSMPlugin "1.0"  
    SERVER_MANAGER_XML TransformImage.xml  
    SERVER_MANAGER_SOURCES vtkTransformImage.cxx  
    GUI_RESOURCES TransformImage.qrc  
)
```





Panel Classes

pqAutoGeneratedObjectPanel Automatically creates widgets based on the properties defined in the server manager proxies. Used internally to create the automatically defined panels. Can be extended if you have small adjustments to make or small features to add.

pqNamedObjectPanel When your widgets have names corresponding to the server manager properties, this class can automatically hook them up. If your initial design comes principally from Qt Designer, this is a good class to extend.

pqObjectPanel Base class for all object panels. Provides little other than a blank widget container. You are responsible for managing Apply/Reset, SM state, and the undo stack.



```
#include "pqTransformImagePanel.h"
```

```
#include "ui_TransformImage.h"
```

```
class pqTransformImagePanel::pqUI : public Ui::TransformImage {};
```

```
pqTransformImagePanel::pqTransformImagePanel(  
    pqProxy *pxy, QWidget *p)
```

```
    : pqNamedObjectPanel(pxy, p)
```

```
{
```

```
    this->ui = new pqUI;
```

```
    this->ui->setupUi(this);
```

```
    this->linkServerManagerProperties();
```

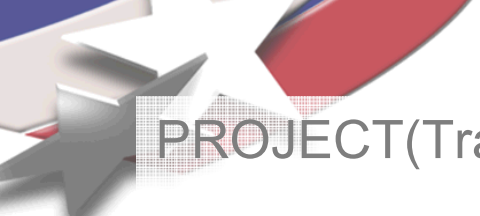
```
}
```

```
pqTransformImagePanel::~~pqTransformImagePanel()
```

```
{
```

```
    delete this->ui;
```

```
}
```



```
PROJECT(TransformImage)
```

```
FIND_PACKAGE(ParaView REQUIRED)  
INCLUDE(${PARAVIEW_USE_FILE})
```

```
QT4_WRAP_UI(UI_SRCS TransformImage.ui)  
QT4_WRAP_CPP(MOC_SRCS pqTransformImagePanel.h)
```

```
ADD_PARAVIEW_OBJECT_PANEL(IFACES IFACE_SRCS  
  CLASS_NAME pqTransformImagePanel  
  XML_NAME TransformImage XML_GROUP filters  
)
```

```
ADD_PARAVIEW_PLUGIN(TransformImageSMPlugin "1.0"  
  SERVER_MANAGER_XML TransformImage.xml  
  SERVER_MANAGER_SOURCES vtkTransformImage.cxx  
  GUI_INTERFACES ${IFACES}  
  GUI_SOURCES ${MOC_SRCS} ${UI_SRCS} ${IFACE_SRCS}  
    pqTransformImagePanel.cxx  
)
```



Autostart Plugins

- **ParaView can automatically launch code when your plugin starts up and when it shuts down.**
- **Create a QObject class with methods you want called on startup and shutdown.**
- **Use `ADD_PARAVIEW_AUTO_START` command to register startup and shutdown methods.**



```
#include <QObject>
```

```
class pqMyApplicationStarter : public QObject  
{  
    Q_OBJECT
```

```
public:
```

```
    pqMyApplicationStarter(QObject* p=0);  
    ~pqMyApplicationStarter();
```

```
    // Callback for startup.  
    void onStartup();
```

```
    // Callback for shutdown.  
    void onShutdown();
```



```
QT4_WRAP_CPP(MOC_SRCS pqMyApplicationStarter.h)
```

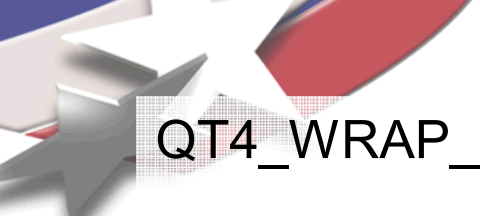
```
ADD_PARAVIEW_AUTO_START(IFACES IFACE_SRCS  
  CLASS_NAME pqMyApplicationStarter  
  STARTUP onStartup  
  SHUTDOWN onShutdown  
)
```

```
ADD_PARAVIEW_PLUGIN(Autostart "1.0"  
  GUI_INTERFACES ${IFACES}  
  GUI_SOURCES pqMyApplicationStarter.cxx  
  ${MOC_SRCS} ${IFACE_SRCS}  
)
```



Menu/Toolbar Plugins

- **Your plugin can create menus and toolbars and add actions.**
 - **Connect actions to Qt slots to perform arbitrary operations.**
- **Create a subclass of QActionGroup that fills itself with QActions in its constructor.**
 - **Hint: You can use Qt Designer to create a “dummy” widget with the QActions you want to return.**
Create all your QActions with a single setupUi call.
- **Use ADD_PARAVIEW_ACTION_GROUP command to register startup and shutdown methods.**



```
QT4_WRAP_CPP(MOC_SRCS MyActions.h)
```

```
ADD_PARAVIEW_ACTION_GROUP(TB_IFACE TB_IFACE_SRCS  
    CLASS_NAME MyActions  
    GROUP_NAME "ToolBar/MyActions"  
)
```

```
ADD_PARAVIEW_ACTION_GROUP(M_IFACE M_IFACE_SRCS  
    CLASS_NAME MyActions  
    GROUP_NAME "MenuBar/MyActions"  
)
```

```
ADD_PARAVIEW_PLUGIN(SourceToolbar "1.0"  
    GUI_INTERFACES ${TB_IFACES} ${M_IFACES}  
    GUI_SOURCES  
        ${MOC_SRCS}  
        ${TB_IFACE_SRCS} ${M_FACE_SRCS}  
        SourceToolbarActions.cxx  
)
```



What's Next

- **More plugin types exist. More on the way.**
 - Keep up to date with Wiki documentation.
- **Coming soon: Branding.**
 - The client applications will be replaced with a generic empty application.
 - Empty application configured to load a set of predefined plugins, called a brand.
 - In the future, you should be able to create new “vertical applications” by writing plugins.
 - Allows you to better share code between vertical applications (brands).