SAND2008-5204P


# Customizing ParaView

Timothy M. Shead
Sandia National Laboratories

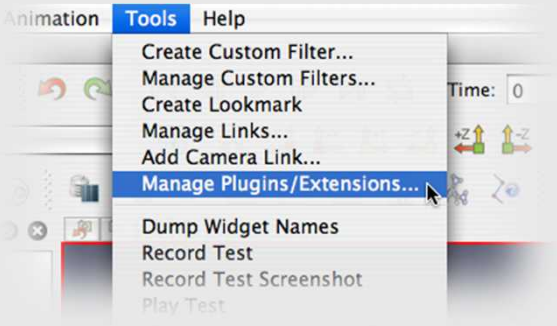
Sandia is a multiprogram laboratory operated by
Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy's
National Nuclear Security Administration
under contract DE-AC04-94AL85000.

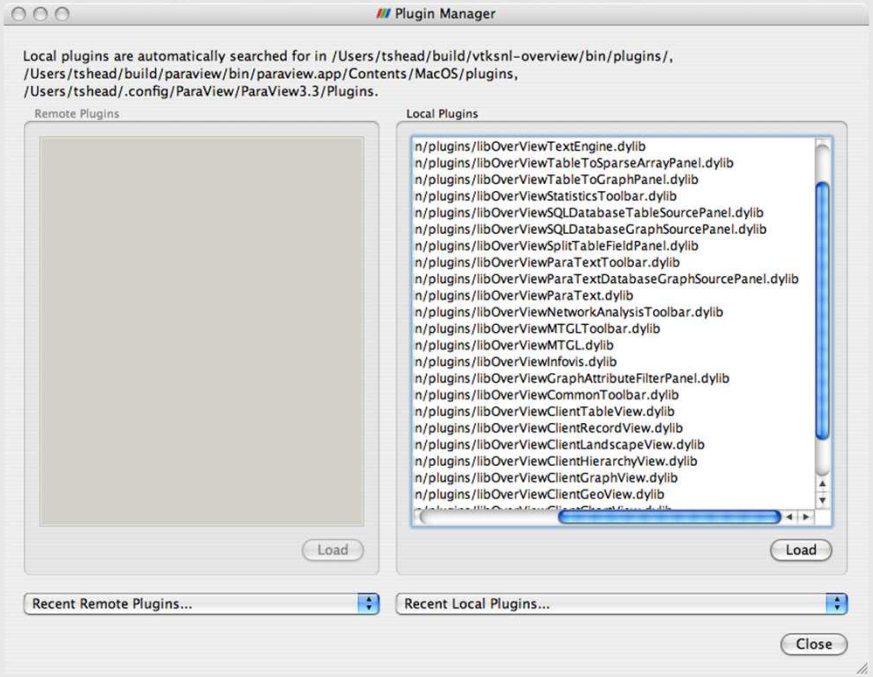

# Extending ParaView

- Traditional ParaView provides a static collection of filters and views:
  - Uses existing filters provided by VTK and ParaView libraries.
- The collection of filters can be extended at compile-time:
  - Sources outside the ParaView tree can be incorporated into the ParaView build.
  - Compiles and links external sources as part of the overall ParaView build.
  - Cons: setting-up the correct build environment for external sources can be tricky.
  - Cons: two-pass build process, dependencies
  - Reference: http://paraview.org/Wiki/ExtendingParaView

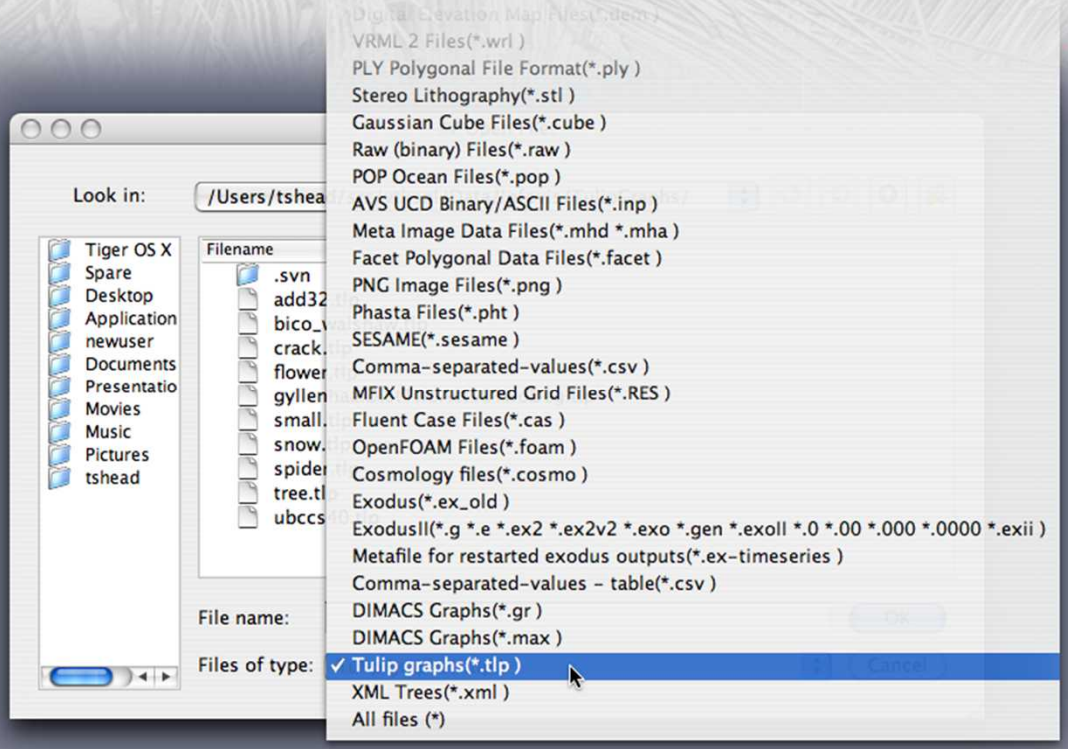
# ParaView Plugins

- Extend the collection of filters at runtime.
- Shared libraries containing new filters are dynamically-linked into the working-set at runtime.
- Plugins can be loaded automatically at startup from known locations, locations specified via environment variable (PV_PLUGIN_PATH) or manually loaded via the plugin manager GUI:

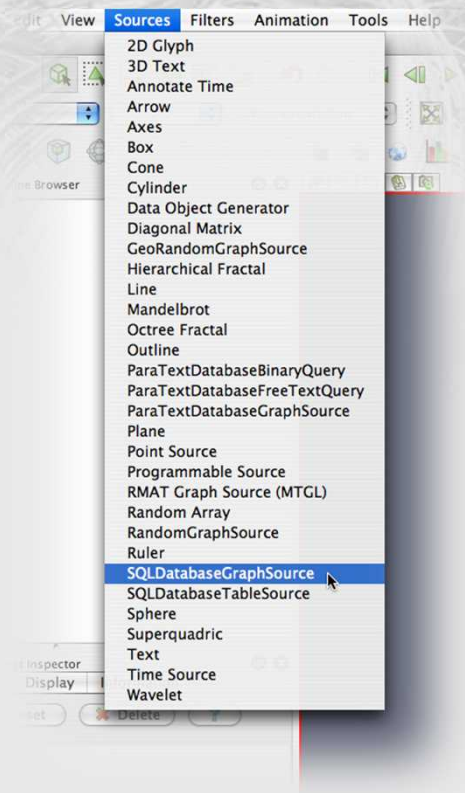


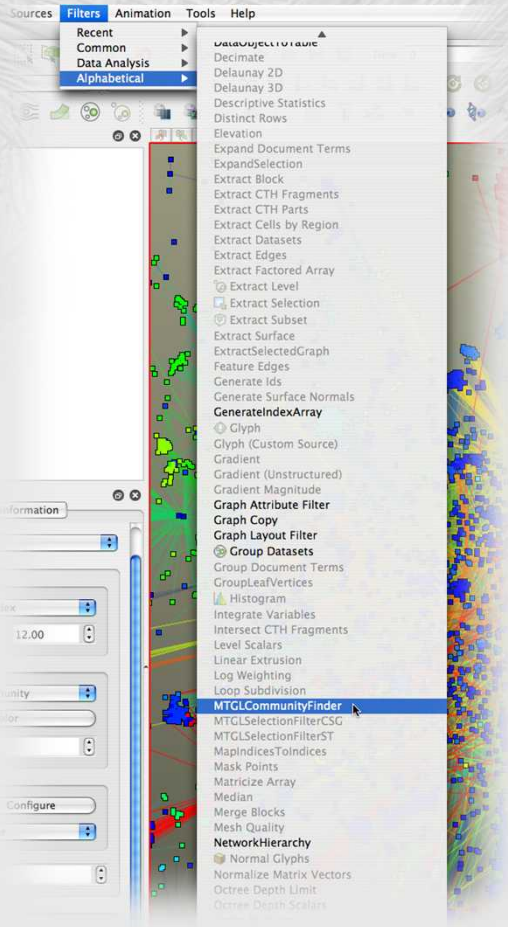



# Plugin Types - Readers & Writers



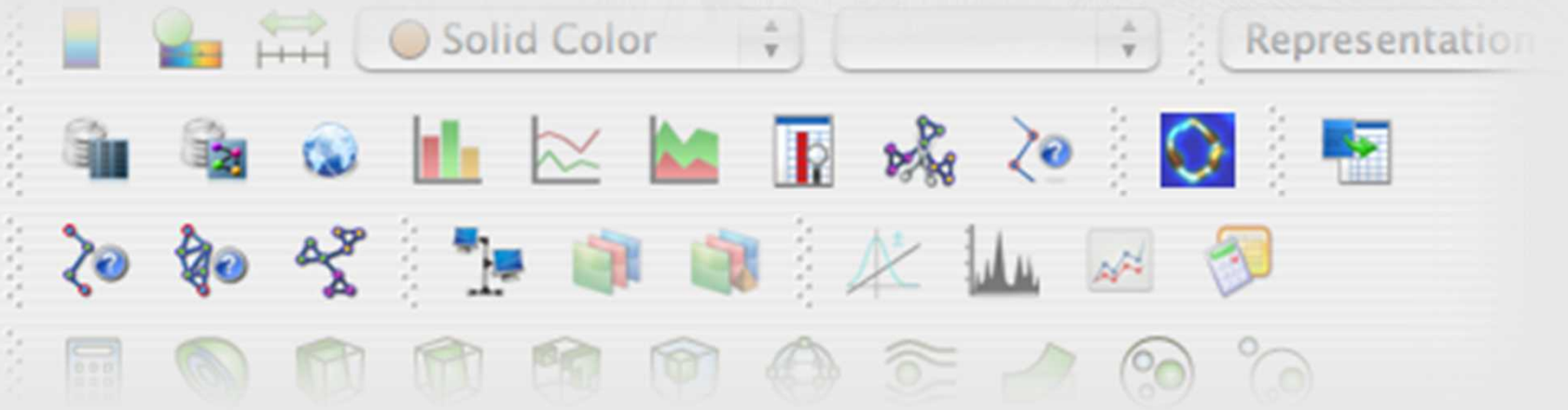
# Plugin Types - General Filters



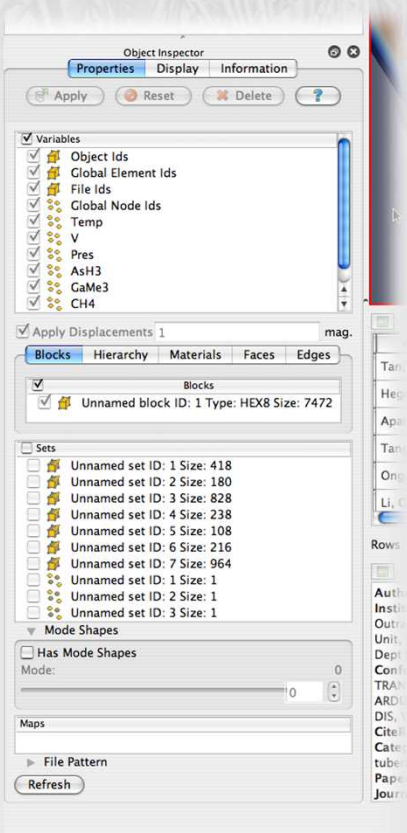
# Plugin Types - Custom Toolbars

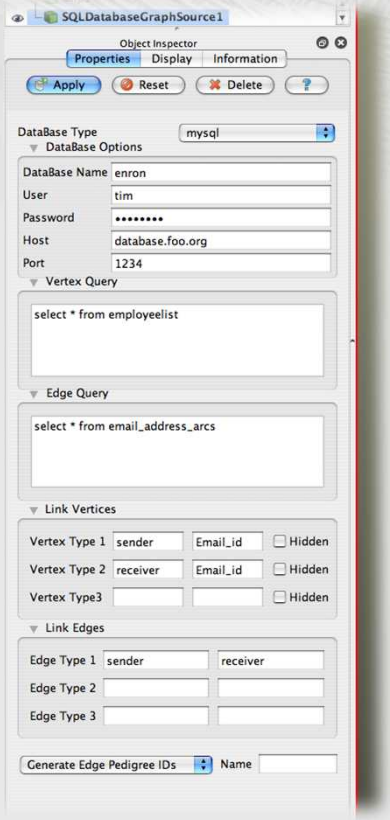
Useful for automating setup of a complex pipeline, an arbitrary view configuration, etc.



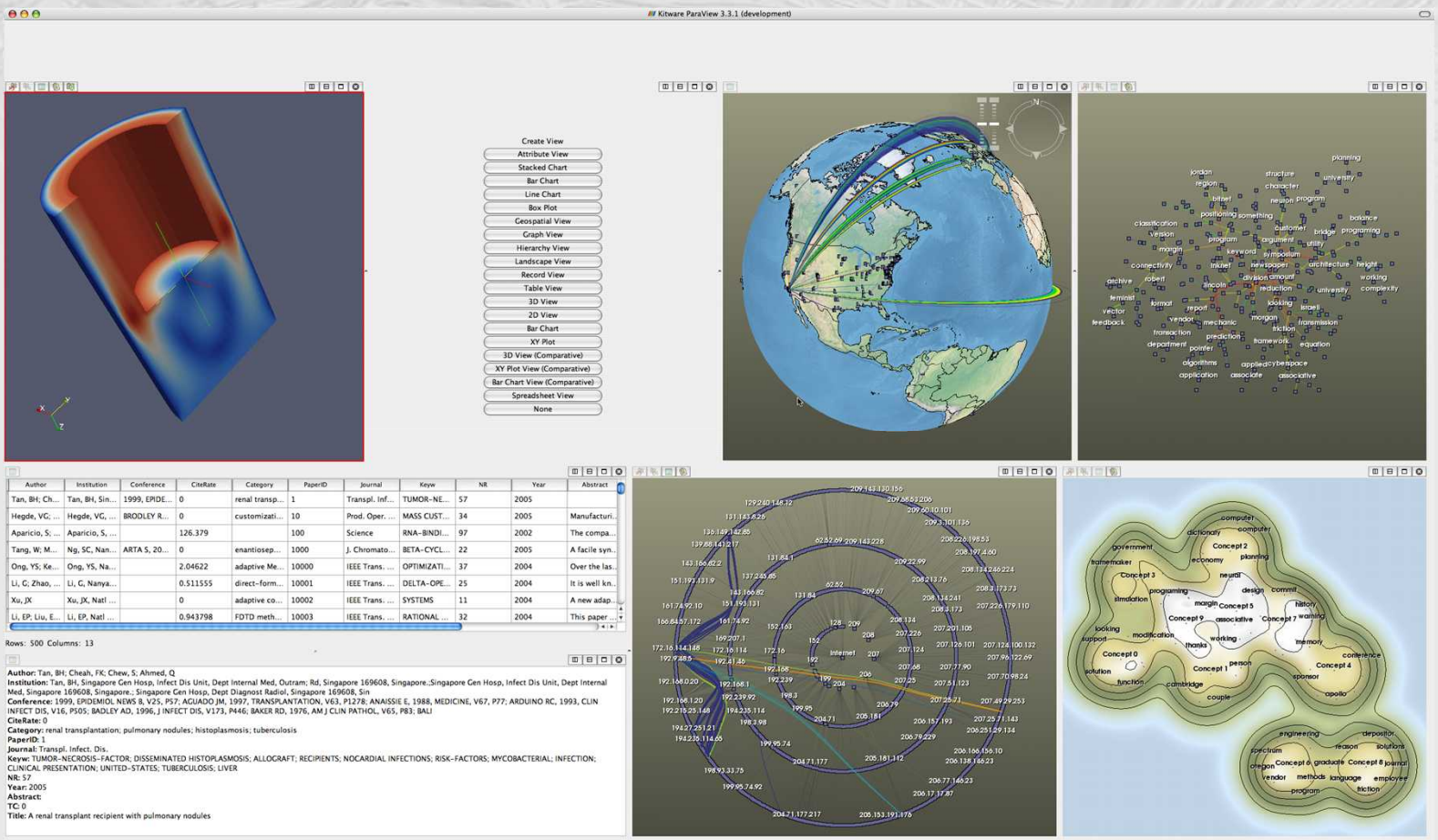
# Plugin Types - Custom Panels

Provides a proxy-specific user interface panel - useful with complex proxies where the auto-generated GUI is insufficient.



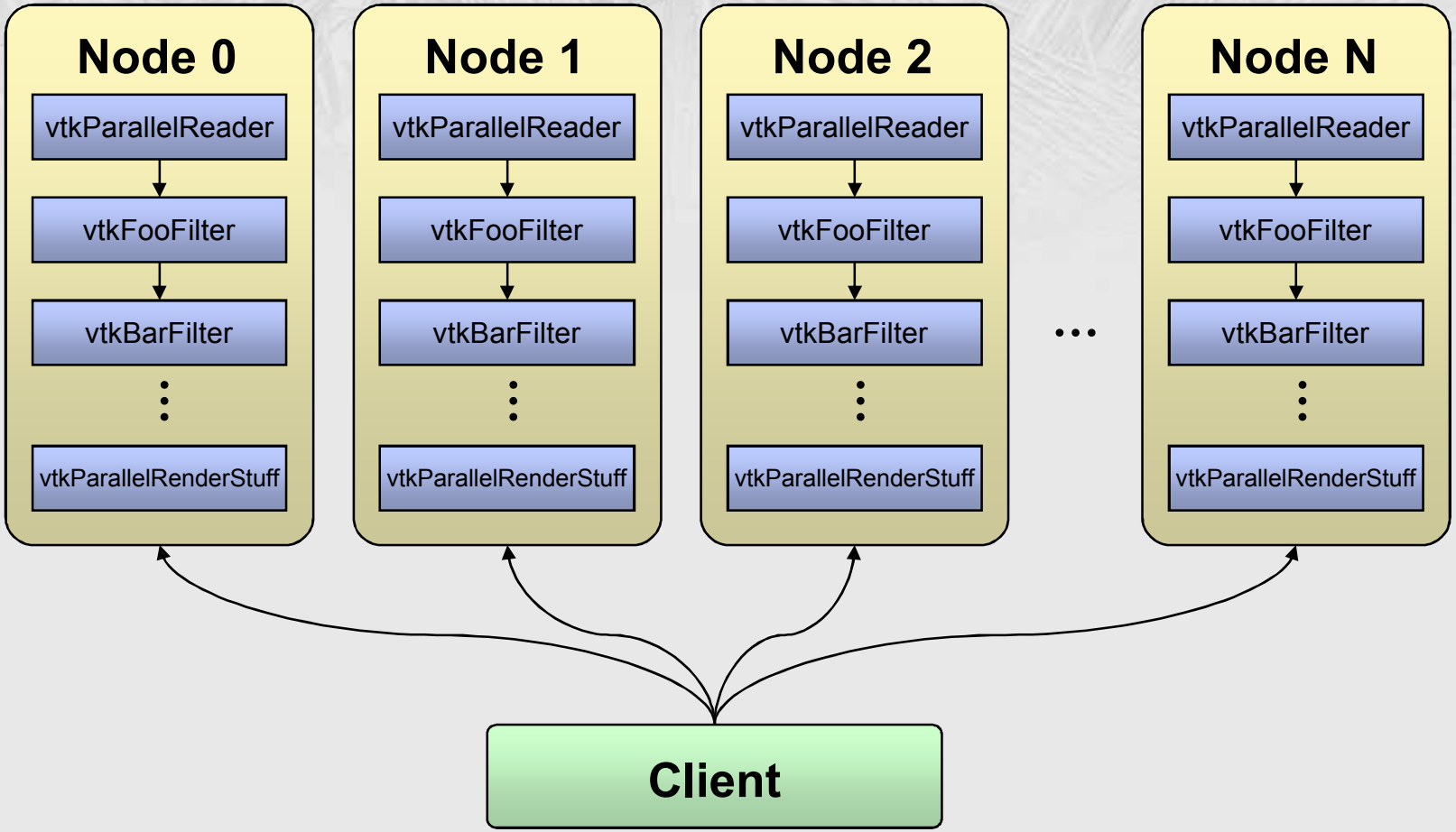
# Plugin Types - Custom Views



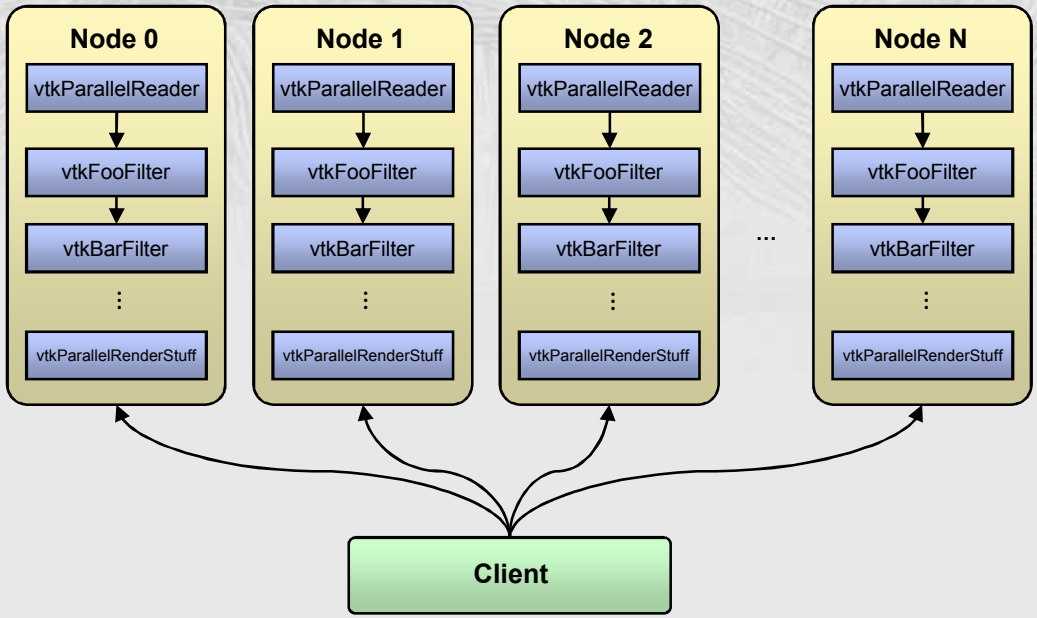
# Plugin Types - Miscellaneous

- Autostart Plugins - Executed automatically at program startup / shutdown.
  - Useful for logging, starting servers, etc.
- Graph Layout Algorithms
- Your Plugin Idea Here!

# An Everyday, Garden-Variety Parallel Pipeline …

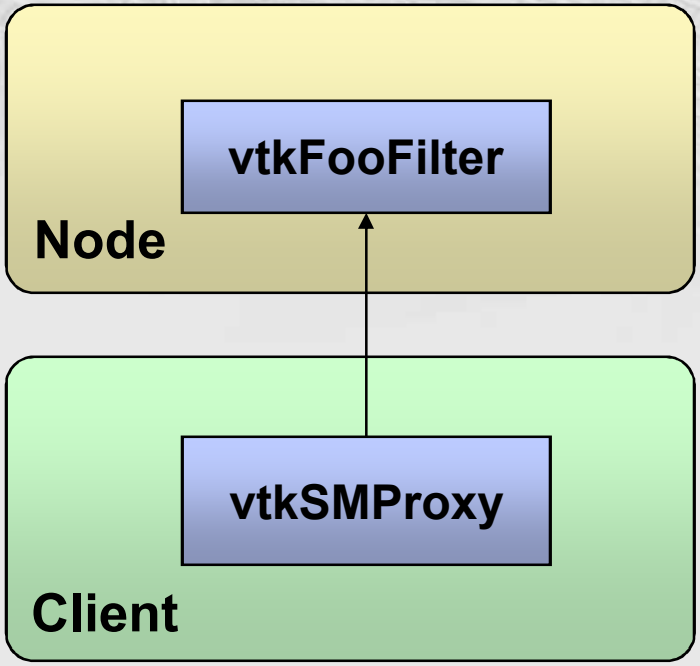


# Raises lots of practical questions …



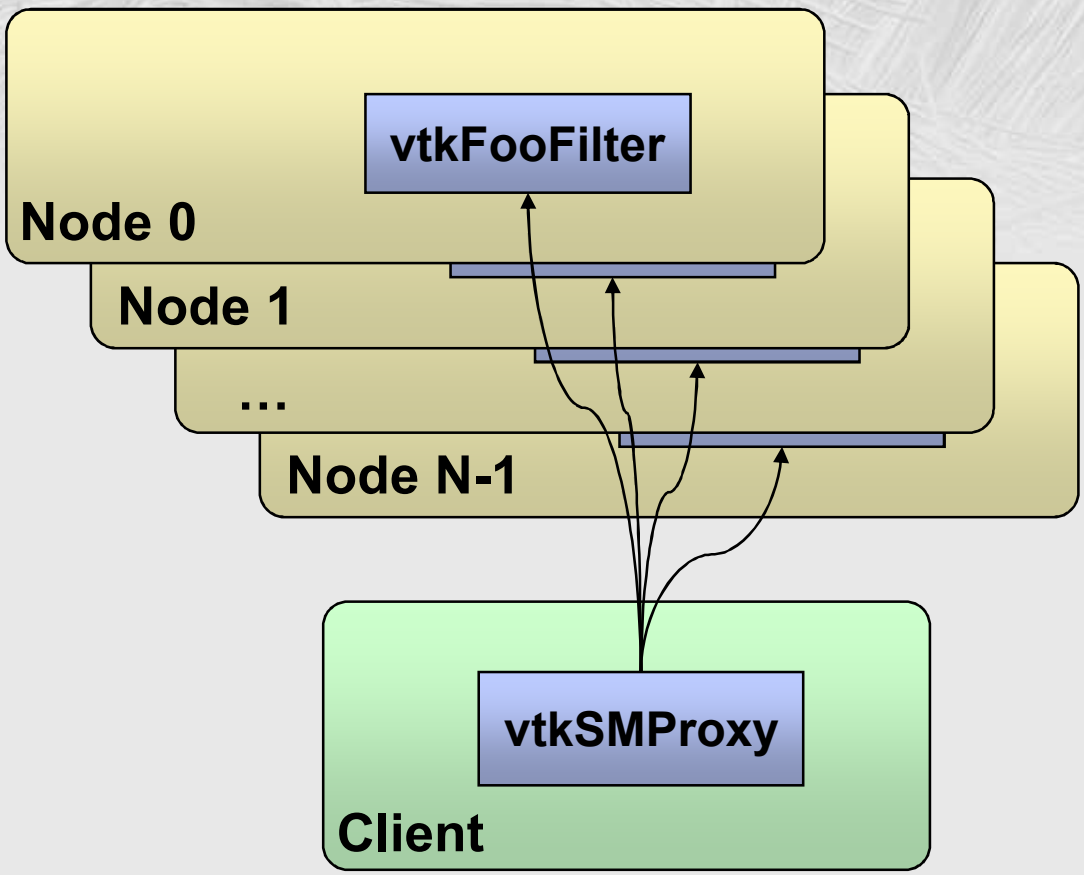

- How does the client refer to a remote VTK filter?
- How does the client create / delete VTK filters on a different host?
- How are remote VTK filters initialized / modified?
- How are remote VTK filters connected to form a pipeline?
- How does the client generate a UI for a filter?

# Referencing Remote VTK Filters




Proxies = Remote VTK Filters

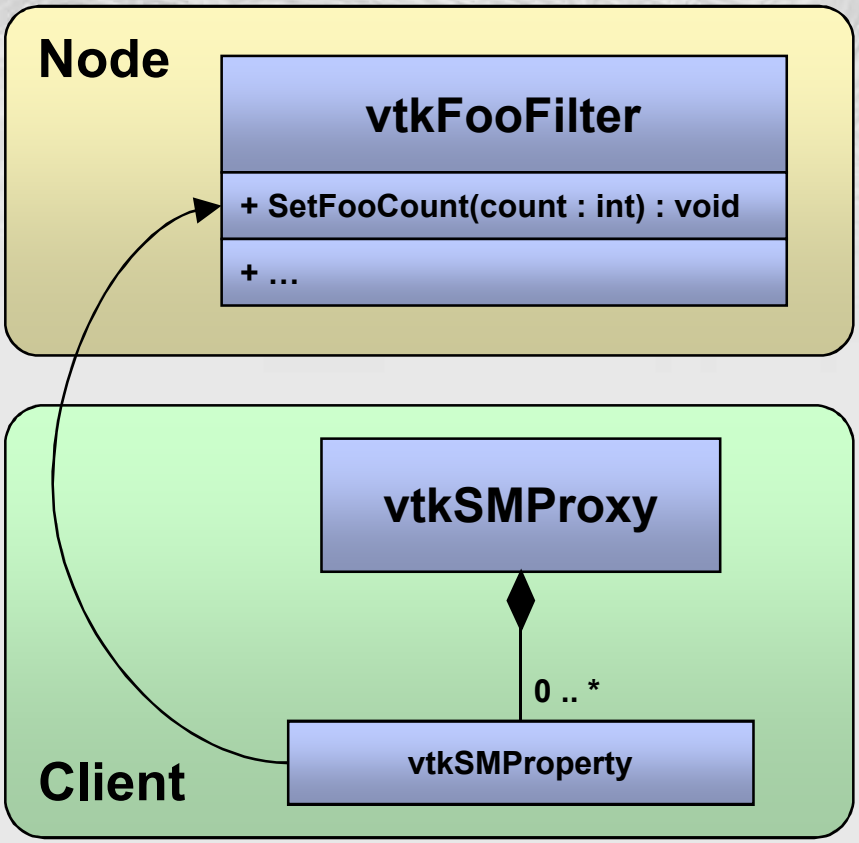
# Proxies and the Parallel Pipeline



# Proxy Naming

- Proxies are identified using a two-level hierarchy: "group" and "name".
- Proxy Groups identify broad categories of proxy.
  - Some current examples: "sources", "filters", "views", "representations", "writers", "lights", "textures".
  - You can create your own, e.g: "layout_strategies".
  - But many groups have special meaning to the user interface!
- Proxy Names identify specific proxy types.
  - Typically "vtkFooFilter" will have a proxy named "FooFilter".
  - This naming scheme isn’t enforced.

# Initializing / Modifying Proxies




Properties = Remote Method Calls

# Property Types

| Type | C++ Class | Representative Method Calls |
|---|---|---|
| Integers | vtkSMIntVectorProperty | SetFoo(int)<br>SetFoo(int[2]) |
| Doubles | vtkSMDoubleVectorProperty | SetFoo(double)<br>SetFoo(double[3]) |
| Strings | vtkSMStringVectorProperty | SetFoo(const char*)<br>SetFoo(const char*, const char*) |
| Proxy | vtkSMProxyProperty | SetFoo(vtkObject*) |
| Filter Input | vtkSMInputProperty | AddInput(…) |

# Property Domains

- Domains restrict the set of values that a property can assume.
  - Example: a domain can limit an integer property to a range of values.
  - Example: a domain can limit a string property to an enumerated list of values.
- Where a property defines the type of data, a domain can provide a higher-level description of how the data will be used.
  - Example: a domain can limit an integer property to boolean true/false values.
  - Example: a domain can specify that a string property will be used to represent filenames.

# But Where Do Proxies and Properties Come From?

- "Server Manager XML" defines each proxy and all its properties:

```
<ServerManagerConfiguration>
  <ProxyGroup name="filters">
    <SourceProxy name="FooFilter" class="vtkFooFilter">
      <InputProperty name="Input" … />
      <IntVectorProperty name="FooCount" … />
    </SourceProxy>
    <!-- More proxies in this group … -->
  </ProxyGroup>
  <!-- More groups in this file … -->
</ServerManagerConfiguration>
```

- The XML is linked into the binary (ParaView or plugin) as a static string, then parsed at runtime to populate a database of proxies.
- The XML provides a useful layer of indirection:
  - Proxies and properties can be named however you like, replacing the names of the underlying filters and methods.
  - You can provide properties for only those methods that you want to expose, simplifying the user interface.
  - You can provide your own preferred default values for properties, replacing those of the underlying filter.
  - One filter could be used in multiple proxies, "preconfigured" for multiple specific use-cases.

# Plugin Examples

- Reader
- Filter
- Toolbar
- Custom Panel

# Sample Reader Plugin XML

```
<ServerManagerConfiguration>
  <ProxyGroup name="sources">
    <SourceProxy name="TulipReader" class="vtkTulipReader">
      <StringVectorProperty name="FileName"
        command="SetFileName" number_of_elements="1">
        <FileListDomain name="files"/>
      </StringVectorProperty>
      <Hints>
        <View type="ClientGraphView"/>
      </Hints>
    </SourceProxy>
  </ProxyGroup>
</ServerManagerConfiguration>
```

# Sample Reader Client XML


```
<ParaViewReaders>
  <Reader name="TulipReader" extensions="tlp"
    file_description="Tulip graphs">
  </Reader>
</ParaViewReaders>
```

# Sample Reader Listfile

```
ADD_PARAVIEW_PLUGIN(MyReaders "1.0"
  SERVER_MANAGER_XML
    MyReadersSM.xml
  SERVER_MANAGER_SOURCES
    ${VTK_SOURCE_DIR}/Infovis/vtkTulipReader.h
  GUI_RESOURCE_FILES
    MyReadersGUI.xml
  )

TARGET_LINK_LIBRARIES(MyReaders
  vtkInfovis
  )
```

# Sample Filter Plugin XML

```
<ServerManagerConfiguration>
  <ProxyGroup name="filters">
    <SourceProxy name="DataObjectToTable" class="vtkDataObjectToTable">
      <InputProperty name="Input" command="SetInputConnection"/>
      <IntVectorProperty name="FieldType" command="SetFieldType"
        number_of_elements="1" default_values="0">
        <EnumerationDomain name="enum">
          <Entry value="0" text="Field Data"/>
          <Entry value="1" text="Point Data"/>
          <Entry value="2" text="Cell Data"/>
          <Entry value="3" text="Vertex Data"/>
          <Entry value="4" text="Edge Data"/>
        </EnumerationDomain>
      </IntVectorProperty>
      <Hints>
        <View type="ClientTableView"/>
      </Hints>
    </SourceProxy>
  </ProxyGroup>
</ServerManagerConfiguration>
```

# Sample Filter Listfile

```
ADD_PARAVIEW_PLUGIN(MyFilters "1.0"
  SERVER_MANAGER_XML
    MyFiltersSM.xml
  SERVER_MANAGER_SOURCES
    ${VTK_SOURCE_DIR}/Infovis/vtkDataObjectToTable.h
  )

TARGET_LINK_LIBRARIES(MyFilters
  vtkInfovis
  )
```

# Sample Toolbar Plugin Source

```
#include <QActionGroup>
#include <QApplication>
#include <QMessageBox>
#include <QStyle>

class MyToolBar : public QActionGroup
{
  Q_OBJECT
public:
  MyToolBar (QObject* p) :
    QActionGroup(p)
  {
    QIcon icon = qApp->style()->standardIcon(QStyle::SP_MessageBoxCritical);

    QAction* a = this->addAction(new QAction(icon, "MyAction", this));
    QObject::connect(a, SIGNAL(triggered(bool)), this, SLOT(onAction()));
  }

public slots:
  void onAction()
  {
    QMessageBox::information(NULL, "MyAction", "MyAction was invoked\n");
  }
};
```

# Sample Toolbar Listfile

```
QT4_WRAP_CPP(MOC_SRCS MyToolBar.h)

ADD_PARAVIEW_ACTION_GROUP(
  IFACES IFACE_SRCS
  CLASS_NAME MyToolBar
  GROUP_NAME "ToolBar/MyActions"
  )

ADD_PARAVIEW_PLUGIN(MyToolBar "1.0"
  GUI_INTERFACES ${IFACES}
  SOURCES MyToolBar.cxx ${MOC_SRCS} ${IFACE_SRCS}
  )
```

# Custom Panel Sources

```
#include "pqObjectPanel.h"

class MyCustomPanel :
  public pqObjectPanel
{
  Q_OBJECT

public:
  MyCustomPanel(pqProxy* proxy, QWidget* p) :
    pqObjectPanel(proxy, p)
  {
    vtkSMProxy* my_proxy = proxy->GetProxy();
    /* Sync widgets with proxy property values here. */
    /* Connect widget signals to the setModified() slot here … */
    QObject::connect(this->Widgets.foo, SIGNAL(textEdited(const QString&)), this, SLOT(setModified()));
  }

private slots:
  virtual void accept()
  {
    vtkSMProxy* my_proxy = this->referenceProxy()->getProxy();
    /* Sync proxy properties with widget states here. */
  }
  virtual void reset()
  {
    vtkSMProxy* my_proxy = this->referenceProxy()->getProxy();
    /* Sync widgets with proxy property values here. */
    pqObjectPanel::reset();
  }
};
```

# Custom Panel Listfile

```
QT4_WRAP_CPP(MOC_SRCS MyCustomPanel.h)

ADD_PARAVIEW_OBJECT_PANEL(
  IFACES IFACE_SRCS
  CLASS_NAME MyCustomPanel
  XML_NAME DataObjectToTable
  XML_GROUP filters
  )

ADD_OVERVIEW_PLUGIN(
  MyCustomPanel "1.0"
  GUI_INTERFACES ${IFACES}
  SOURCES MyCustomPanel.cxx ${MOC_SRCS} ${IFACE_SRCS}
  )
```

# Plugin Execution Context

- Remember that plugins can be loaded into the client, the server, or both.
- In many cases a server won't have access to the same libraries as the client.
  - Example: a plugin that contains a VTK filter and a corresponding custom panel might not load successfully on the server, since the custom panel has a Qt dependency.
  - Solution: separate plugins for server and client - one containing just the filter for use with server and client, and one containing just the panel for use with the client.

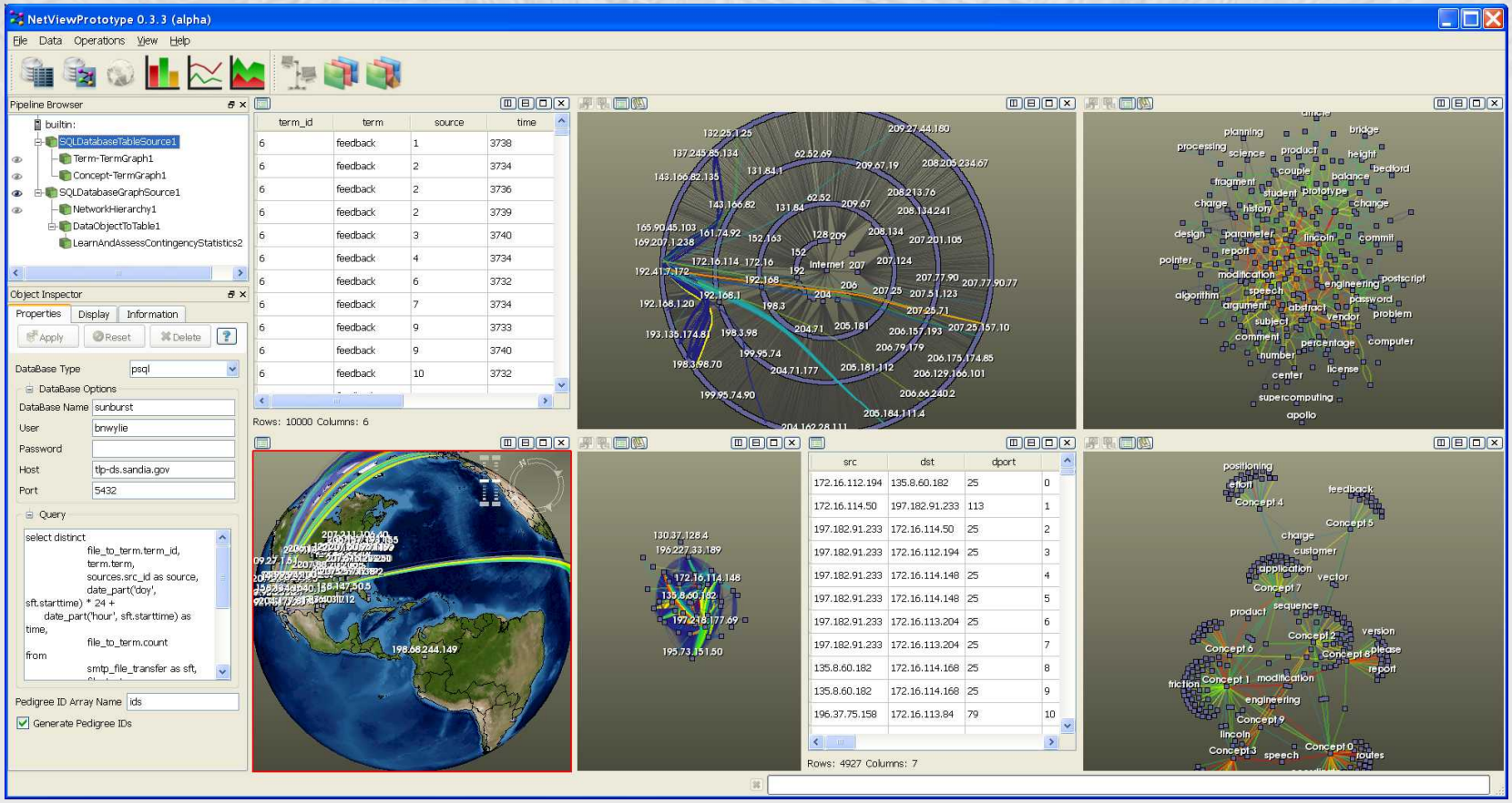
# Useful Plugin References

- Server Manager XML
  - The ParaView Guide, Version 3, section 18.6, pp 262 - 273.
  - The `ParaView/Servers/ServerManager/Resources/` directory.
- General Plugin Information
  - The ParaView Guide, Version 3, chapter 19.
  - http://paraview.org/Wiki/Plugin_HowTo
- Sample Plugins
  - The ParaView/Examples/Plugins directory.
  - The ParaView/Plugins directory.

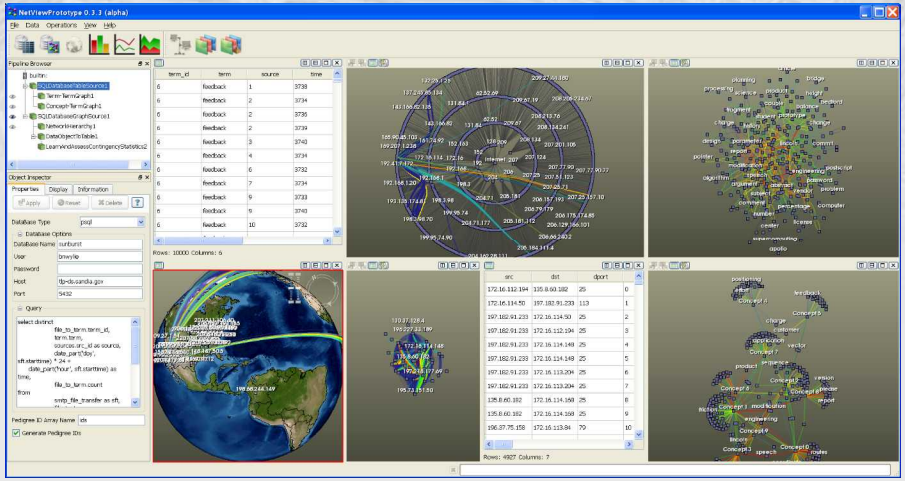
# Vertical Applications

- Traditional ParaView provides a pipeline-oriented interface.
  - Users manipulate the pipeline directly, and must understand it at a low level.
  - Nothing we've seen so far changes that - we add to the list of pipeline components, and we grow the UI by adding views, toolbars, etc.
- Sometimes, you just gotta simplify …

# "Minor Simplification" - OverView



# "Minor Simplification" - Steps



- Copy-n-paste the contents of ParaView/Applications/Client/
- Feels dirty, but it's < 4000 lines-of-code.
- You can modify the `MainWindow` class to simplify:
  - Remove / simplify menus, docking windows, toolbars.
  - Add common operations, custom toolbars, etc.

# Starting From Zero - Demo