Eclipse ICE User Manual

Jay Jay Billings, Andrew Bennett, Alex McCaskey, Robert Smith,
and Greg Watson

Oak Ridge National Laboratory

December 26, 2017

Chapter 1

Installation Details

1.1 Installation

You have been provided a USB stick with all you will need for this tutorial. On this drive is the following:

- An ICE application for your operating system
- A clone of the ICE git repository
- All tutorial documentation and slides for today
- Data files for the tutorial.

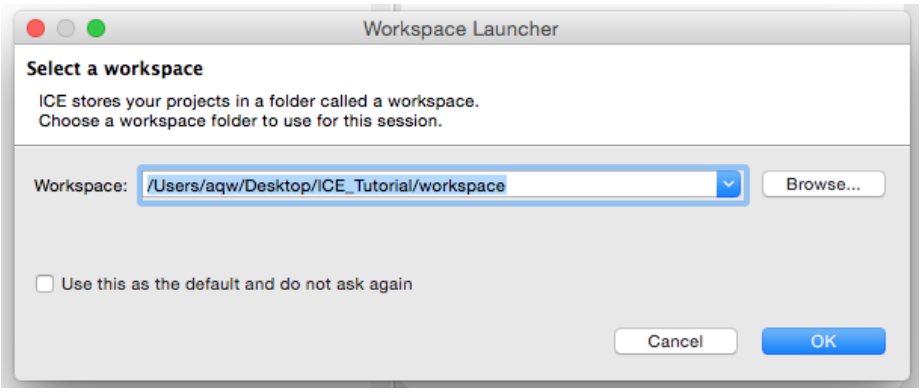
1.1.1 ICE Installation

A number of files must first be copied from the USB stick to your local machine. Please follow the steps below:

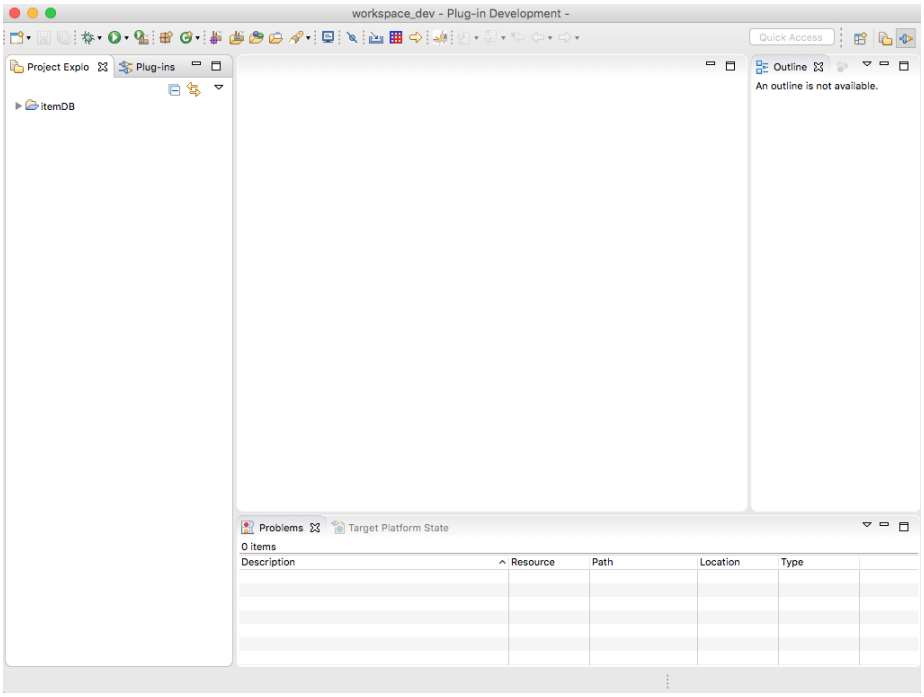
1. Choose a location that is easy to remember and copy the correct `ice-product*.zip` file for your computer's operation system (Linux, Mac OS X, or Windows).
2. Unzip this file to obtain the ICE application executable.
3. Copy the git repository `ice-repository.zip` file to your computer.
4. Unzip this file.

1.1.2 Starting ICE for the first time

To start ICE, double-click on the application executable, i.e. $ICE_SDK file, which is located in the ice.sdk.product*



When ICE opens you should see an empty `Plug-in Development` perspective similar to the image below.

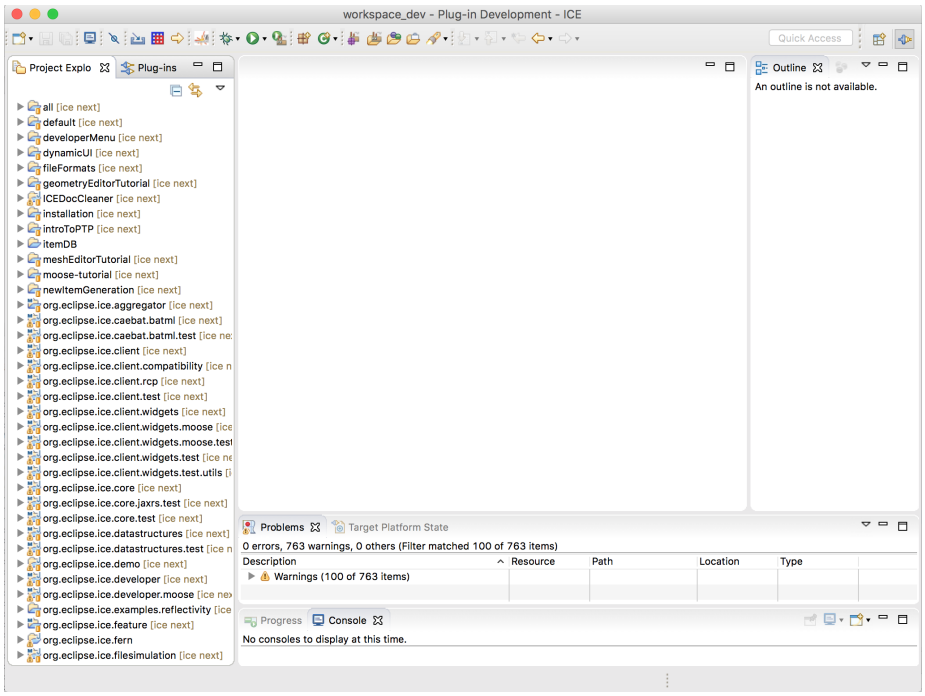


1.1.3 Setting up ICE

In order to create a dashboard for your application, you will need to configure ICE to be able to develop ICE. This *Software Development Kit (SDK) version* of ICE will contain all the components necessary to generate a new instance of ICE that will become your application's dashboard. The first step in doing this is to load the ICE bundles into your workspace. We have provided a developer menu to assist in this process:

1. Select `Developer` → `ICE` → `Import Local Repository`
2. Using the directory dialog, navigate to the git repository you copied from the USB drive.
3. Select `Open`
4. In the `Search results` list, select the checkbox next to the repository
5. Select `Finish`

This should import all the ICE bundles and you should have something like the image below.



At this point, you are ready to begin developing a dashboard for your application!

Chapter 2

The Eclipse ICE Item Project Generator

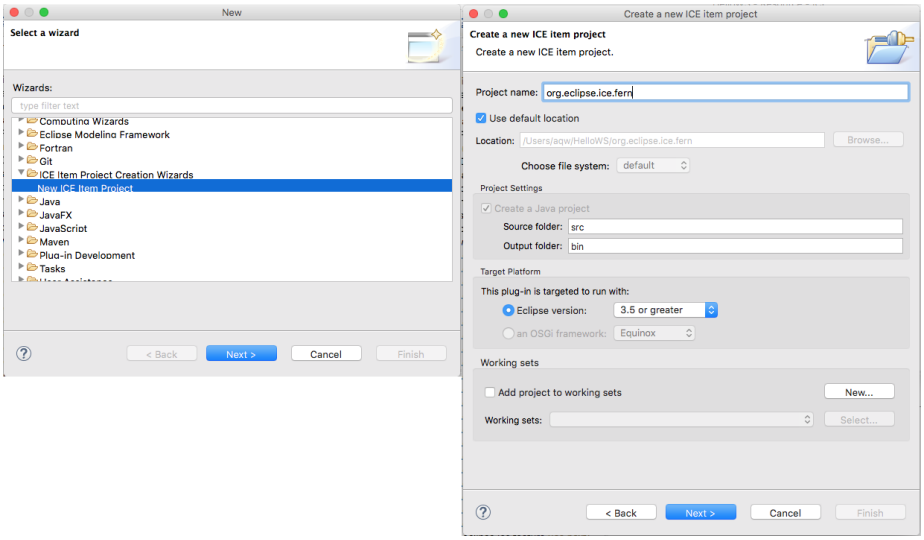
2.1 Overview

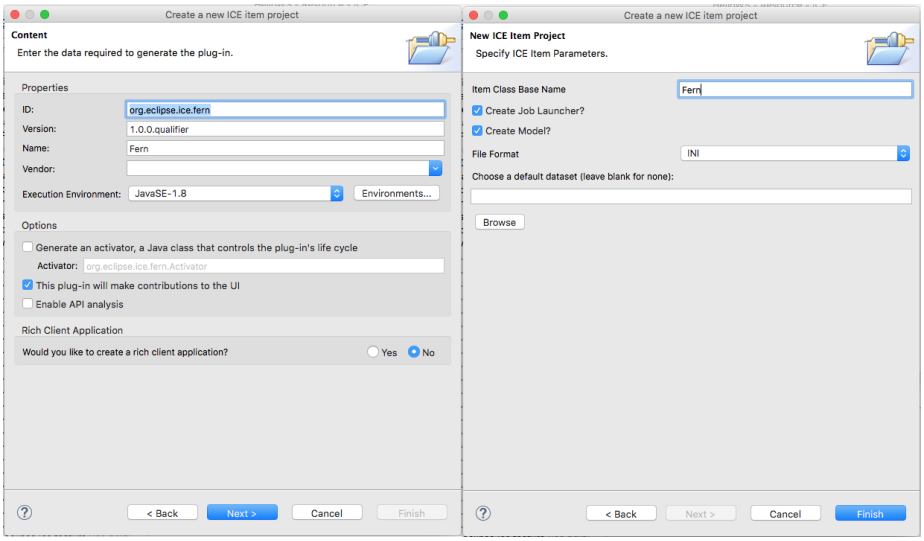
This tutorial will teach you how to create your own ICE Items via the built in tools within ICE. To demonstrate these tools, we will walk through the development of an ICE Item project for the FERN code, a fast, efficient nuclear reaction network solver.

After creating a new ICE Item plugin project, we will demonstrate how to provide a few lines of code to get a Model Item showing in ICE that creates input files for FERN. After that we will add a little code to the FERN JobLauncher to be able to execute FERN locally, remotely, or via the provided Docker image. Before we begin, ensure that you have ICE cloned (Developer > ICE > Clone ICE or Developer > ICE > Import Local Repository) into your workspace.

2.2 Creating the Project

To create a new ICE Item project, navigate to `File` > `New` > `Other` and open the `ICE Item Creation Wizards` folder and select `ICE Item Creation Wizard`. You will be met with a standard new project wizard page, in which you can name your project. We will call ours `org.eclipse.ice.fern`. Once you have named your project click the `Next >` button.





Now you are able to customize the plugin-specific portions of the project.

On this page you need to tell the wizard what you want to use as a base name for your item classes. We will call this one `Fern`. Then, we will specify some information about how the item will handle input data. Fern uses the INI file format to specify data, so we will tell our item to use the built-in functionality for INI files. To do this select `INI` from the `File Format` dropdown.

When you have entered all of the required information you can click the `Finish` button to generate your new ICE Item plugin project. When the project has finished generating you should be able to explore the code that has been created. Within the source directory there will be two packages, each containing two Java classes:

- `org.eclipse.ice.fern.launcher`
  - `FernLauncher.java`
  - `FernLauncherBuilder.java`
- `org.eclipse.ice.fern.model`
  - `FernModel.java`
  - `FernModelBuilder.java`

To add functionality to the project we need to edit the `FernLauncher` and `FernModel` classes.

2.3 Adding Functionality to the New Items

2.3.1 The Fern Model

The `FernModel` will be responsible for creating and validating input parameters for FERN, in the form of a new FERN input file. In order to make the generated code run there are several pieces of information that need to be changed. First, we will need to set up the basic Item idenfification information. This information is set in the setupItemInfo() method. Modify the outputName to match the following (or something of your choosing, with a .ini file extenstion).

```
outputName = "fern_config.ini";
```

The String for the *setName* method will serve as the display name for this Item, so set it as `Fern Model`. As for the String for *setDescription*, this will also be used on the UI for the Item, so provide some text like the following: `This Item constructs input files for the FERN reaction network solver`. The export string will serve as the name of the action that the user can select to write the provided data to file. Set it to something like: `Export to INI`. You should now have a method that looks like this:

```
@Override
protected void setupItemInfo() {
```

```
    setName("Fern Model");
    setDescription("This Item constructs " +
        "input files for the FERN reaction " +
        "network solver");
    outputName = "fern_output.ini";
    exportString = "Export to INI";
    allowedActions.add(0, exportString);
    ioFormat = "INI";
    defaultFileName = "";
}
```

The *allowedActions.add()* line ensures that the export string is provided to ICE as an allowed action, and displayed in the Item Process drop down.

With the identification information configured properly we can begin to implement the Form for this Fern Model. This is done in the *setupForm()* method. The generator has begun the process of implementing this method by instantiating a Form for you to use, getting a reference to the IOService (which provides IReader/IWriter realizations), and providing a commented out example of how to fill out an ICE Form.

For this FERN input model, we want to add the following sections with data entries: a network section with numSpecies, numReactions, numReactionGroups, massTol, fluxFrac, networkFile, rateFile data entries, an initialConditions section with T9, startTime, endTime, initialTimeStep, and density, and an output section with a single popFile data entry. To achieve this for this Item, we will need to add three `DataComponents`, one for the network section, another for the initialConditions section, and a final one for the outputs section. To each of those DataComponents we will add appropriate IEntry instances for each of the data entries we have.

Add the following to your setupForm() method:

```
    // Create the network section
    DataComponent networkComp = new DataComponent();
    networkComp.setName("network");
    networkComp.setDescription("The parameters needed " +
        "to describe the nuclear " +
      "reaction network");
    networkComp.setId(1);

    // Create the IEntries we need for this DataComponent
    StringEntry numSpecies = new StringEntry();
    numSpecies.setName("numSpecies");
```

```
numSpecies.setDescription("The number of species to
    consider");
numSpecies.setDefaultValue("16");

StringEntry numReactions = new StringEntry();
numReactions.setName("numReactions");
numReactions.setDescription("The number of reactions to
    consider");
numReactions.setDefaultValue("48");

StringEntry numReactionGrps = new StringEntry();
numReactionGrps.setName("numReactionsGroups");
numReactionGrps.setDescription("The number of reaction " +
  "groups to consider");
numReactionGrps.setDefaultValue("19");

StringEntry massTol = new StringEntry();
massTol.setName("massTol");
massTol.setDescription("The mass tolerance to consider");
massTol.setDefaultValue("1e-7");

StringEntry fluxFrac = new StringEntry();
fluxFrac.setName("fluxFrac");
fluxFrac.setDescription("The flux fraction to consider");
fluxFrac.setDefaultValue(".01");

FileEntry networkFile = new FileEntry(".inp");
networkFile.setProject(project);
networkFile.setName("networkFile");
networkFile.setDescription("The network file for this
    problem");

FileEntry rateFile = new FileEntry(".data");
rateFile.setProject(project);
rateFile.setName("rateFile");
rateFile.setDescription("The rate file for this problem");

networkComp.addEntry(numSpecies);
networkComp.addEntry(numReactions);
networkComp.addEntry(numReactionGrps);
networkComp.addEntry(massTol);
networkComp.addEntry(fluxFrac);
networkComp.addEntry(networkFile);
networkComp.addEntry(rateFile);

// Create the initial conditions section
```

```
DataComponent initConditionsComp = new DataComponent();
initConditionsComp.setName("initialConditions");
initConditionsComp.setId(2);
initConditionsComp.setDescription("The parameters " +
  "needed to describe the initial " +
  "conditions for the problem");

StringEntry t9 = new StringEntry();
t9.setName("T9");
t9.setDescription("The temperature in Kelvin x 10^9");
t9.setDefaultValue("7.0");

StringEntry startTime = new StringEntry();
startTime.setName("startTime");
startTime.setDescription("The start time for the
    simulation.");
startTime.setDefaultValue("1e-20");

StringEntry endTime = new StringEntry();
endTime.setName("endTime");
endTime.setDescription("The end time for the simulation");
endTime.setDefaultValue("1e-3");

StringEntry initialTimeStep = new StringEntry();
initialTimeStep.setName("initialTimeStep");
initialTimeStep.setDescription("The initial time step " +
  "for the simulation.");
initialTimeStep.setDefaultValue("1.2345e-22");

StringEntry density = new StringEntry();
density.setName("density");
density.setDescription("The initial density.");
density.setDefaultValue("1e8");

initConditionsComp.addEntry(t9);
initConditionsComp.addEntry(startTime);
initConditionsComp.addEntry(endTime);
initConditionsComp.addEntry(initialTimeStep);
initConditionsComp.addEntry(density);

// Create the outputs section
DataComponent outputComp = new DataComponent();
outputComp.setName("output");
outputComp.setDescription("The parameters needed to output
    data.");
outputComp.setId(3);
```

```
StringEntry popFile = new StringEntry();
popFile.setName("popFile");
popFile.setDescription("The name of the output populations
    file");
popFile.setDefaultValue("popFile.csv");

outputComp.addEntry(popFile);

// Add the components to the Form
form.addComponent(networkComp);
form.addComponent(initConditionsComp);
form.addComponent(outputComp);
```

Now we have a Form constructed for a typical FERN execution.

The default generated implementation of the process method is sufficient to be able to create new Fern INI input files.

2.3.2 Fern Launcher

The Fern Launcher handles the actual execution of the FERN application. The generator creates the FernLauncher as a subclass of ICE's JobLauncher, which provides a large array of features and functionality. As a subclass of JobLauncher, the FernLauncher enables users to execute Fern locally or remotely. To do so, we just need to add a small amount of code that customizes the ICE job launching capabilities for Fern.

The first bit of code to add to the FernLauncher specifies the name of the actual Fern executable. In the setupItemInfo() method, set the execCommand to the following:

```
execCommand = "${installDir}fern-exec";
```

This tells ICE that the Fern executable is called `fern-exec`, and to set the overall execution command to it's install path plus the executable name. The installDir flag will tell ICE to insert the user-specified executable location (provided through the graphical form editor') into the execCommand, with a trailing OS-specific path separator. This install directory is specified through the Hosts Table on the editory.

We also need to inform the JobLauncher what other files are involved in this execution. To do that, the JobLauncher provides an addInputType() method.

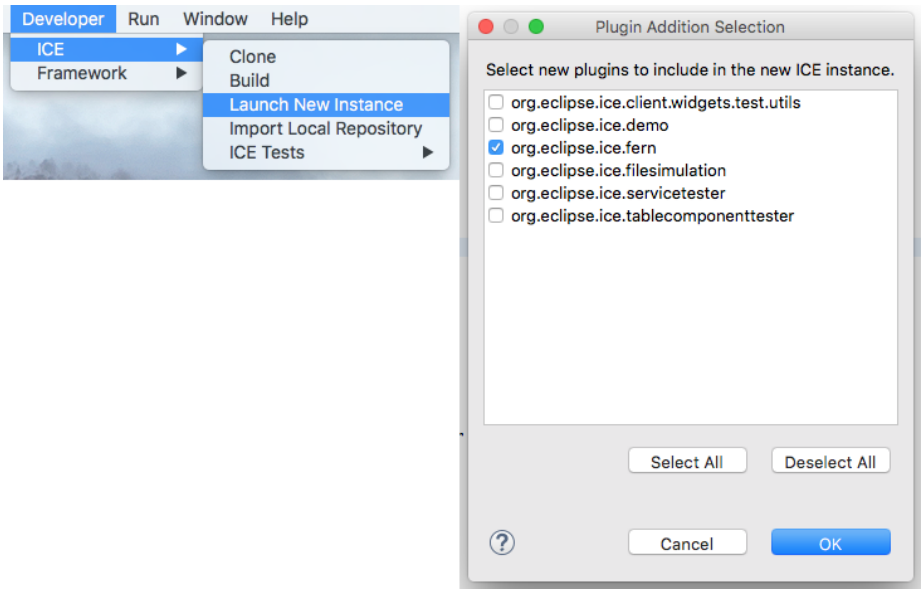
Add the following to setupForm():

```
addInputType("Network File", "networkFile",
        "Network File Description", ".inp");
addInputType("Rate File", "rateFile", "
        Rate File Description", ".data");
```

And that should be it. The generator has taken care of everything else for us. We are now ready to launch ICE with our Fern plugin, and use the Fern Items we have just created.

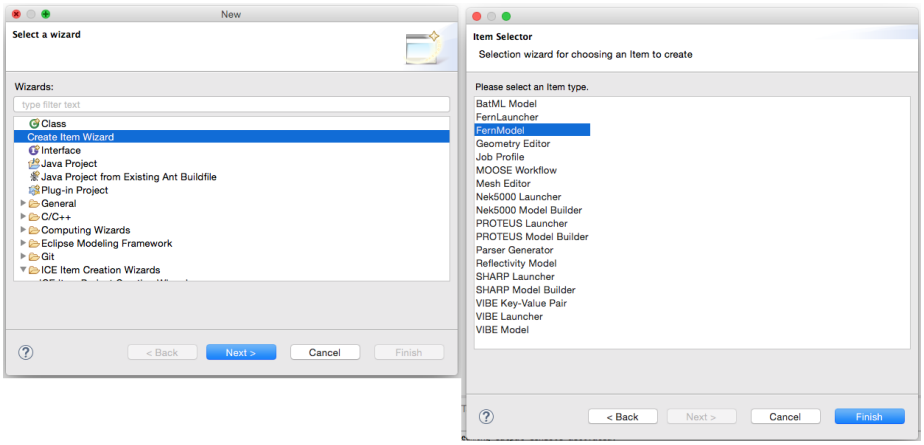
2.4 Using the New Fern Items

Now, using these new Items is easy. From the Developer top-level menu, select ICE > Launch New Instance. This will display a dialog asking you which new plugins you'd like to include as part of the new ICE instance.

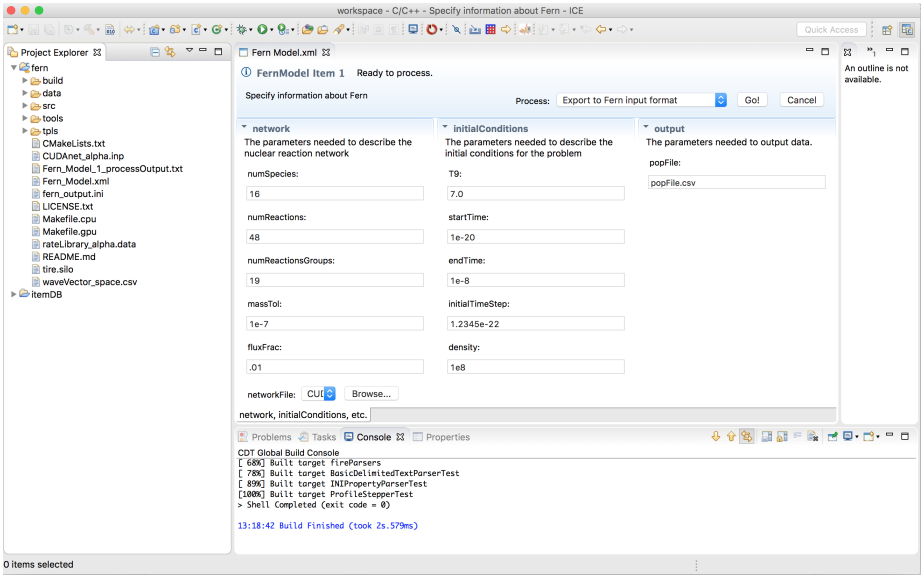


Select `org.eclipse.ice.fern` and click Ok. This will create and launch a new instance of ICE that includes your custom Item plugin.

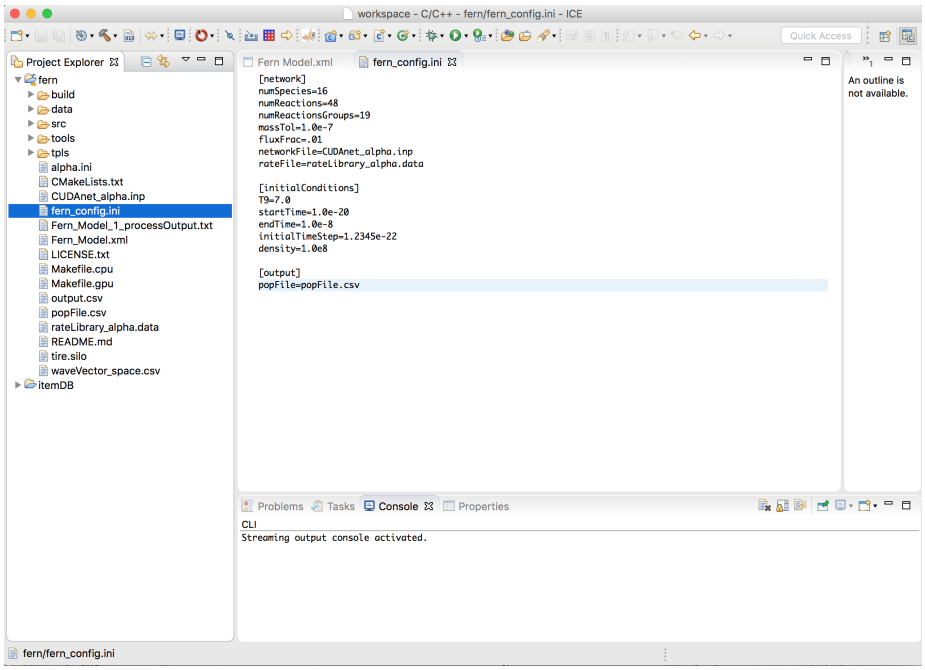
With a new ICE instance open, close the Welcome view if necessary and go to `File` > `New` > `Other` and select the Create Item Wizard.



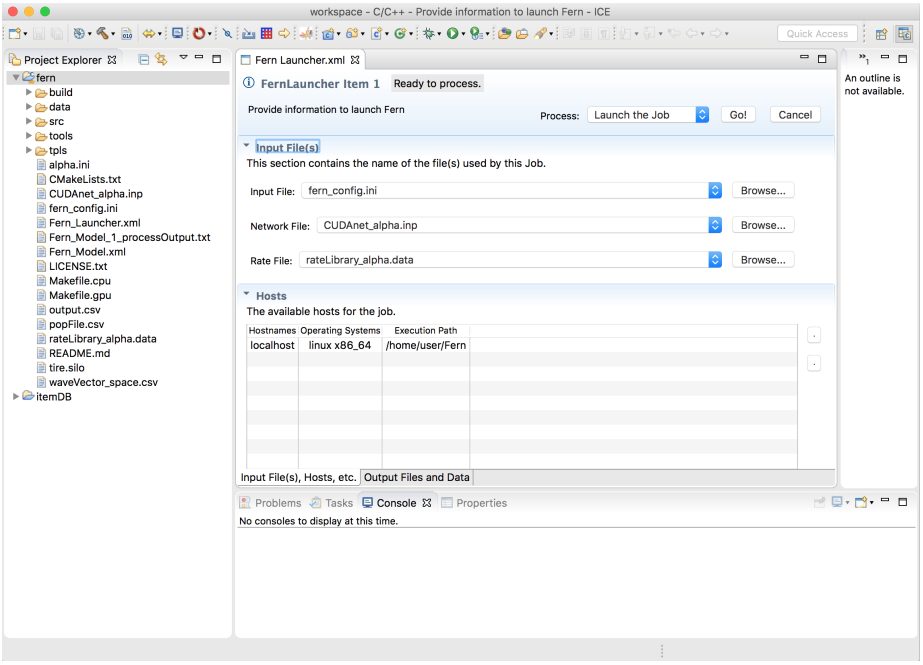
After selecting the FernModel Item, you will be presented with the view in the figure below.



Here you can modify the various defaults with the values you would like for a given Fern simulation. Once done, simply save the Item and click Go on the Export to INI Process. This will execute the process of creating a new INI Fern input file for use with the Fern Launcher. You can check the result by opening the fern_output.ini file, as shown below.



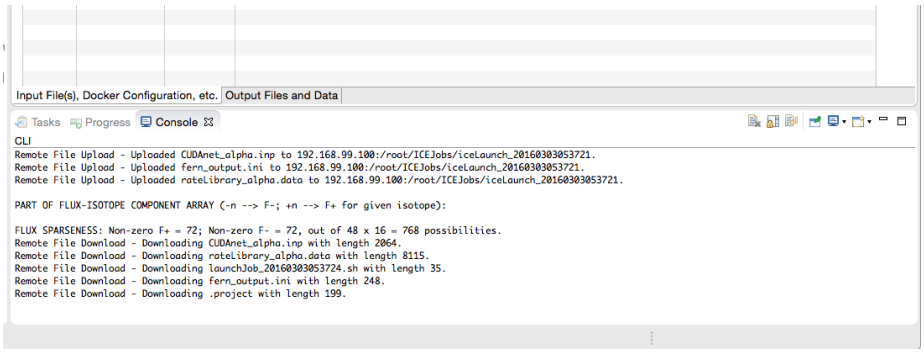
Now you can similarly create a new Fern Launcher. After creating the Launcher, you should see a view like below.



To configure a launch, simply set the correct input file, along with its dependent network and rate files.

At this point, if you had Fern built on your local machine, or if you had it built on some remote host, you could configure that in the Hosts table. ICE would then execute Fern based on that input.

After the execution you should see the results in the Console, as shown below.



The execution should have produced a CSV file with the computed populations. You can double-click that file to view them graphically in the ICE Plot Editor.

Chapter 3

Working with Resource Components

Resource Components are ICE Components which contain a grid of visualization resources. These resources can display files from a variety of sources, such as CSV files or VisIt visualizations. Geometry and Mesh editors allow for editing of shapes or meshes. All three can be added to an Item to offer visualization of data.

3.1 Prerequisites

This tutorial assumes you will be making use of the sample classes in the org.eclipse.ice.demo.visualization package for convenience. The two relevant classes are VisualizationModel and VisualizationModelComplete. The former is an example of a bare bones ICE Item, while the latter is the same class with all the extra code required for the visualization components already added in.

If you are writing your own extension of ICE's Item class, then the same principles can be used to add these components to it. See the New Item Generation Tutorial, in the docs/newItemGeneration/ folder of the ICE repo for details on how to create an Item.

You will also need access to an installation of VisIt version 2.9.2. If not already installed, there is a copy of the software in the VisIt folder of your USB drive. Windows users should open the Windows installer, while users of other OSs can just copy the appropriate folder onto their machine.

3.2 Adding the Components

First, open the Item file by double clicking it in the `Package Explorer`. For the tutorial, this will be VisualizationModel.java in the org.eclipse.ice.demo.visualization.model package. Its location can be seen in figure 3.1.

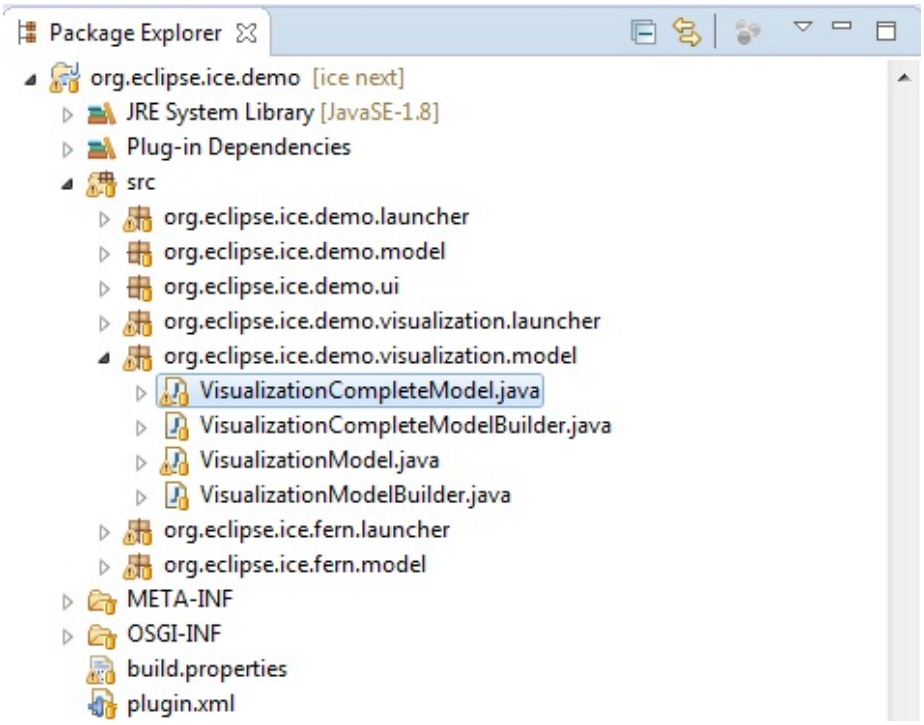


Figure 3.1: The package structure for org.eclipe.ice.demo bundle

First, add the neccesary imports for the components being added to your Item.

```
import java.io.IOException;
import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.eavp.viz.modeling.ShapeController;
import org.eclipse.eavp.viz.modeling.ShapeMesh;
import org.eclipse.eavp.viz.modeling.base.BasicView;
import org.eclipse.ice.datastructures.form.GeometryComponent;
import org.eclipse.ice.datastructures.form.MeshComponent;
import org.eclipse.ice.datastructures.form.ResourceComponent;
import org.eclipse.ice.datastructures.resource.VizResource;
```

Next, copy and paste the following example code into the end of your Item's setupForm() method.

```
//Create the resource component
ResourceComponent resourceComponent = new ResourceComponent();

//Set the component's data members
resourceComponent.setName("Resources");
resourceComponent.setDescription("Results");
resourceComponent.setId(2);

//Declare the files and resources
VizResource csvResource = null;
VizResource visItResource = null;
IFile csvFile = null;
IFile visItFile = null;


//If the file was found, create the CSV resource and add it to the component
try{

//Open the files
csvFile = ResourcesPlugin.getWorkspace().getRoot().getProject("itemDB")
.getFile("fib8.csv");
visItFile = ResourcesPlugin.getWorkspace().getRoot().getProject("itemDB")
.getFile("tire.silo");

//If the file was found, create the CSV resource and add it to the component.
if(csvFile.exists()){
csvResource = new
                    VizResource(csvFile.getLocation()
                    .toFile());
     resourceComponent.addResource(csvResource);
}

//If the file was found, create the VisIt resource and add it to
//the component
if(visItFile.exists()){
visItResource = new
                    VizResource(visItFile.getLocation()
                    .toFile());
resourceComponent.addResource(visItResource);
}
}
catch(IOException e){
```

```
e.printStackTrace();
}

//Create the geometry component
ShapeController geometryRoot = new ShapeController(new
    ShapeMesh(), new BasicView());
GeometryComponent geometryComponent = new
    GeometryComponent();
geometryComponent.setGeometry(geometryRoot);

//Create mesh component
MeshComponent meshComponent = new MeshComponent();

//Add the components to the form
form.addComponent(resourceComponent);
form.addComponent(geometryComponent);
form.addComponent(meshComponent);

//Set the context on the Form
form.setContext("visualization");
```

This code will add a Resource Component, Geometry Component, and Mesh Component to your Item and load fib8.csv and tire.silo into the Resource Component.

You will also need to add the org.eclipse.ice.demo bundle to your operating system's run configuration, as explained in the new item generation tutorial (found in docs/newItemGeneration in the ICE repo).

You can compare your code to the XSEDEVisualizationModel.java file, which has a working version of the Item with the tutorial code put in.

When done, launch ICE. If you have never launched ICE before, go to the org.eclipse.ice.product bundle, right click the .launch file for your operating system, and select the first option under the `Run as...` submenu.

3.3 Using the Resource Component

ICE uses an instance of VisIt, a visualization code from Lawrence Livermore National Laboratory, running in another process for visualization. ICE can also use ParaView, but this tutorial will only cover VisIt, as the processes for conencting to ParaView and using a ParaView Plot Editor are largely similiar to those for VisIt.

3.3.1 Establishing a VisIt Connection

In order to visualize resources containing VisIt files, ICE must be connected to a running VisIt installation. To set up this connection, select `Windows` → `Preferences...` in ICE's menu bar. (On Mac OS X, `Preferences` is instead located under `ICE` in the menu bar.)

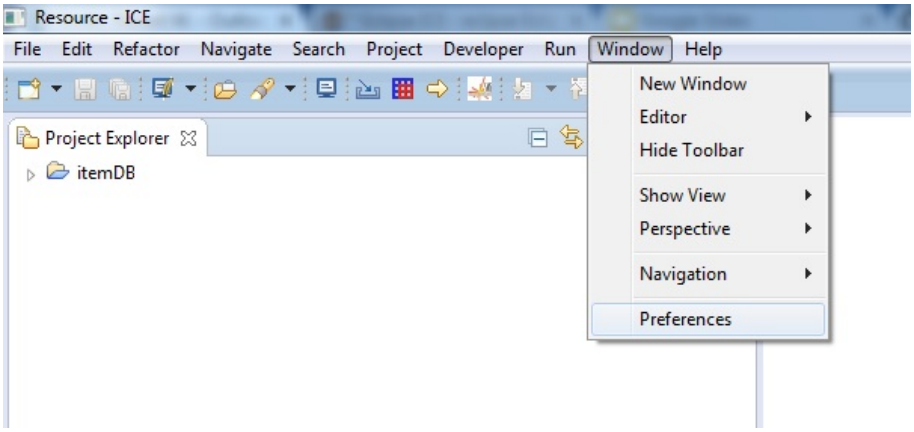


Figure 3.2: The ICE `Preferences Menu` for Windows and Linux. It will be located under `ICE` instead of `Window` on Mac.

Select `Visualization` → `VisIt` in the tree on the left side of the `Preferences` window.

Press the button with a "+" symbol in the upper right of the `Preferences Menu` (highlighted in Figure 3.3) to add a new row to the table. Click on the `Path` cell of the new row and put the path of your installation of VisIt.

For example, on `Windows`, if assuming a username of "username", and VisIt was installed in the default location, the path will be:

`C:\Users\username\AppData\Local\Programs\LLNL\VisIt 2.9.2.`

On `Linux`, the path will be based on where you extracted the visit2_9_2.linux-x86 folder, ending with /visit2_9_2.linux-x86_64/bin. If you unzipped it to the desktop, it will be:

`/home/username/Desktop/visit2_9_2.linux-x86_64/bin`

On `Mac OS`, the path will be based on the location you put the Visit application. If placed in the Applications folder it will be

`/Applications`

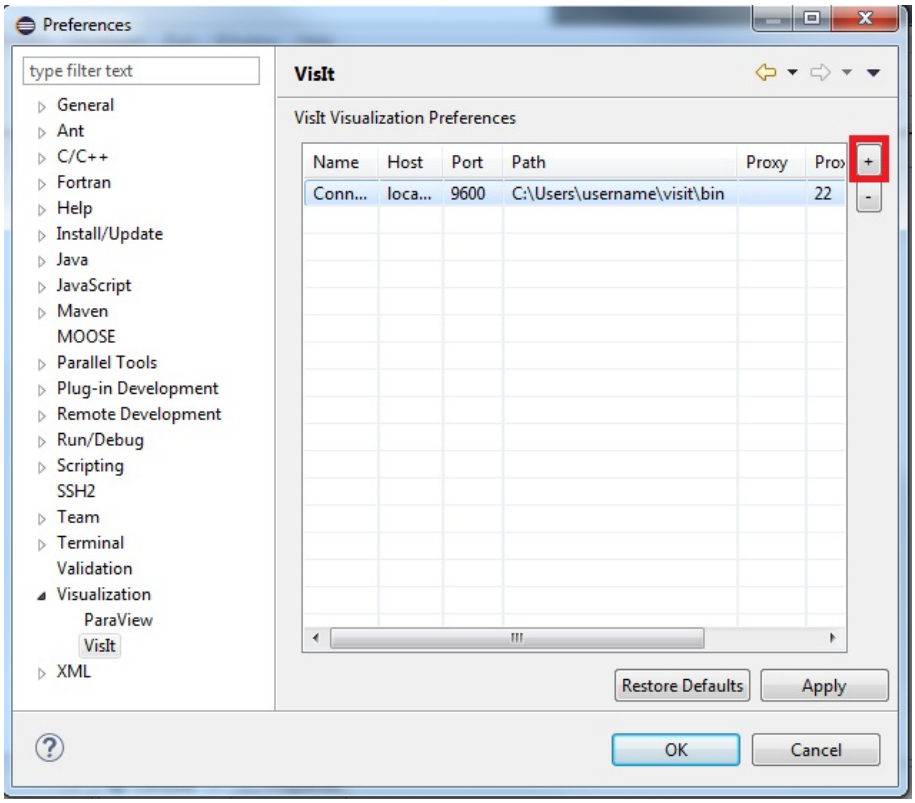


Figure 3.3: The `VisIt` preferences page in the `Preferences` dialog. The hilighted button will add a row for a new connection to the table.

Press `Apply`, then `OK`, both in the lower right hand corner of the `Preferences Menu`. ICE will now open and connect to this VisIt installation each time ICE is opened.

3.3.2 Opening Your Item

Before opening your new Item, select the itemDB folder in the `Project Explorer` and press the import button, highlighted below.

Select the files you want to visualize and click OK. For the tutorial, these should be the fib8.csv and tire.silo files from the USB drive's Data directory. You should now see the two files within the itemDB folder.

Then select `File` → `New` → `Other...` from the toolbar. In the new dialog, select the `Create Item Wizard` and hit `Next`. Then select `Visualization Model`

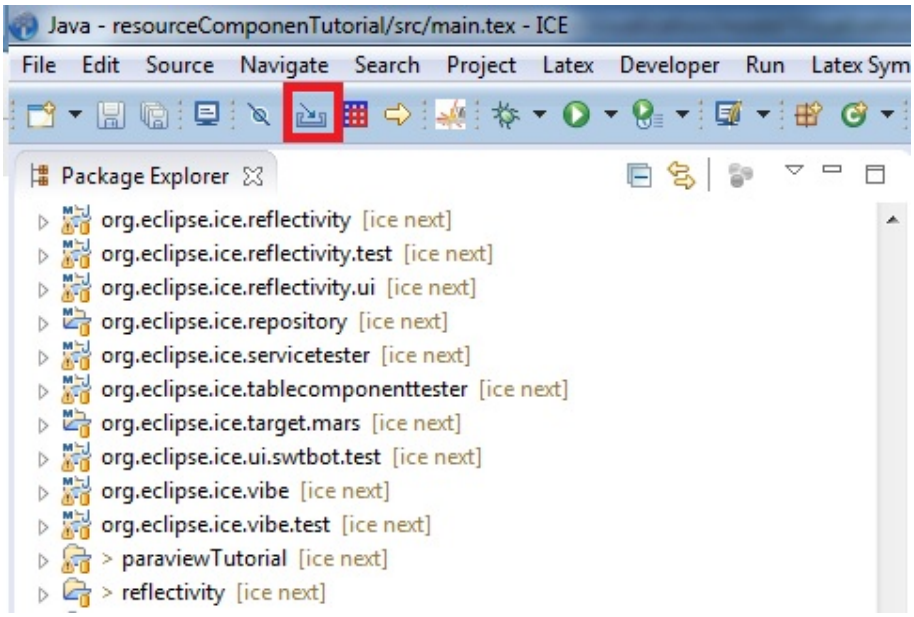


Figure 3.4: ICE's `import` button.

and press `Finish`. You can also select the `Visualization Model (Pre-completed)` if you skipped the first part of the tutorial.

Finally, switch to the ICE Perspective to ensure that the neccesary Views will be open. To do this, click the `Open Perspective` button in the upper right of the workbench screen.

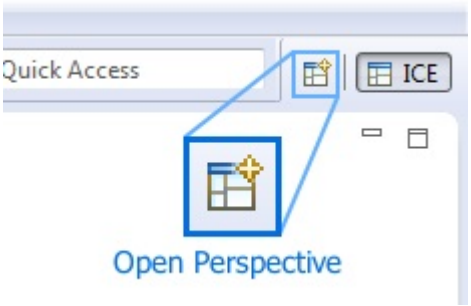


Figure 3.5: ICE's `Open Perspective` button.

In the dialog, select `ICE` and press OK.

3.3.3 Managing the Resources

Your Item should look like this when it loads.

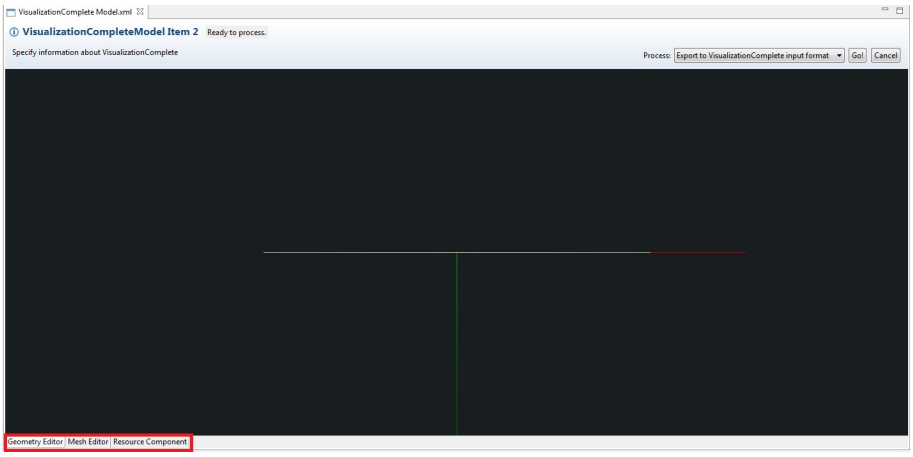


Figure 3.6: The `Visualization Model` after it is initially opened. Hilighted in the lower left are the tabs for the three components, the `Geometry Editor`, the `Mesh Editor`, and the `Resource Component`.

In the lower left of the `Visualization Model` are three tabs, hilighted in Figure 3.6. Switch to the `Resource Component` tab in your Item and to the `Resources` tab on the left, as shown below.

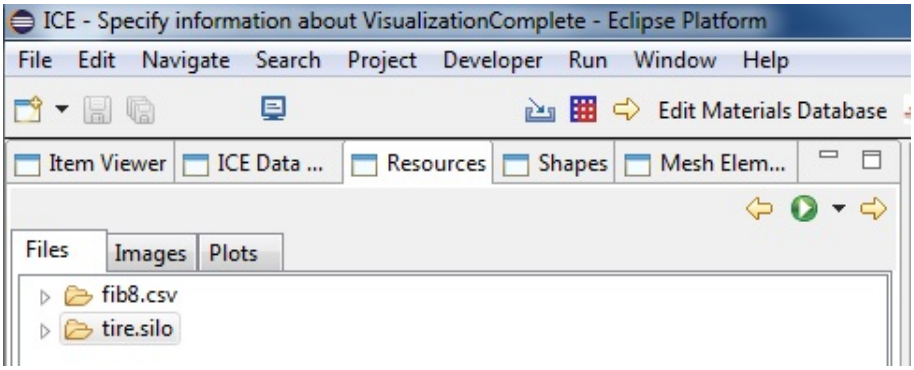


Figure 3.7: The `ICE Perspective`'s `Resources` tab.

Double click on both the file names to load them into the Resource Component.

At the top left of the `Visualization Model` will be controls for the component's layout.

Figure 3.8: The layout controls for the `Resource Component`.

The `Clear` button will close all plots in the component. The other two controls will allow you to specify the number of rows and columns in the grid. Be careful when reducing them, as any plots which no longer fit in the grid will be closed.

If you hover over a plot, a button will appear in its upper left hand corner. Clicking it will close that plot.

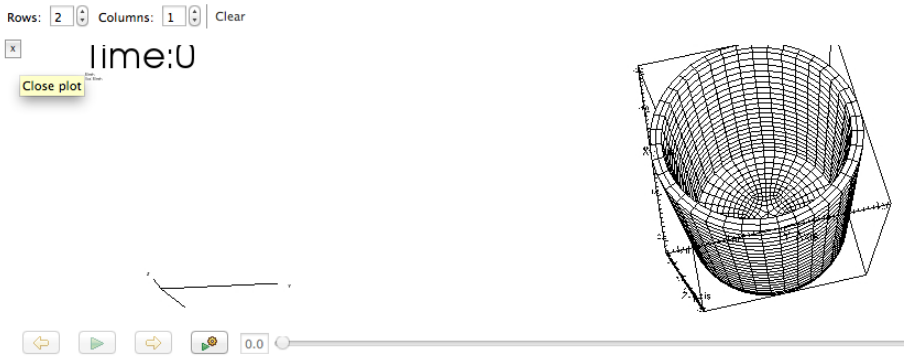


Figure 3.9: The X button in the upper left will close the plot.

3.3.4 Interacting with VisIt Plots

A VisIt plot will contain a 3D visualization of some model. You can click and drag within the plot to rotate the image and zoom by scrolling your mouse wheel. Right clicking in the plot will open a context menu, providing options for how the model will be displayed.

At the bottom of the plot will be a series of controls for animation. If your plot does not have time series data, they will be greyed out.

The plot can be set to display an arbitrary time step by either dragging the slider or by typing a time into the box to its left.

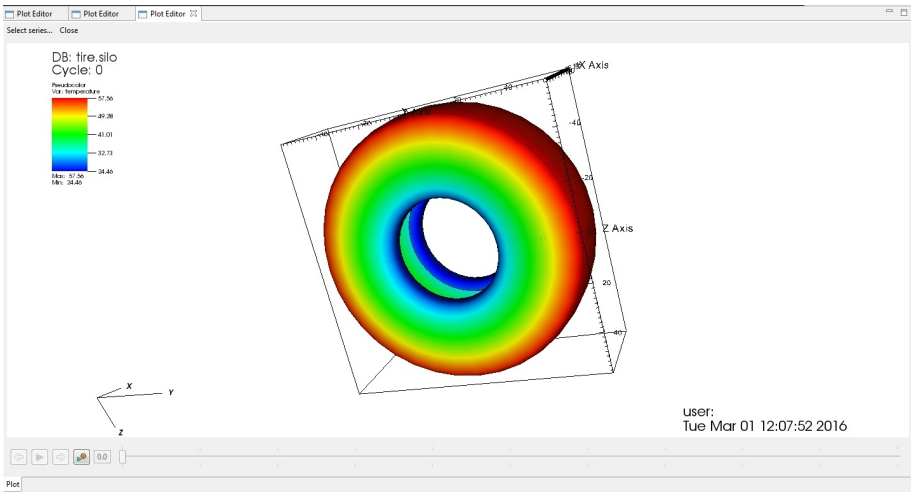


Figure 3.10: An example of a file open in VisIt displayed in a `Plot Editor`.

Figure 3.11: The `Plot Editor`'s animation controls.

3.3.5 Interacting with CSV Plots

The top of the CSV plot has a row of buttons which control various aspects of the graph's presentation. Right clicking will open a context menu allowing you to choose which of the available series to plot.

3.3.6 Editing 3D Structures

ICE also contains capabilities to render graphics with the Geometry Editor and Mesh Editor. Programatically populating these editors with custom input is beyond the scope of this tutorial. However, what follows will be a brief overview of the editors' functionality.

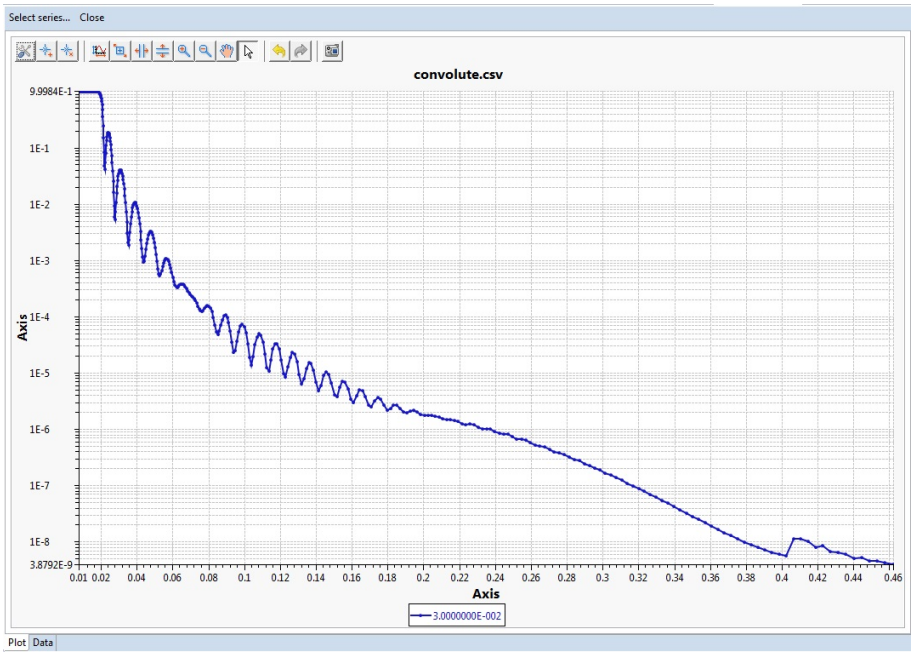


Figure 3.12: An example of a CSV file. The Y axis is displayed on a logarithmic scale.

The Geometry Editor

Now switch to the `Geometry Editor` tab in you the `Visualization Model` and the `Shapes` tab on the left.

Clicking the `Add Primitives` button will display a drop down of primitive shapes which can be added to the scene.

Complex shapes can similarly be added using the `Add Complex` button.

Primitive shapes can be added under complex shapes by selecting anything beneath the desired parent complex shape before adding the new primitive.

The three other buttons are responsible for creating copies of or removing selected shapes from the tree.

The Transformation View, in the lower left of the workbench screen, has spaces to set the rotation, scale, and translation of a selected object.

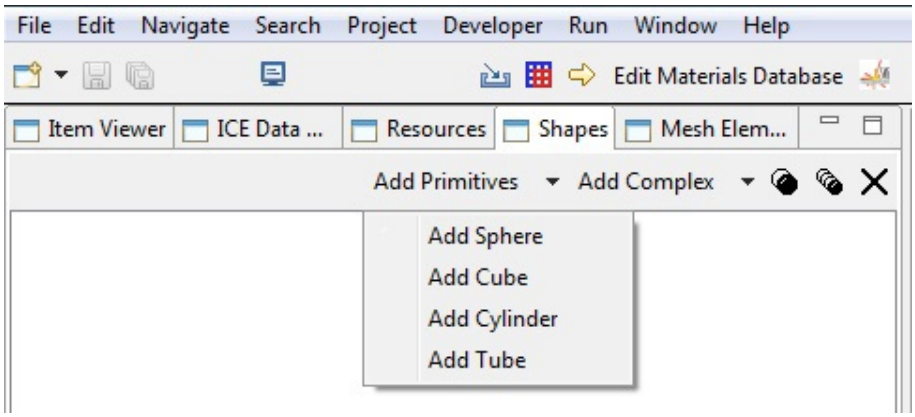


Figure 3.13: The `Add Primitives` button displays a menu of shapes to add.

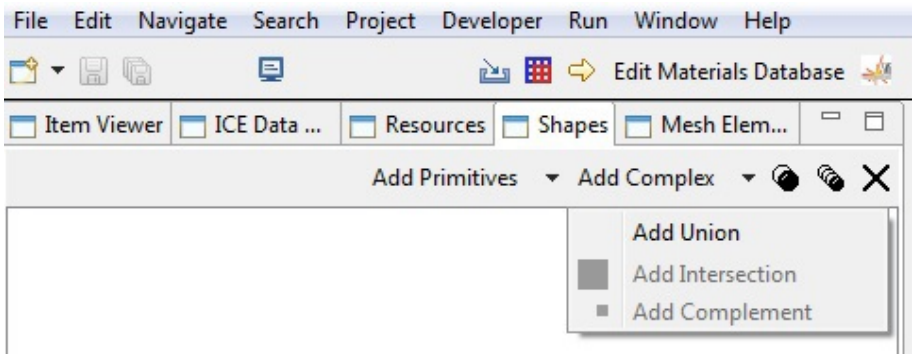


Figure 3.14: The `Add Complex` button can add a Union of shapes.

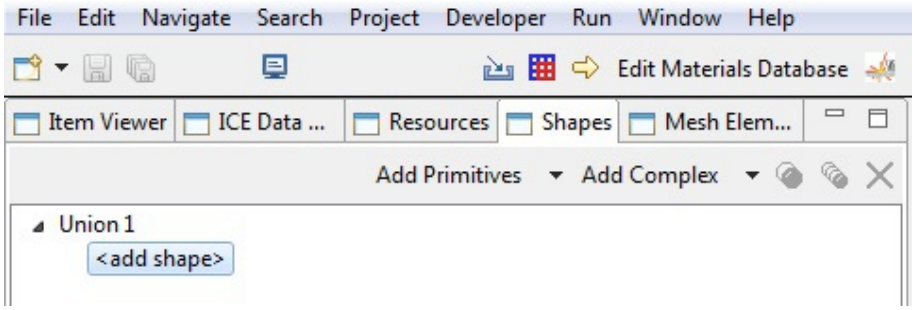


Figure 3.15: An example of a constructive solid geometry tree in the `Shapes` tab. Adding shapes with the highlighted leaf selected will cause them to become children of Union 1.

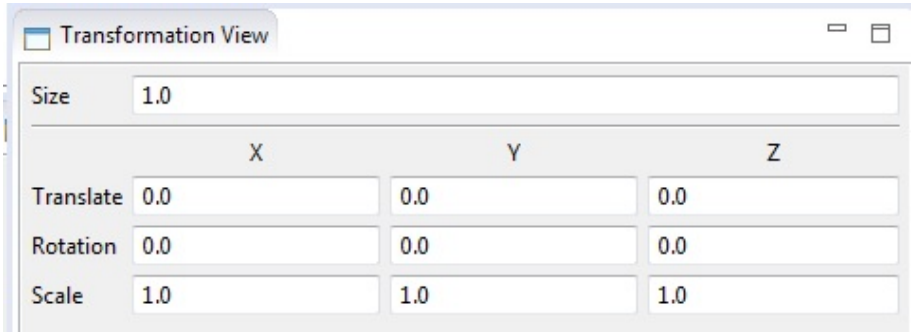


Figure 3.16: The `Transformation View`.

The Mesh Editor

Now switch to the `Mesh Editor` tab in your Item and `Mesh Elements` tab on your left.

Clicking within the grid will create a vertex, until the fourth completes the polygon.

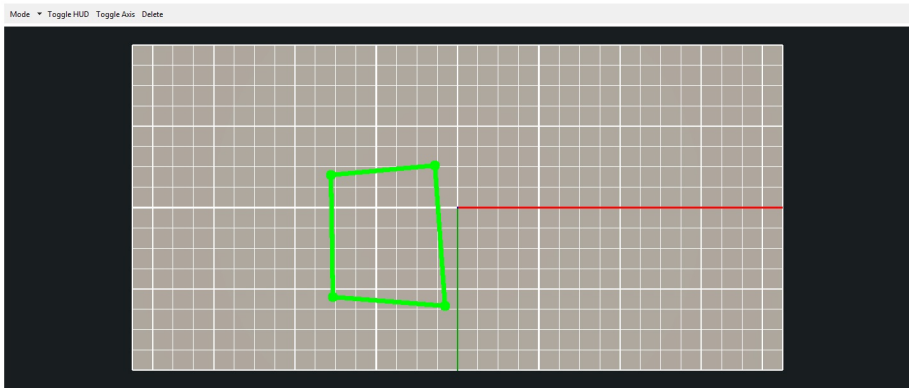


Figure 3.17: A new polygon, colored green because the user has not yet permanently added it to the mesh.

Click again to make the polygon permanent, signified by turning purple, or hit `Esc` to cancel.

The `Mode` button in the top left allows you to switch between `Add Elements` mode, used previously, and `Edit Elements` mode.

In edit mode, you can click a vertex (or vertices) to select them.

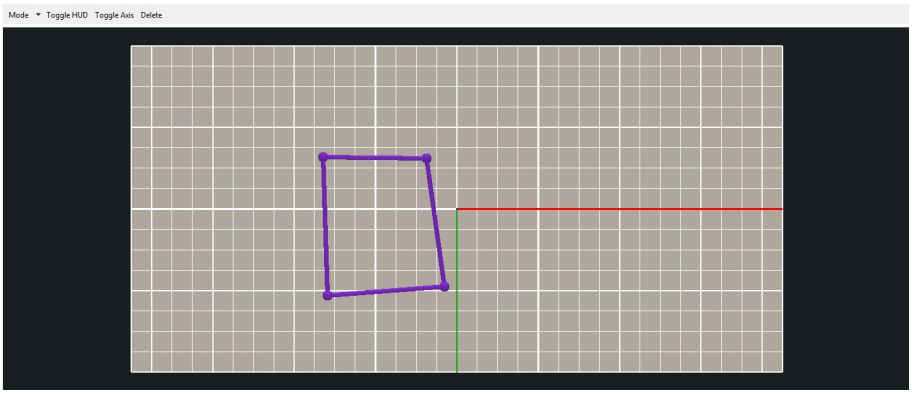


Figure 3.18: The polygon turns purple after clicking, showing that it has been finished.

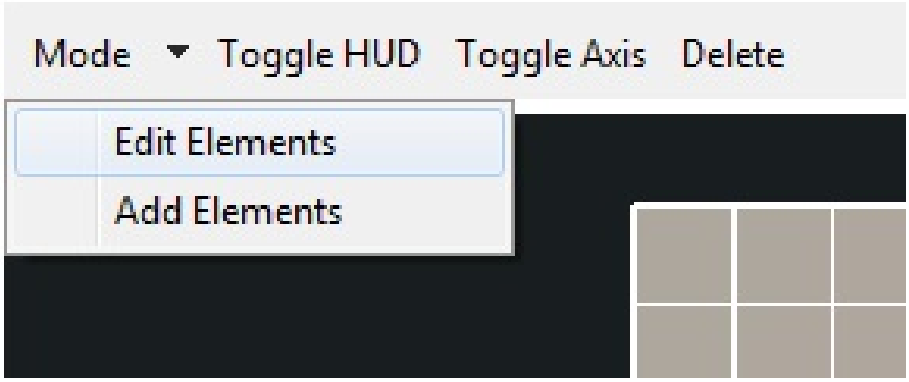


Figure 3.19: The `Mode` button allows switching between `Edit Elements` mode and `Add Elements` mode.

You can click and drag a vertex to move all selected vertices around the grid.

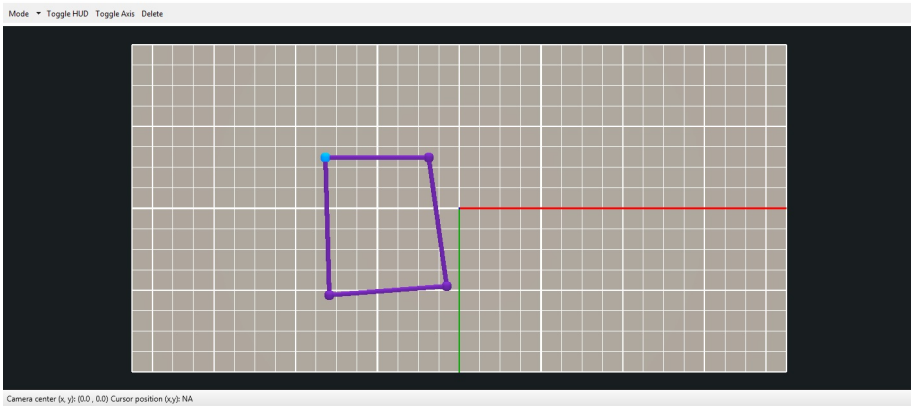


Figure 3.20: A selected vertex will turn blue.

3.4 Further Reading

This tutorial has only given a brief overview of the ways in which you can use ICE's visualization tools. For more detailed information, look under the `docs` folder in the ICE repository. The visualization folder contains a tutorial on the CSV and VisIt plots, while the geometryEditor and meshEditor folders have tutorials on the geometry and mesh editors, respectively.

Chapter 4

Dynamically Extending the Eclipse ICE UI

4.1 Dynamic UI

This tutorial will show you how to create custom, dynamic UI extensions to Eclipse ICE.

What you will need for this tutorial:

- Experience creating Eclipse plugins
- Experience writing UI code with SWT
- Experience creating an ICE Item

4.1.1 Introduction

ICE makes some educated guesses based on the type of your components and information that it can glean from your data to figure out the best way that it can generate the the UI. However, after you create your first set of Items, you might find yourself wondering if you can change the way that ICE auto-generates the UI to better fit your needs. ICE lets you do this by setting the Context of your Form and Components with the setContext() operation. Setting the context with a string that is unique to your project will let ICE look up UI extensions that you create and publish through the Eclipse 4 framework.

There are several important things to consider before you start extending the UI. First, how much work do you want to do? Some of the UI constructs in ICE are quick to change, such as EntryComposites for showing Entries, but others, like GeometryPage and MeshPage, could require significant work because of the level of graphics involved.

This tutorial will show you how to change two pieces of ICEs UI: The page for showing GeometryComponents and an EntryComposite. We will only show you how to change the GeometryPage, not how to actually generate new 3D graphics. The source code for this tutorial is in the ICE repo in a project called org.eclipse.ice.demo.

4.1.2 Create an ICE Item Project

Use the ICE Item Project Generation Wizard to create a new Item with a GeometryComponent and a DataComponent with one Entry in the Form. You can copy the following code into your setupForm() operation:

```
@Override
    public void setupForm() {
      form = new Form();

      ioService = getIOService();

      // Create a geometry component
      GeometryComponent geomComp = new GeometryComponent();
      geomComp.setName("Geometry");
      geomComp.setDescription("A geometry");
      geomComp.setContext("demo-geometry");
      geomComp.setGeometry(
            new ShapeController(new Shape(), new BasicView()));

      // Create a data component
      DataComponent dataComp = new DataComponent();
      dataComp.setName("Data");
      dataComp.setDescription("Some Data");
      dataComp.setContext("demo");
      // Need to set the id since geomComp is number 1
      dataComp.setId(2);

      // Create an Entry for the data component
      IEntry entry = new StringEntry();
      entry.setName("Data Entry");
      entry.setDescription("An Entry with Important Data");
```

```
    entry.setContext("demo-entry");
    // Add the Entry to the data component
    dataComp.addEntry(entry);

    // Add both components to the Form, showing the data
        component first.
    form.addComponent(dataComp);
    form.addComponent(geomComp);

    // Set the context on the Form
    form.setContext("demo");

    return;
  }
```

Note that your import packages lines should look this the following:

```
import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.eavp.viz.service.modeling.AbstractView;
import org.eclipse.eavp.viz.service.modeling.ShapeController;
import org.eclipse.eavp.viz.service.modeling.ShapeMesh;
import org.eclipse.ice.datastructures.entry.IEntry;
import org.eclipse.ice.datastructures.entry.StringEntry;
import org.eclipse.ice.datastructures.form.DataComponent;
import org.eclipse.ice.datastructures.form.Form;
import org.eclipse.ice.datastructures.form.FormStatus;
import org.eclipse.ice.datastructures.form.GeometryComponent;
import org.eclipse.ice.io.serializable.IIOService;
import org.eclipse.ice.io.serializable.IReader;
import org.eclipse.ice.io.serializable.IWriter;
import org.eclipse.ice.item.model.Model;
```

You may need to add some of these packages to your Manifest file.

If you have not created an Item in ICE before, please see `the ICE Item Generation tutorial` to do this.

Add this plugin to the launch configuration for your system, launch ICE and make sure that you can successfully create a DemoModel Item. You should have see a Form with two pages as in figures 4.1 and 4.2.

The rest of the tutorial will look at replacing the routine that draws the Entry on the first page and the entire Geometry Page with your own custom

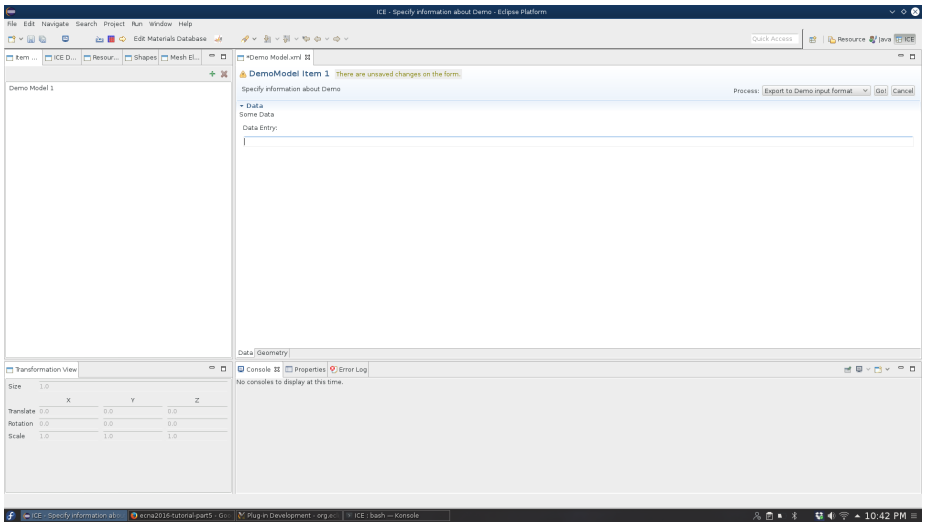


Figure 4.1: Default Entry Composite on a Form

page. You wont have to make any additional changes to your Item (assuming it works as shown above). Note that in the code above we need to use different context names for the Entry and the GeometryComponent so that the context can uniquely identify the routines that draw each.

4.1.3 Create an IPageProvider and IPageFactory

We will replace the Geometry Page first. Create a new package.This requires two pieces to properly locate your Page, through your IPageFactory, and to draw your Page, through you IPageProvider.

Start by adding the following packages to your import packages block in your Manifest file:

- org.eclipse.ice.client.widgets.providers
- org.eclipse.ice.client.widgets.providers.Default
- org.eclipse.ui.forms
- org.eclipse.ui.forms.editor
- org.eclipse.ui.forms.widgets
- org.eclipse.e4.core.contexts
- org.eclipse.e4.core.di

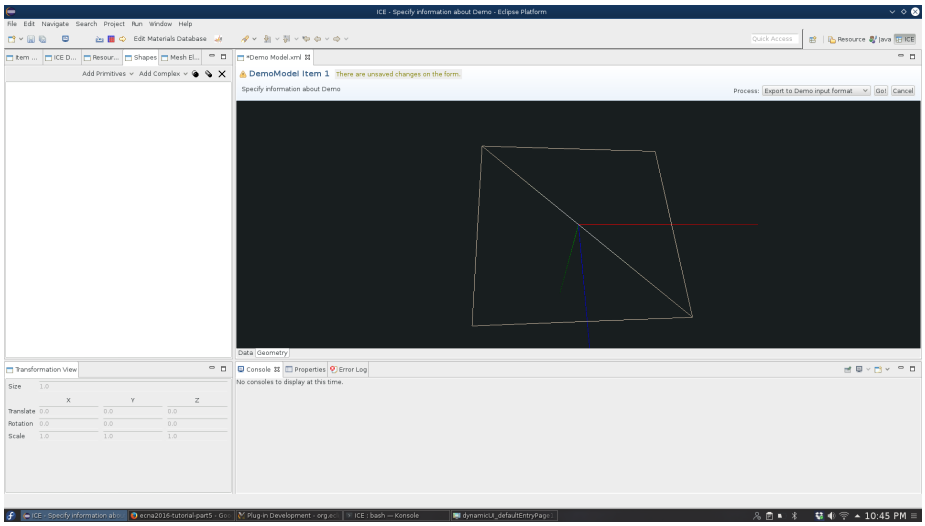


Figure 4.2: Default Geometry Page

- org.eclipse.e4.ui.model.application
- org.eclipse.e4.ui.model.application.ui

Create a class that implements IPageProvider and copy the following code into it:

```
public class DemoGeometryPageProvider implements IPageProvider {

    @Override
    public String getName() {
        // TODO Auto-generated method stub
        return "demo";
    }

    @Override
    public ArrayList<IFormPage> getPages(FormEditor formEditor,
            ArrayList<Component> components) {

        ArrayList<IFormPage> pages = new ArrayList<IFormPage>();
        // Get the GeometryComponent and create the GeometryPage.
        if (!(components.isEmpty())) {
            GeometryComponent geometryComponent =
                (GeometryComponent) (components
                    .get(0));
```

```
            if (geometryComponent != null) {

                // Make the GeometryPage
                DemoGeometryPage geometryPage = new
                    DemoGeometryPage(formEditor,
                        "GPid", geometryComponent.getName());

                // No need to set the geometry component for the
                    demo, but
                // something like would be necessary in a real
                    application.
                // geometryPage.setGeometry(geometryComponent);

                // Add the page
                pages.add(geometryPage);
            }

        }

        return pages;
    }

}
```

This class will provide your page to the platform. We need a second class that actually draws the content on the Geometry page. We will create a third class here, for convenience, that inherits from ICEs ICEFormPage base class to act as the GeometryPage. We dont actually need to show the content of the Geometry component for now because we just want to show that we can change the page. So, create your class and copy the following code into it to just show a label in place of the original 3D editor:

```
public class DemoGeometryPage extends ICEFormPage {

    public DemoGeometryPage(FormEditor editor, String id, String
        title) {
        super(editor, id, title);
        // TODO Auto-generated constructor stub
    }

    /**
     * <p>
     * Provides the page with the geometryApplication's
         information to display
     * geometry.
```

```
     * </p>
     *
     * @param managedForm
     *          the managed form that handles the page
     */
    @Override
    public void createFormContent(IManagedForm managedForm) {

        // Local Declarations
        final ScrolledForm form = managedForm.getForm();
        GridLayout layout = new GridLayout();

        // Setup the layout and layout data
        layout.numColumns = 1;
        form.getBody().setLayoutData(
                new GridData(SWT.FILL, SWT.FILL, true, true, 1,
                    1));
        form.getBody().setLayout(new FillLayout());

        // Just create some text and say hello
        Label geometryText = new Label(form.getBody(), SWT.FLAT);
        geometryText.setText("Draw something based on the
            geometry.");

        return;

    }

}
```

4.1.4 Create a new IEntryComposite

Individual Entrys that ICE draws can be extended in the same way. First, create a subclass of AbstractEntryComposite that implements render and copy the following code into it:

```
public class DemoEntryComposite extends AbstractEntryComposite {

    public DemoEntryComposite(Composite parent, IEntry refEntry,
        int style) {
        super(parent, refEntry, style);
    }
```

```
    @Override
    public void render() {

        Button button = new Button(this, SWT.PUSH);
        button.setText("My button");

        Label label = new Label(this, SWT.FLAT);
        label.setText(super.getEntry().getValue() + " in new
            Entry widget.");
        setLayout(new FillLayout());
        this.layout();

        return;
    }

}
```

Next, create a provider to publish your Entry Composite to the platform and copy the following code into it:

```
public class DemoEntryCompositeProvider implements
    IEntryCompositeProvider {

    @Override
    public String getName() {
        // TODO Auto-generated method stub
        return "demo-entry";
    }

    @Override
    public IEntryComposite getEntryComposite(Composite parent,
        IEntry entry,
            int style, FormToolkit toolKit) {
        // TODO Auto-generated method stub
        return new DemoEntryComposite(parent, entry, style);
    }

}
```

4.1.5 Publishing through the e4 extension point

The most common way to publish these extensions to the platform is to create an extension at the org.eclipse.e4.workbench.model extension point. You can do this by creating an executable processor. Create a new class and add the following code:

```
public class DemoWidgetsProcessor {

   /**
    * This operation executes the instructions required to
       register the demo
    * widgets with the e4 workbench.
    *
    * @param context
    *           The e4 context
    * @param app
    *           the model application
    */
   @Execute
   public void execute(IEclipseContext context, MApplication app)
       {

      // Add the geometry provider
      IPageProvider provider = ContextInjectionFactory
            .make(DemoGeometryPageProvider.class, context);
      context.set("demo-geometry", provider);

      // Add the EntryComposite provider
      IEntryCompositeProvider compProvider =
          ContextInjectionFactory
            .make(DemoEntryCompositeProvider.class, context);
      context.set("demo-entry", compProvider);

   }

}
```

Next, either create an extension point graphically or add the following code to your plugin.xml file:

```
   <extension
         id="org.eclipse.ice.demo.ui.processor"
```

```
        name="ICE Demo UI Processor"
        point="org.eclipse.e4.workbench.model">
     <processor
           apply="always"
           beforefragment="true"
           class="org.eclipse.ice.demo.ui.DemoWidgetsProcessor">
     </processor>
  </extension>
```

Your extensions will now be available in the workbench. Your entry composite is should look like fig. 4.3.

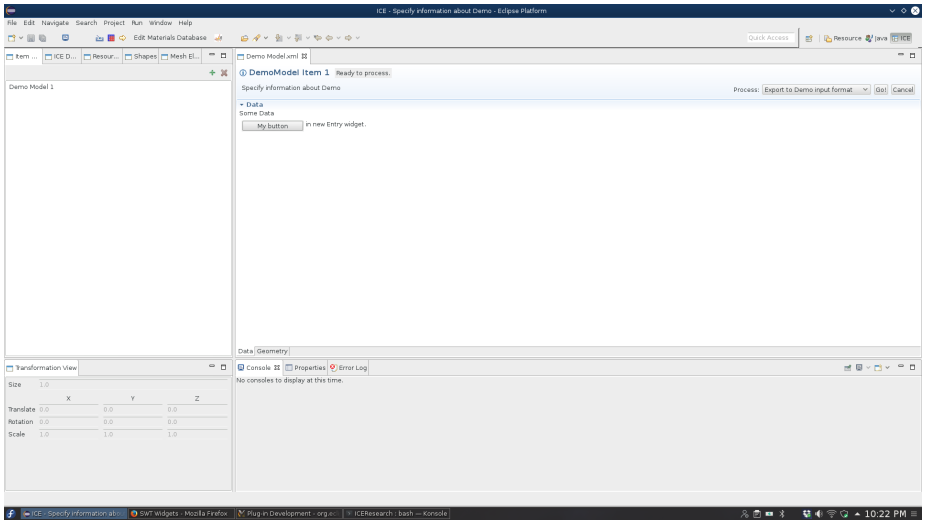


Figure 4.3: Updated Entry Widget

You geometry component should look like fig. 4.4.

4.1.6 Publishing through Context Functions

Make sure to check `Activate this plug-in when one of its classes loaded.` in your MANIFEST.mf file.

You can alternatively publish your new extensions to the OSGI framework so that they can be dynamically injected into the ICE's UI code. These extensions are registered as standalone services that are dynamically located at runtime based on the context name that you specified in your data structures. The benefit of this method over the extension point is that it can be used to

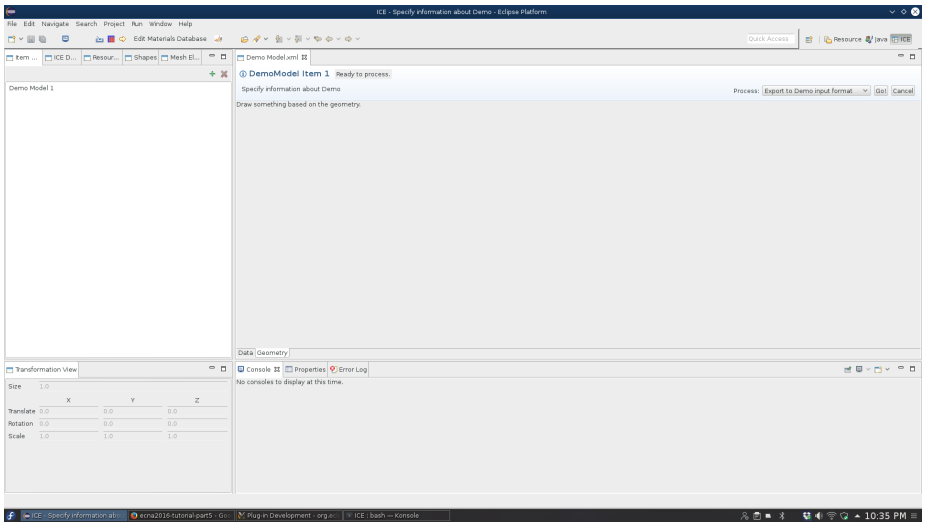


Figure 4.4: The updated Geometry page.

dynamically update based on the *present* context of the UI, not the initial context. This means that in addition to the type of data structure involved the UI can be tailored based on the particular area of the workbench where it would be drawn and the current runtime state.

Create a new context function for your IEntryComposite. You should create a subclass of ContextFunction with the following code:

```
public class DemoEntryCompositeContextFunction extends
    ContextFunction {

    @Override
    public Object compute(IEclipseContext context, String
        contextKey) {
        IEntryCompositeProvider provider = ContextInjectionFactory
                .make(DemoEntryCompositeProvider.class, context);
        // add the new object to the application context
        MApplication application =
            context.get(MApplication.class);
        IEclipseContext ctx = application.getContext();
        ctx.set(IEntryCompositeProvider.class, provider);
        return provider;
    }

}
```

Create an OSGI component with the following contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
name="org.eclipse.ice.demo.entry"> <implementation
   class="org.ecli
       pse.ice.demo.ui.DemoEntryCompositeContextFunction"/>
   <property name="service.context.key" type="String"
       value="demo-entry"/>
   <service> <provide
   interface="org.eclipse.e4.core.contexts.IContextFunction"/>
       </service>
</scr:component>
```

Now, create a context function for your Geometry Page. You should create a subclass of ContextFunction with the following code:

```
public class DemoGeometryPageContextFunction extends
    ContextFunction {

    @Override
    public Object compute(IEclipseContext context, String
        contextKey) {
        IPageProvider provider = ContextInjectionFactory
                .make(DemoGeometryPageProvider.class, context);
        // add the new object to the application context
        MApplication application =
            context.get(MApplication.class);
        IEclipseContext ctx = application.getContext();
        ctx.set(IPageProvider.class, provider);
        return provider;
    }

}
```

and an OSGI component with the following contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
name="org.eclipse.ice.demo.geometry.page"> <implementation
class="org.eclipse.ice.demo.ui.DemoGeometryPageContextFunction"/>
    <property
name="service.context.key" type="String" value="demo-geometry"/>
    <service>
```

```
    <provide
        interface="org.eclipse.e4.core.contexts.IContextFunction"/>
  </service>
</scr:component>
```

4.1.7 Replacing the whole Page Factory

If you want to replace the way that all pages in ICE are drawn, you can replace the entire Page Factory. Create a new class in your package that inherits from DefaultPageFactory, which will save you some work over implementing an entire new factory from scratch. See fig. 4.5.

Copy the following code into your new subclass:

```
public class DemoGeometryPageFactory extends DefaultPageFactory {

    @Override
    public ArrayList<IFormPage>
        getGeometryComponentPages(FormEditor editor,
            ArrayList<Component> components) {
        DemoGeometryPageProvider pageProvider = new
            DemoGeometryPageProvider();
        return pageProvider.getPages(editor, components);
    }
}
```

This will now build on your previous DemoGeometryPageProvider to provide it to the framework through a factory. You can override other operations to provide access to other custom pages. This method may be more efficient than creating separate page extension for each page type. To publish this to the platform, create a Context Function with the following code:

```
public class DemoGeometryPageFactoryContextFunction extends
    ContextFunction {

    @Override
    public Object compute(IEclipseContext context, String
        contextKey) {
        IPageFactory factory = ContextInjectionFactory
                .make(DemoGeometryPageFactory.class, context);
        // add the new object to the application context
        MApplication application =
            context.get(MApplication.class);
```

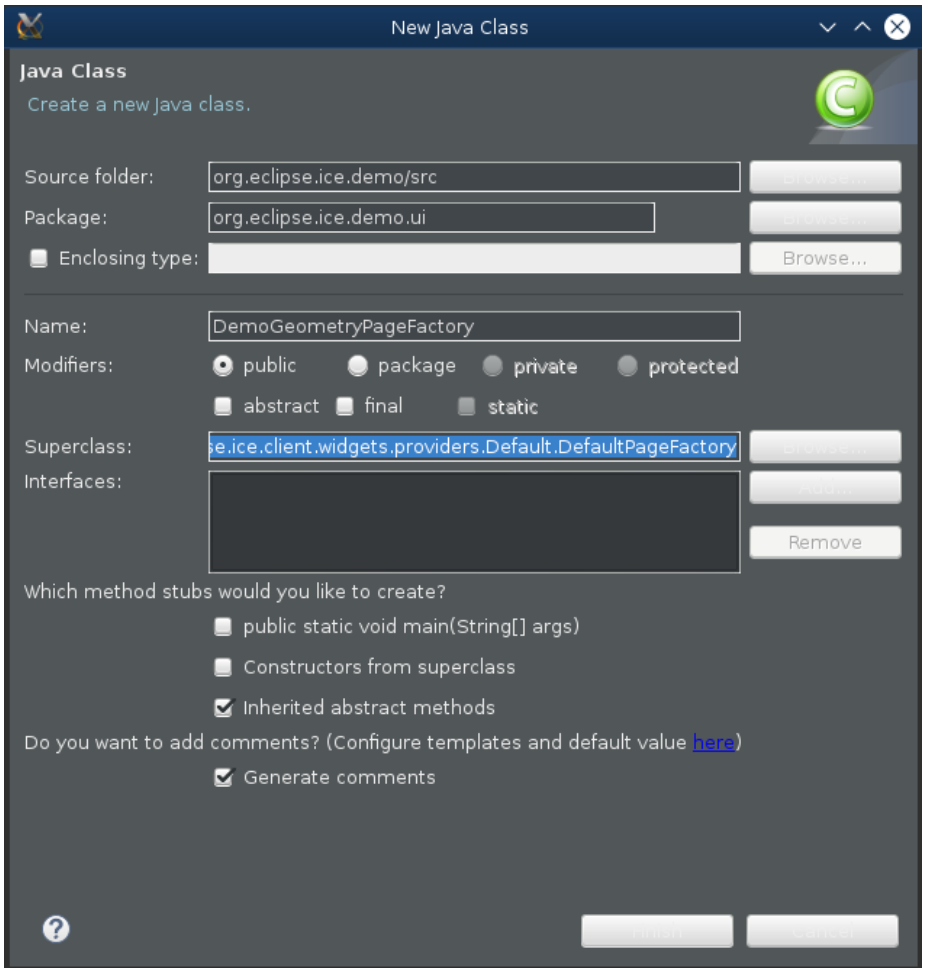


Figure 4.5: Generating the Page Factory class.

```
        IEclipseContext ctx = application.getContext();
        ctx.set(IPageFactory.class, factory);
        return factory;
    }

}
```

Your OSGI component should be published as such:

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
```

```
name="org.eclipse.ice.demo.factory"> <implementation
class="org.eclipse.ice.demo.ui.DemoGeometryPageFactoryContextFunction"/>
<property name="service.context.key" type="String"
    value="demo"/> <service>
      <provide
          interface="org.eclipse.e4.core.contexts.IContextFunction"/>
   </service>
</scr:component>
```

and your MANIFEST.mf file should contain:

```
Service-Component: OSGI-INF/*.xml
```

Chapter 5

Scripting

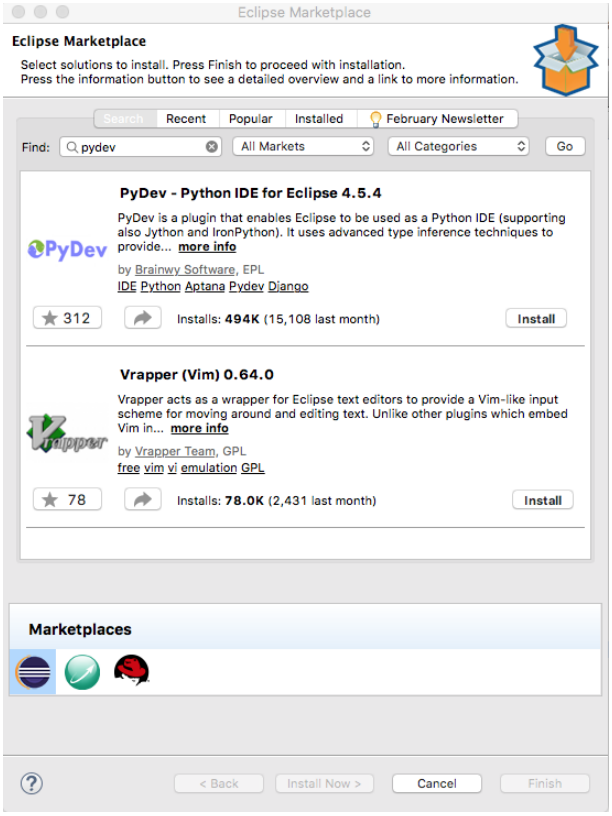
5.1 Scripting with EASE

In addition to interacting with tools via the ICE user interface, ICE also provides a scripting framework based on the Eclipse Advanced Scripting Environment (EASE).

5.1.1 PyDev Installation (optional)

Although it is possible to edit Python scripts in Eclipse using the default text editor, however it is much more productive to use the PyDev Eclipse development environment if you are planning to do a lot of script development. In addition to the usual syntax coloring and other advanced editing features you'd expect in Eclipse, PyDev also provides the ability to run and debug Python programs from within the Eclipse environment.

PyDev can be easily installed from using the Eclipse Marketplace client as follows. From the ICE menu bar, select `Help` → `Eclipse Marketplace...`, type "pydev" in the Find field, then click the `Go` button. After a few seconds, you should see an entry for PyDev as shown in image below. Simply click the `Install` button and follow the prompts to install the feature. Once Eclipse has been restarted, any Python scripts ending in ".py" will be recognized by PyDev and opened in the Python editor by default.

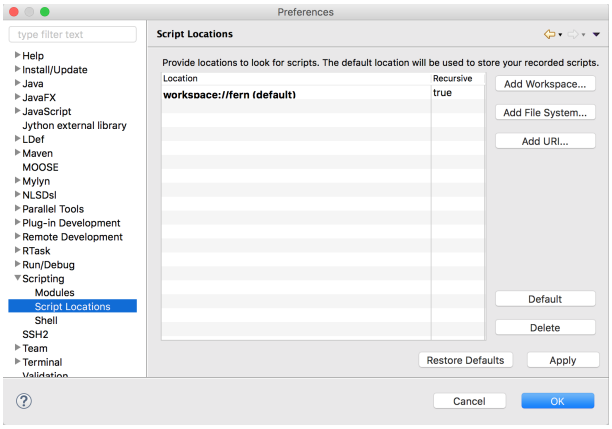


5.1.2 EASE Configuration

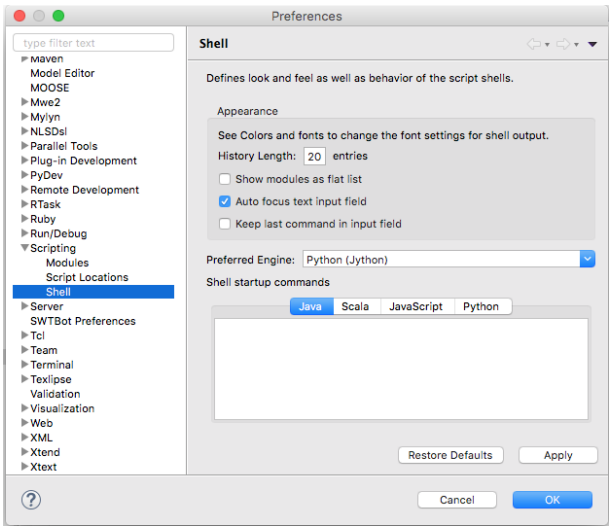
There are a number of configuration settings that can be used to customize the behavior of EASE. These configuration settings are accessed via the `Scripting` preference page in the Eclipse preferences. To open the `Scripting` preference page, select `Window` → `Preferences...` in the ICE menu bar. (On Mac OS X, Preferences...is instead located under Eclipse ICE in the menu bar.) Open the `Scripting` tree item on the left side of the preferences window and you will see the different preferences that can be configured.

EASE works best if you keep your scripts in one or more projects in your workspace. You can then associate these *Script Locations* so that EASE knows they are used for managing your scripts. You can either use an existing project, or create a new project for the scripts. In this tutorial, we're going to use the existing `fern` project to store our scripts, but you could just as easily create a new one. To configure the projects, select the `Script Locations` preference item to reveal the dialog shown below. You can then use the `Add Workspace...` button to select one or more projects from the workspace. For this tutorial,

choose the `fern` project.



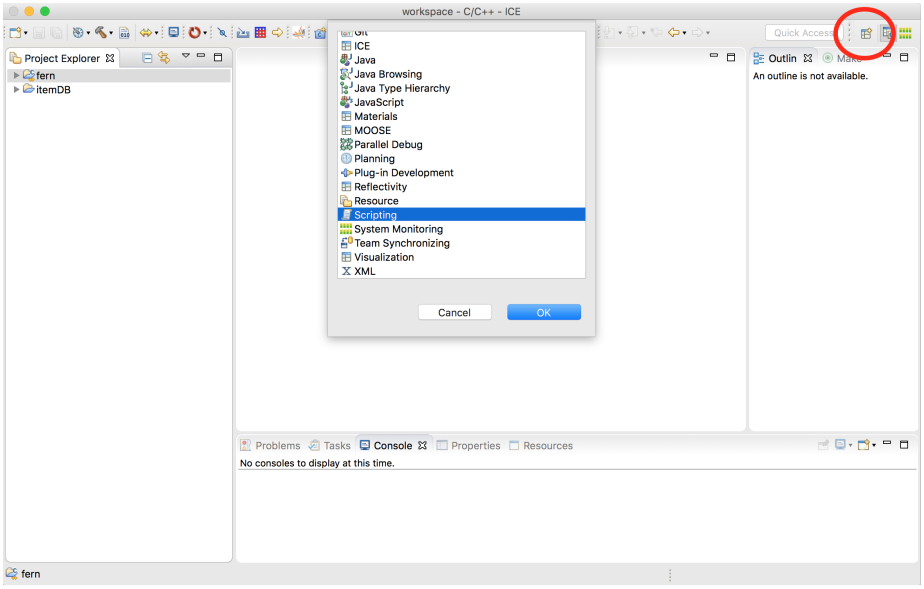
By default, EASE is configured to use the JavaScript (Rhino) engine. Since this tutorial assumes that the preferred environment is Python, we recommend changing this default. To set the script engine default, select the `Shell` preference item. Next, select `Python (Jython)` from the `Preferred Engine:` dropdown as shown below, then click on `OK`.



5.1.3 Creating and Running Scripts

There is nothing special about creating and running EASE scripts. They can be created in a variety of ways using the development tools available in ICE, and then run later when needed. For this tutorial, we will be creating the scripts using PyDev (or any text editor) and running them directly using the `Run As` → `EASE Script` context menu.

EASE also provides a perspective for creating, managing, debugging, and running scripts called the `Scripting` perspective. This perspective contains views for running script commands interactively, and for exploring script modules. To switch to the `Scripting` perspective use the `Perspective Switcher` icon or select the `Window` → `Perspective` → `Open Perspective` → `Other...` menu, then select `Scripting` from the list. You should see the perspective shown below.

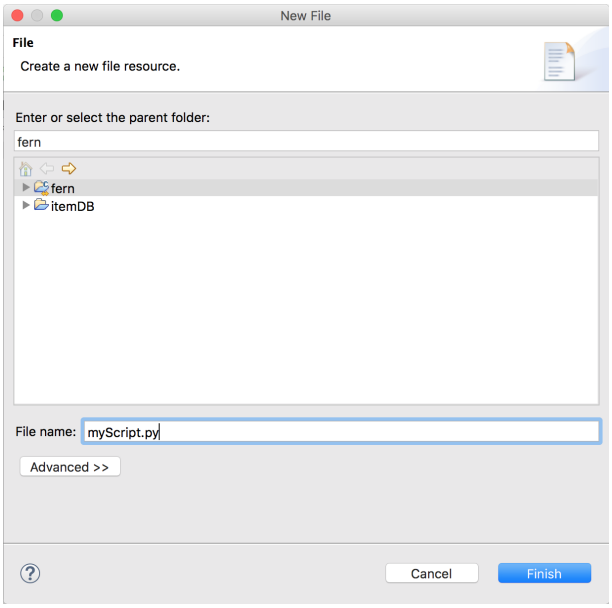


EASE hides many of the details that would normally be required to manipulate Java objects and perform actions in the Eclipse IDE. It does this by encapsulating typical actions into simple script commands that can be easily invoked from scripts that you write. These commands are collected together into "modules". You can see which modules are available, and the commands that they contain, using the `Modules Explorer` view. This view is visible in the top right corner of the `Scripting` perspective.

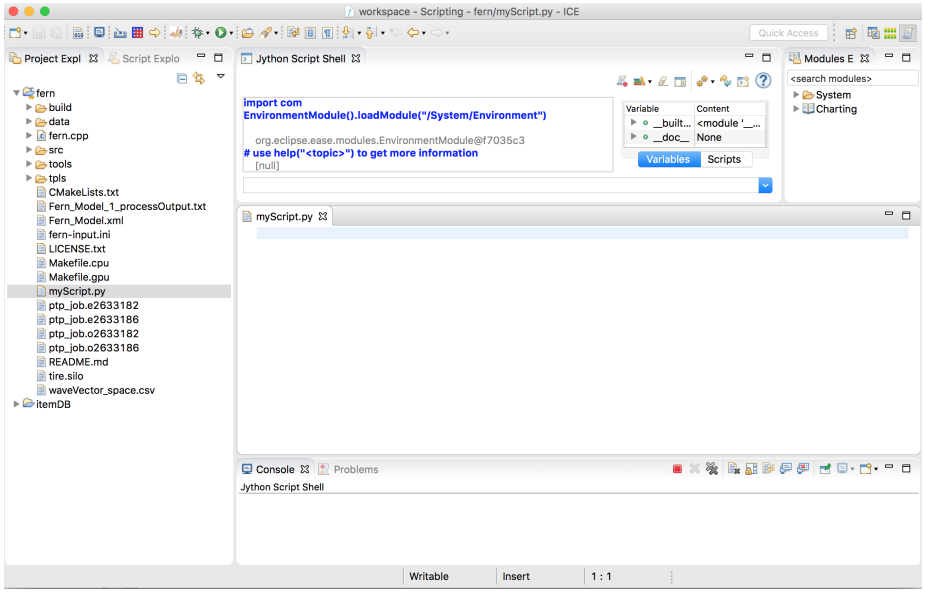
5.1.4 Creating a Python Script

The easiest way to create a Python script is to simply create a text file ending in ".py" in an existing project. If your scripts will be used with stand-alone Python programs, you can use PyDev to create a Python project, but in general, any kind of project will suffice.

For this tutorial, we're going to use the `fern` project that has already been created, and should be visible in the `Project Explorer` view. You can now create a Python file in this folder by right clicking on the folder and selecting `New` → `File` from the context menu. This will display the `New File` dialog as shown below.



At this point all that remains to be done is enter the name of the file in the `File name:` field and click on the `Finish` button. This will create the file and automatically open the PyDev editor (assuming you installed PyDev) or a simple text editor. You should see an editor view something like that shown below.



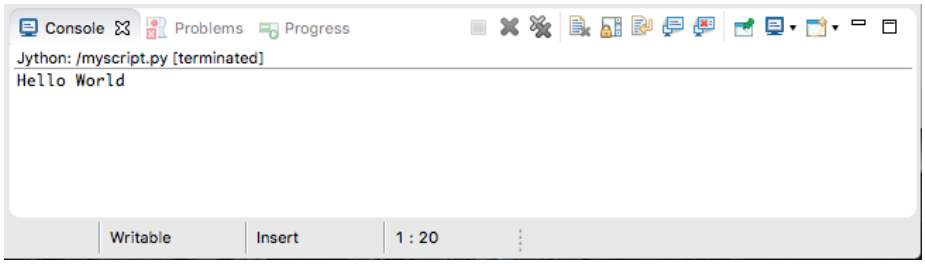
Since the `fern` project was selected as a scripting location previously, the new script file should also be visible in the `Script Explorer` view in the `Scripting` perspective. Make sure the `Scripting` perspective is selected for this view to be visible.

5.1.5 Writing a Python Script

Our first python script will just print "Hello World" on the console. Enter the following text in the editor and save the file:

```
print 'Hello World'
```

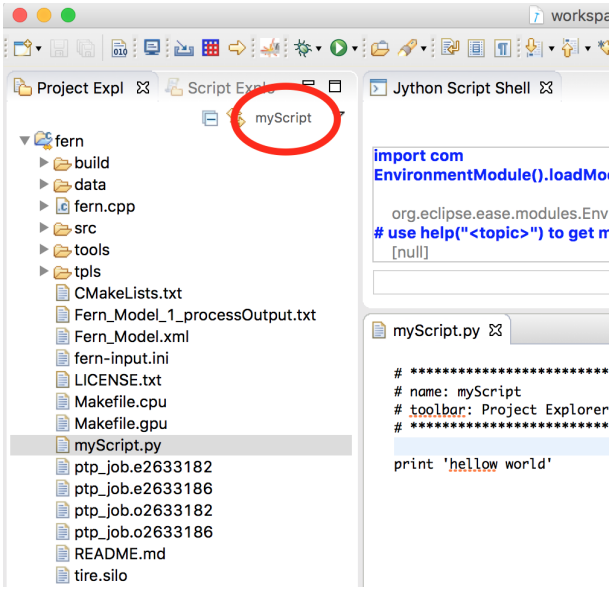
Once you have entered the Python script, is can be easily launched using the run button (green arrow) on the `Script Exporer` view. Simply select the script file you want to run and click on the run button. Any (textual) output generated by the script will be displayed in the Console view shown below.



Rather than using the run button, we would like to run this script using our own button on the `Project Explorer` view. To do this, add the following lines to the start of the script, then save the file:

```
# ********************************************************************
# name                : My Script
# toolbar             : Project Explorer
# ********************************************************************
```

As soon as you save the file, you should notice a couple of things. The name of the script file will change to `My Script` in the `Script Explorer` view, and if you switch to the `Project Explorer` view, you should see a `My Script` button in the view toolbar as shown below.



It is possible to include additional information, including a tooltip popup if desired:

```
# script-type          : Python
# description          : Start a file browser using current selection.
```

You can also modify the user interface by changing the button to an icon and adding a popup conext menu. This is accomplished by adding the following lines to the script:

```
# popup    : enableFor(org.eclipse.core.resources.IResource)
# image    : platform:/plugin/org.eclipse.ui.ide/icons/full/elcl16/configs.png
```

5.1.6 Interacting with ICE

In order to interact with Java classes in ICE from the Python script, we need to include additonal modules. In order to load modules, we use the `loadModule()` function in Python. The argument to this function is a string representation of the module path. For this tutorial we will need to load the `/System/Platform`, `/System/Resources`, and `/System/UI` modules.

In the `Modules Explorer` view, open the `System` folder, then drag the `Platform`, `Resources`, and `UI` items into the script file after the initial comments (you can also type these line in manually if you wish). This should insert the following lines into your script:

```
loadModule('/System/Platform');
loadModule('/System/Resources');
loadModule('/System/UI');
```

Once these module have been loaded, a number of additional functions become available. We want to obtain a reference to the core ICE service, which is used as the starting point for manipulating ICE models. We also want to obtain a reference to the `fern` project. This is done by adding the following lines:

```
coreService = getService(org.eclipse.ice.core.iCore.ICore)
project = getProject("fern")
```

Once a reference to the core services has been obtained, we can use this to obtain a reference to the Reflectivity Model and set some parameters. We supply the project to the *createItem()* method so that the generated files will be saved in the project folder. This is done by adding the following lines:

```
item = coreService.createItem("Reflectivity Model", project)
```

```
model = coreService.getItem(int(item))
component = model.getComponents().get(0)
entry = component.retrieveAllEntries().get(0)
entry.setValue("waveVector_space.csv")
```

Note that the `createItem()` method will return a string representing the number of that item, so `int()` is used to convert it to an integer, which is the argument expected by `getItem()`.

Now the only thing left to do it process the model. This is done using the core service `processItem()` as follows:

```
res = coreService.processItem(model.getId(), "Calculate Reflectivity", 1)
```

Finally, lets display one of the resulting CSV files if it was successful.

```
if str(res) == "Processed":
   output = getFile("reflectivityModel_%d_rfd.csv" % model.getId())
   openEditor(output)
```

Remember to save the editor using `Ctrl/Cmd-S` or the `File` → `Save` command from the ICE menu bar before running the script.

5.1.7 Using the Sample Scripts

We have provided a number of sample scripts to show how ICE can be scripted using EASE. These scripts are located in the `org.eclipse.ice.examples.reflectivity` package that is already loaded in your workspace. The scripts can also be obtained by cloning the ICE Git repository[1], then manually importing the `org.eclipse.ice.examples.reflectivity` package.

There are four sample scripts that demonstrate how a reflectivity model can be created, configured and executed. The scripts are described in more detail in the following sections.

createAndProcessPython.py

This is a simple script that demonstrates how to create a reflectivity model and process the model to obtain a result. The default model inputs are used for the computation.

[1] `http://github.com/eclipse/ice.git`

```
# ******************************************************************************
# Copyright (c) 2015 UT-Battelle, LLC.
# All rights reserved. This program and the accompanying materials
# are made available under the terms of the Eclipse Public License v1.0
# which accompanies this distribution, and is available at
# http://www.eclipse.org/legal/epl-v10.html
#
# Contributors:
#   Initial API and implementation and/or initial documentation - Kasper
# Gammeltoft.
#
# This is an example script designed to show how to use ease with ICE. It
# creates a new Reflectivity Model and processes it, using the default mock
# data and inputs.
# ******************************************************************************

# Load the Platform module for accessing OSGi services
loadModule('/System/Platform')

# Get the core service from ICE for creating and accessing objects.
coreService = getService(org.eclipse.ice.core.iCore.ICore);

# Create the reflectivity model to be used and get its reference. The create item
# method will return a string representing the number of that item, so use int() to
# convert it to an integer.
reflectModel = coreService.getItem(int(coreService.createItem("Reflectivity Model")))

# This is usually where you would do your own customization and automation regarding
# the reflectivity model you just created. Maybe change the layers, or do some custom
# calculations.


# Finally process the model to get the results.
coreService.processItem(reflectModel.getId(), "Calculate Reflectivity", 1);
```

createAndEditPython.py

This script extends the `createAndProcessPython.py` script by editing the input to the model programmatically. The model is then processed to obtain the results.

```
# ******************************************************************************
# Copyright (c) 2015 UT-Battelle, LLC.
# All rights reserved. This program and the accompanying materials
# are made available under the terms of the Eclipse Public License v1.0
# which accompanies this distribution, and is available at
# http://www.eclipse.org/legal/epl-v10.html
#
# Contributors:
#   Initial API and implementation and/or initial documentation - Kasper
# Gammeltoft.
#
# This is an example script designed to show how to use ease with ICE. It
# creates a new Reflectivity Model and processes it, but also edits the input
# to the table beforehand.
# ******************************************************************************

# Load the Platform module for accessing OSGi services
loadModule('/System/Platform')
```

```
# Get the core service from ICE for creating and accessing objects.
coreService = getService(org.eclipse.ice.core.iCore.ICore);

# Create the reflectivity model to be used and get its reference. The create item
# method will return a string representing the number of that item, so use int() to
# convert it to an integer.
reflectModel = coreService.getItem(int(coreService.createItem("Reflectivity Model")))

# Gets the list component used as the data for the table (is on tab 2)
listComp = reflectModel.getComponent(2)

# Gets the third material and sets its thickness to 400
mat1 = listComp.get(2)
mat1.setProperty("Thickness (A)", 400)

# Get the total thickness and set the second material's thickness to depend
# on the thicknesses of the other materials
totThickness = 0
for i in xrange(0, listComp.size() - 1):
    if(i != 1):
        totThickness += listComp.get(i).getProperty("Thickness (A)")

# Set the thickness of the second material so that the total sums to 1000 (A)
listComp.get(1).setProperty("Thickness (A)", 1000-totThickness);


# Finally process the model to get the results.
coreService.processItem(reflectModel.getId(), "Calculate Reflectivity", 1);
```

iterateChangeParameterPython.py

This script demonstrates how to create multiple reflectivity models with varying input parameters. The models are created and processed sequentially.

```
# ************************************************************************
# Copyright (c) 2015 UT-Battelle, LLC.
# All rights reserved. This program and the accompanying materials
# are made available under the terms of the Eclipse Public License v1.0
# which accompanies this distribution, and is available at
# http://www.eclipse.org/legal/epl-v10.html
#
# Contributors:
#   Initial API and implementation and/or initial documentation - Kasper
# Gammeltoft.
#
# This is an example script designed to show how to use ease with ICE. It
# creates several new Reflectivity Models and changes the thickness parameter
# to show the effect that creates.
# ************************************************************************

# Load the Platform module for accessing OSGi services
loadModule('/System/Platform')

# Get the core service from ICE for creating and accessing objects.
coreService = getService(org.eclipse.ice.core.iCore.ICore);

# Set a initial value for the thickness of the nickel layer. This will be doubled
# for each iteration to show how this parameter effects the model
nickelThickness = 250;

for i in xrange(1, 5):
    # Create the reflectivity model to be used and get its reference. The create item
```

```
    # method will return a string representing the number of that item, so use int() to
    # convert it to an integer.
    reflectModel = coreService.getItem(int(coreService.createItem("Reflectivity Model")))

    # Get the nickel layer from the model. It should be in the list, which is component 2,
    # and it is the third layer in that list (which is item 2 as the list is zero based).
    listComp = reflectModel.getComponent(2);
    nickel = listComp.get(2);

    nickel.setProperty("Thickness (A)", nickelThickness);

    nickelThickness += 250;

    # Finally process the model to get the results.
    coreService.processItem(reflectModel.getId(), "Calculate Reflectivity", 1);
```

listFromScratchPython.py

This script demonstrates how to create a reflectivity model and programmably create and set up the layers in the model. The model is then process to obtain the results.

```
# ************************************************************************
# Copyright (c) 2015 UT-Battelle, LLC.
# All rights reserved. This program and the accompanying materials
# are made available under the terms of the Eclipse Public License v1.0
# which accompanies this distribution, and is available at
# http://www.eclipse.org/legal/epl-v10.html
#
# Contributors:
#   Initial API and implementation and/or initial documentation - Kasper
# Gammeltoft.
#
# This is an example script designed to show how to use ease with ICE. It
# creates a new Reflectivity Model and shows how to customize and build up
# the layers in the model from scratch.
# ************************************************************************

# Needed imports from ICE
from org.eclipse.ice.datastructures.form import Material

# Load the Platform module for accessing OSGi services
loadModule('/System/Platform')

# Get the core service from ICE for creating and accessing objects.
coreService = getService(org.eclipse.ice.core.iCore.ICore);

# Create the reflectivity model to be used and get its reference. The create item
# method will return a string representing the number of that item, so use int() to
# convert it to an integer.
reflectModel = coreService.getItem(int(coreService.createItem("Reflectivity Model")))

# Gets the list component used as the data for the table (is on tab 2)
listComp = reflectModel.getComponent(2)

# Now we want to build up the list from our own data, so we can do that here.

# The first step would be to clear the list so that we can start adding to it. Clearing
# the list requires the locks as multiple operations are happening and we need to
# protect the list from multiple threads trying to access it at the same time.
listComp.getReadWriteLock().writeLock().lock()
```

```
listComp.clear()
listComp.getReadWriteLock().writeLock().unlock()

# Create the layer of air
air = Material()
air.setName("Air")
air.setProperty("Material ID", 1)
air.setProperty("Thickness (A)", 200)
air.setProperty("Roughness (A)", 0)
air.setProperty(Material.SCAT_LENGTH_DENSITY, 0)
air.setProperty(Material.MASS_ABS_COHERENT, 0)
air.setProperty(Material.MASS_ABS_INCOHERENT, 0)

# Create the Aluminum Oxide layer
AlOx = Material()
AlOx.setName("AlOx")
AlOx.setProperty("Material ID", 2)
AlOx.setProperty("Thickness (A)", 25)
AlOx.setProperty("Roughness (A)", 10.2)
AlOx.setProperty(Material.SCAT_LENGTH_DENSITY, 1.436e-6)
AlOx.setProperty(Material.MASS_ABS_COHERENT, 6.125e-11)
AlOx.setProperty(Material.MASS_ABS_INCOHERENT, 4.47e-12)

# Create the Aluminum layer
Al = Material()
Al.setName("Al")
Al.setProperty("Material ID", 3)
Al.setProperty("Thickness (A)", 500)
Al.setProperty("Roughness (A)", 11.4)
Al.setProperty(Material.SCAT_LENGTH_DENSITY, 2.078e-6)
Al.setProperty(Material.MASS_ABS_COHERENT, 2.87e-13)
Al.setProperty(Material.MASS_ABS_INCOHERENT, 1.83e-12)

# Create the Aluminum Silicate layer
AlSiOx = Material()
AlSiOx.setName("AlSiOx")
AlSiOx.setProperty("Material ID", 4)
AlSiOx.setProperty("Thickness (A)", 10)
AlSiOx.setProperty("Roughness (A)", 17.2)
AlSiOx.setProperty(Material.SCAT_LENGTH_DENSITY, 1.489e-6)
AlSiOx.setProperty(Material.MASS_ABS_COHERENT, 8.609e-9)
AlSiOx.setProperty(Material.MASS_ABS_INCOHERENT, 6.307e-10)

# Create the Silicon layer
Si = Material()
Si.setName("Si")
Si.setProperty("Material ID", 5)
Si.setProperty("Thickness (A)", 100)
Si.setProperty("Roughness (A)", 17.5)
Si.setProperty(Material.SCAT_LENGTH_DENSITY, 2.07e-6)
Si.setProperty(Material.MASS_ABS_COHERENT, 4.7498e-11)
Si.setProperty(Material.MASS_ABS_INCOHERENT, 1.9977e-12)

# Add all of the materials back to the list (in top to bottom order)
listComp.getReadWriteLock().writeLock().lock()
listComp.add(air);
listComp.add(AlOx);
listComp.add(Al);
listComp.add(AlSiOx);
listComp.add(Si);
listComp.getReadWriteLock().writeLock().unlock()

# Finally process the model to get the results.
coreService.processItem(reflectModel.getId(), "Calculate Reflectivity", 1);
```

Chapter 6

The Eclipse ICE Developer Menu

Overview

The Eclipse Integrated Computational Environment (ICE) has had a great track record of providing a comprehensive environment for general scientific computing. Tasks such as model input generation, local and remote simulation execution, and post-simulation data analysis and visualization are all very well supported in the application. These tasks take care of the majority of needs for *using* general scientific computing codes, but what about *developing* those applications to begin with? Is there any way ICE can be extended to provide support for the development of science codes?

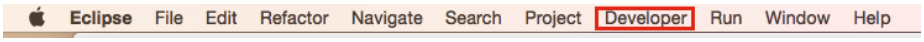


Figure 6.1: The ICE Developer Menu

The answer is yes! In 2015 ICE was extended to provide support for scientific application *development* through a custom, extensible **Developer** top-level menu. This menu is shown in Figure 6.1 and provides custom actions that enable efficient scientific application development for both novice and expert users of a given science code.

The Developer Menu is completely customizable through Eclipse Extension Points. Specifically, ICE exposes a new extension point: *org.eclipse.ice.developer.code*. This extension point provides the means to specify details about a scientific code: its name, category (Framework, Nuclear, Other, etc...), and where its repository is hosted, just to name a few. This point also enables the addition of *com-*

mands that point to some custom subclass of *org.eclipse.core.commands.AbstractHandler* that performs some task related to the development of the code.

With these extensions exposed as part of an ICE product execution, ICE handles all the complexity of picking them up and populating the Developer Menu dynamically at runtime. This feature is shown in Figure 6.2, with an ICE category for ICE development, and another for scientific frameworks. This process is exactly how ICE developers work on ICE itself - by pulling down the ICE binary and leveraging the Developer Menu to clone and build ICE, all within ICE. ICE also provides these hooks for other scientific codes, like the Multiphysics Object Oriented Simulation Environment (MOOSE) for general finite-element simulations. ICE provides a hook for cloning MOOSE and for forking a templated repository for MOOSE Application development.

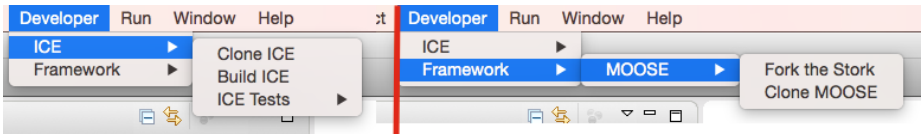


Figure 6.2: The Developer Menu ICE and MOOSE Actions

Extending the Developer Menu is easy, and relies on simply creating a new plugin and exposing a new extension. Let's see how to do this task in detail:

Extending the ICE Developer Menu

For this tutorial, we are going to create a hook into the Developer menu to clone a scientific code called Fern. Fern is an application that provides an efficient nuclear reaction network solver. It is hosted at `https://github.com/jayjaybillings/fern`.

To get started, we need to download the ICE plugins to our workspace, which we can actually do through the Developer menu (pretty cool to use the Developer menu to extend the Developer menu!). Click Developer > ICE > Clone ICE to get all of ICE's plugins into the workspace. With ICE cloned to the workspace, we can now begin to extend the Developer menu. First, we will need to create a new plugin.

Create a New Plugin Project

Creating a new plugin for an extension to the ICE Developer menu is simple, just click File > New > Plugin Project (or Other, then select Plugin Project). When the wizard opens, name your new plugin with something similar to Figure 6.3

and deselect the `Generate an activatitor` option. On the next page, simply uncheck the create a plugin from template button and click Finish.

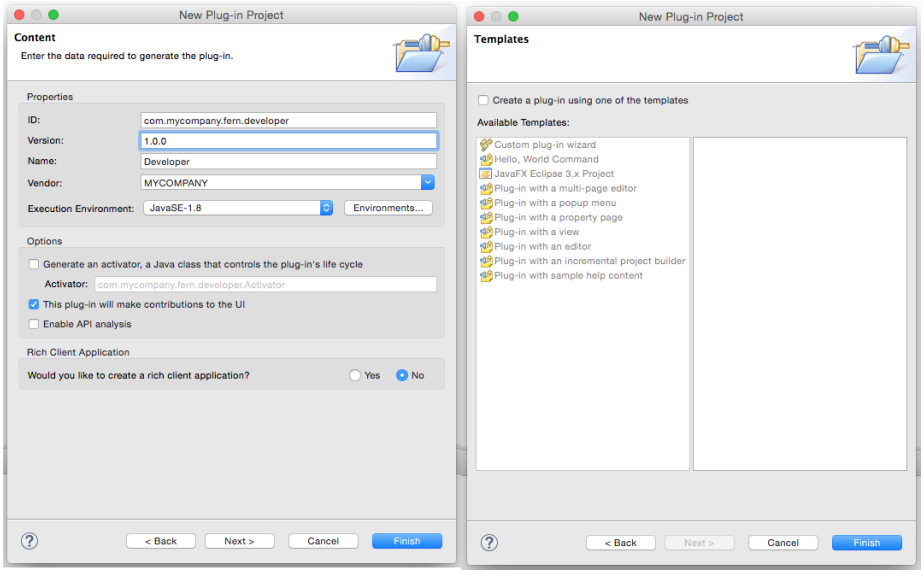


Figure 6.3: Creating a new Plugin Project

When the MANIFEST.MF file editor opens up, add the *org.eclipse.ice.developer* plugin as a Required Plugin on the Dependencies tab.

Create a New ICE Developer Extension

Now let's create a new Extension to connect ICE and the Developer Menu with a Clone Fern action. To do so, go to the Extensions tab of the plugin MANIFEST file and click add. ICE provides an org.eclipse.ice.developer.code extension point that lets users define various details about their codes.

You will then be presented with the view in Figure 6.4. Enter a descriptive ID and Name for this extension and click Save.

To define your scientific application, right click on the org.eclipse.ice.developer.code extension in the All Extensions section and select New > Code in the context menu. This action will present you with the view in Figure 6.5 where you can select the code category, code display name, the repository URL, and the branch you would like to work with.

To create a new developer action for your code, right click the code element of the extension tree and select New > Command. You will then be presented

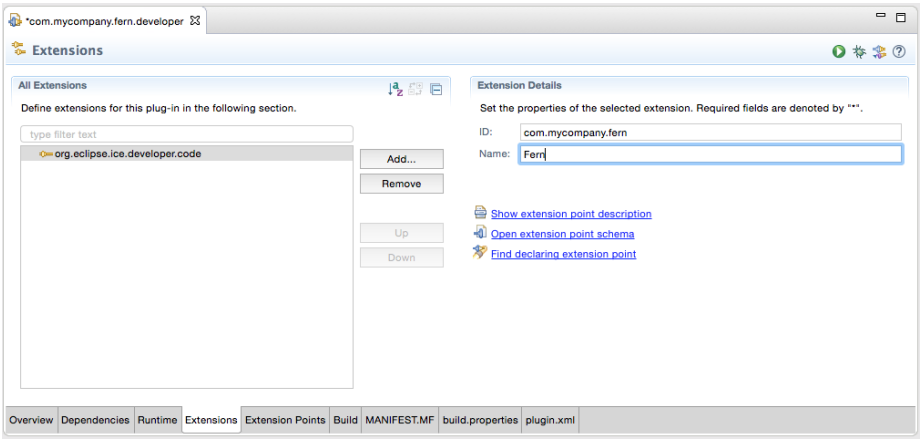


Figure 6.4: Configure the ID and Name of this Code Extension.

with the view at the top of Figure 6.5 where you can input the name of this action and the AbstractHandler subclass that performs the action. ICE provides a default GitCloneHandler that you may select. For this tutorial, select that and click Save. You should now have a view similar to the bottom of Figure 6.5.

Setup ICE to Run with the New Plugin

To see this new Developer command in action, we need to launch a new instance of ICE. This can be done by opening the Run Configurations Wizard in Run > Run Configurations. Under the Eclipse Applications element in the tree on the left, select the ICE launch configuration for your OS. Open the Plugins tab and add your new developer plugin (see Figure 6.6) by enabling it. Then, click Run to launch ICE with your developer plugin. With ICE running, navigate to the Developer menu and select Nuclear > Fern > Clone Fern (see Figure 6.7).

This will kick off the action you specified to clone the Fern repository and pull in any Eclipse projects to the workspace, as shown in Figure

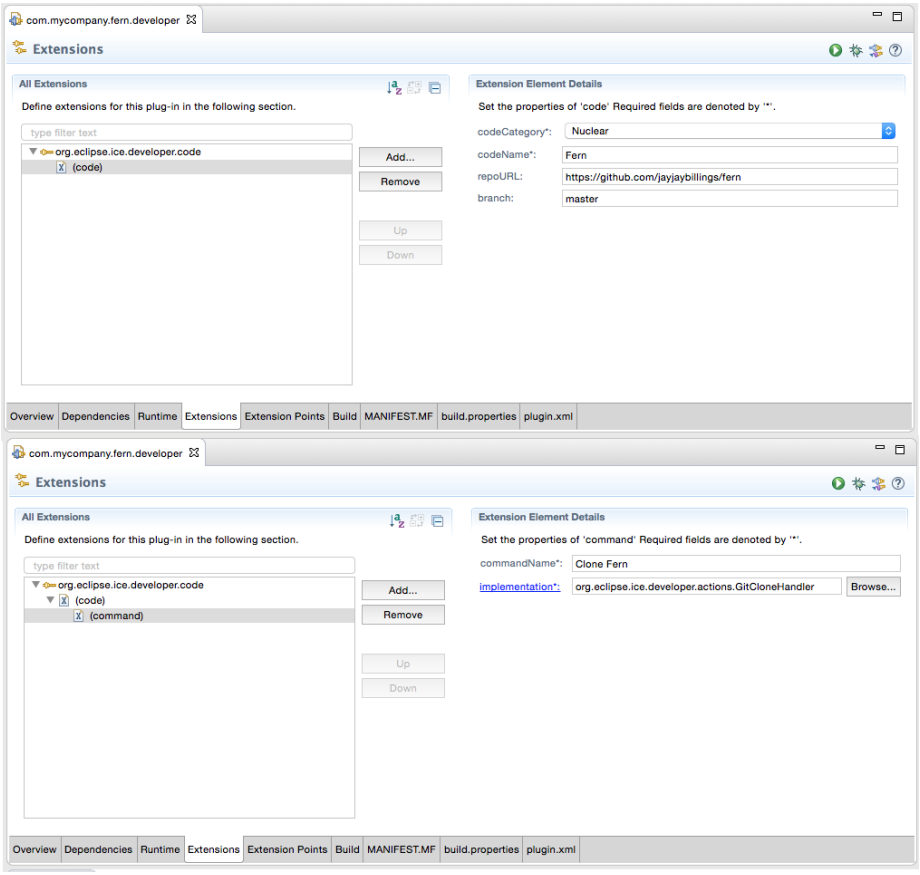


Figure 6.5: Create a new Code description for this extension.

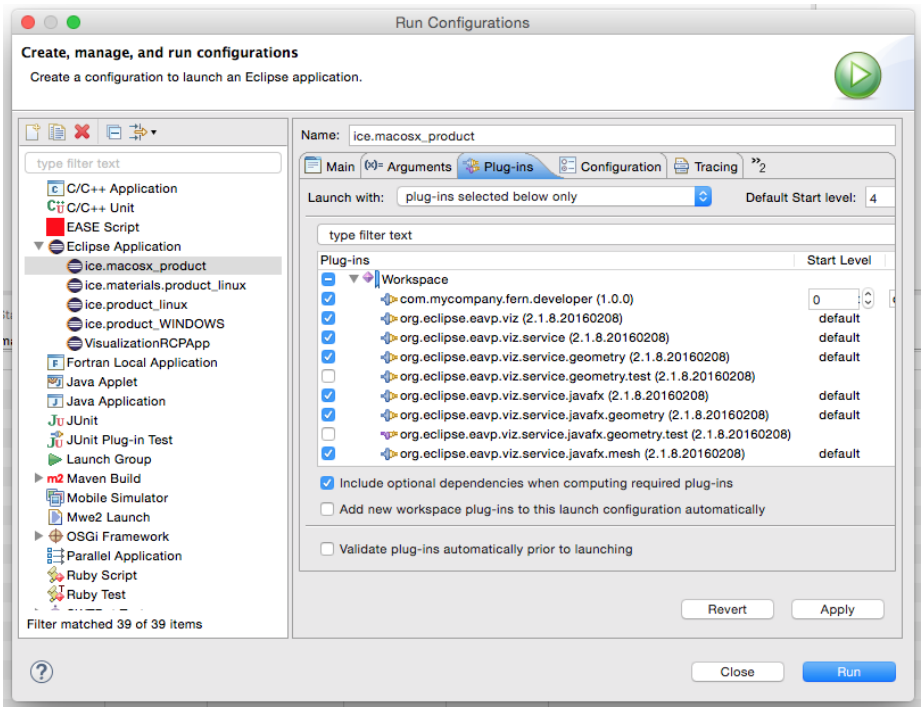


Figure 6.6: Add your plugin to the run configuration.

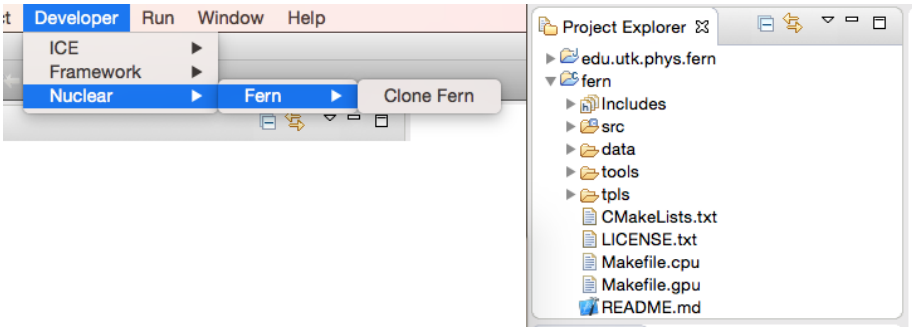


Figure 6.7: Use the Developer Menu to Clone ICE.

Chapter 7

Geometry Editor

7.1 Geometry Editor

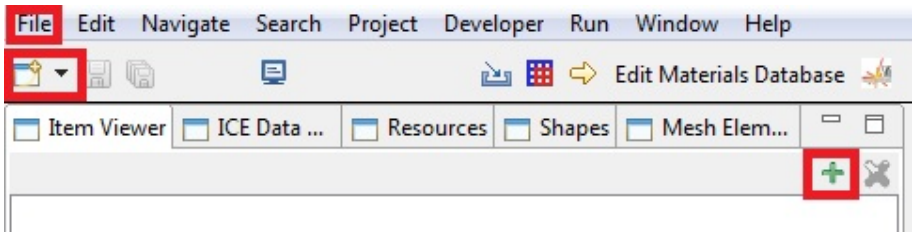
This article is designed to outline the basic controls of ICE's Geometry Editor.

7.1.1 Getting Started

Once ICE is installed on your computer, there are no further dependencies or preparation required to use the Geometry Editor.

7.1.2 Opening a Geometry Editor

To open a Geometry Editor in ICE, you have three options:



Top to bottom, they are:

1) Click the File menu, then New, then Other... and select the Create Item Wizard in the new dialog and press Next. Then, select Geometry Editor from the list and press Finish.

2) Click the New button. Select Create Item Wizard in the new dialog and press Next. Then, select Geometry Editor from the list and press Finish.

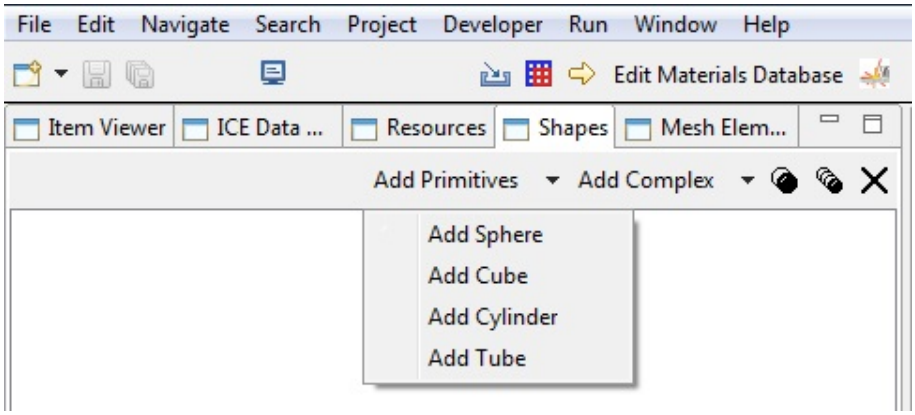
3) Enter the ICE Perspective by clicking the Open Perspective button in the upper right corner of the screen, select ICE from the dialog that pops up, and click OK. Afterwards, click the Create an Item button, select Geometry Editor, and click OK.

7.1.3 Working with the Geometry Editor

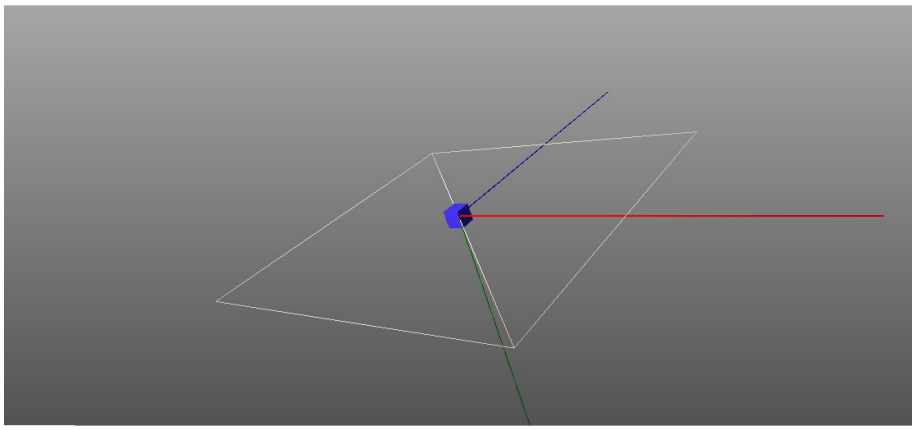
Camera

You can change the perspective of the camera by clicking and dragging inside the Geometry Editor. You can see this initially by the way the three axes and the reference plane move as you rotate the camera about the origin. If you hold shift while dragging, you will instead move the central point the camera is focused on. If you hold ctrl, you can drag the mouse up or down to zoom in or out, respectively. Scrolling the mouse wheel offers another way to zoom the camera.

Primitive Shapes



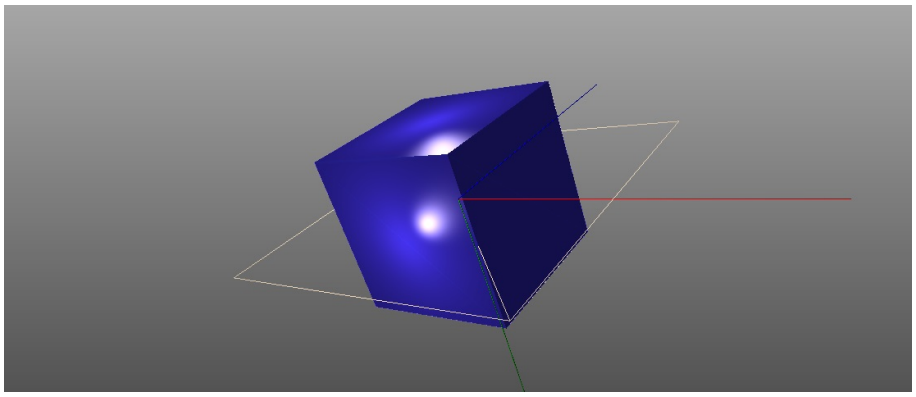
Simple shapes can be added with the Add Primitives dropdown menu in the Shapes view. Simply select an object from the menu to add it to the scene.



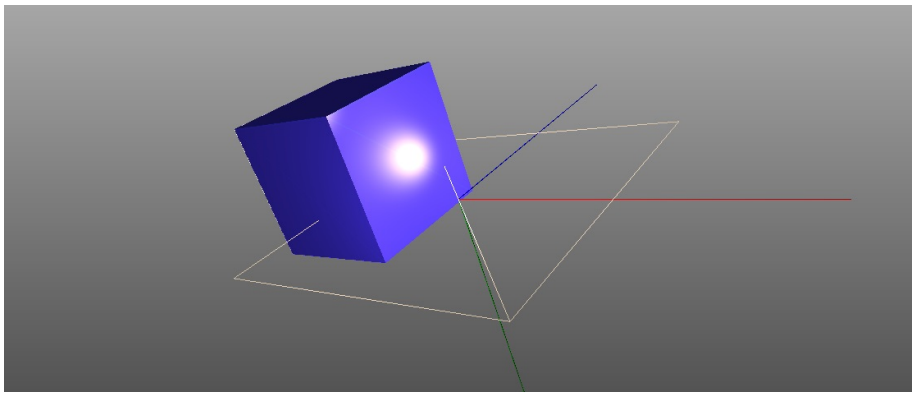
Clicking on the shape's name in the Shapes View will cause that shape and any children to appear in the Properties view.

images/GeometryPropertiesView.jpg

Editing the values in this view will change the selected shape accordingly.



Mesh Properties - These controls will allow the user to set any custom properties of the mesh. A cube has “sideLength”, which will control the size of the cube.

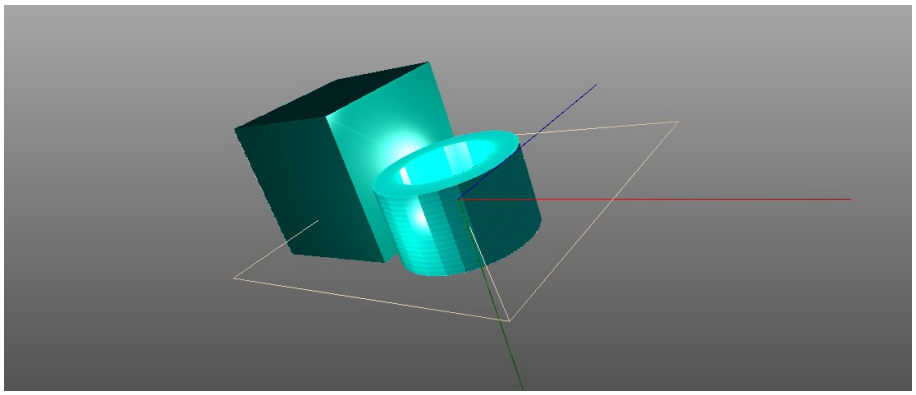


Center - These controls allow the shape to be moved about hte three dimensional space by setting the X, Y, and Z coordinates of its center.

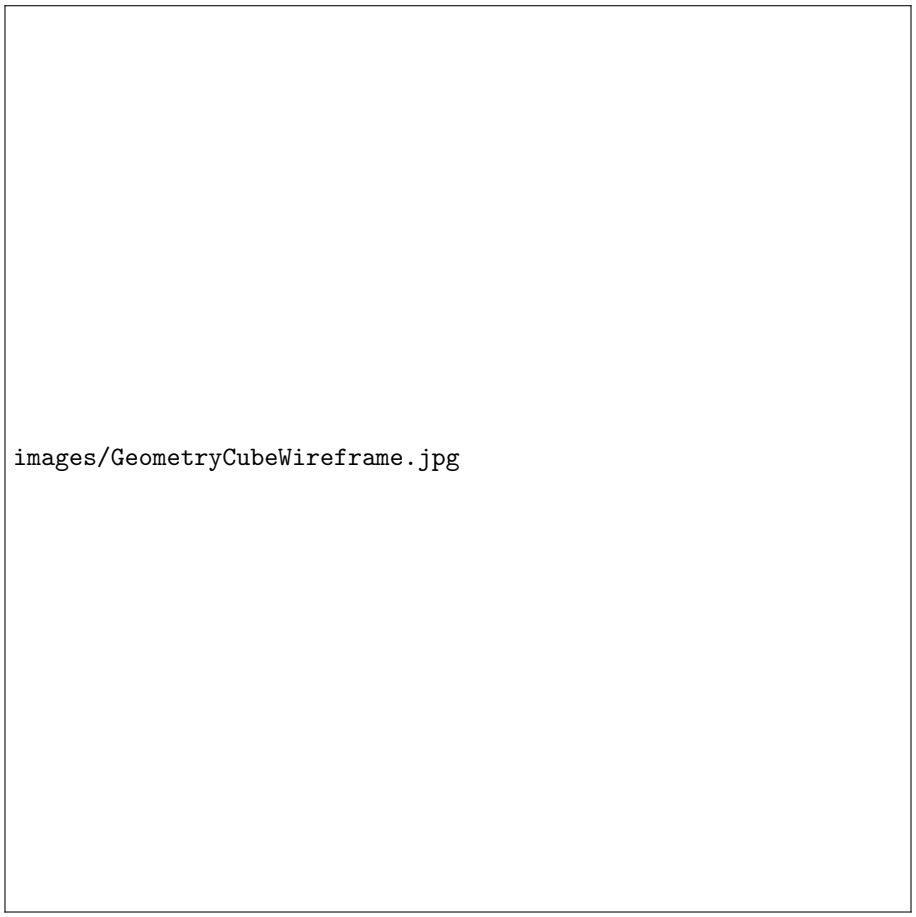
Triangle Mesh Data - This table list the coordinates for all vertices in each triangle used to define the shape.

images/GeometryCubeColor.jpg

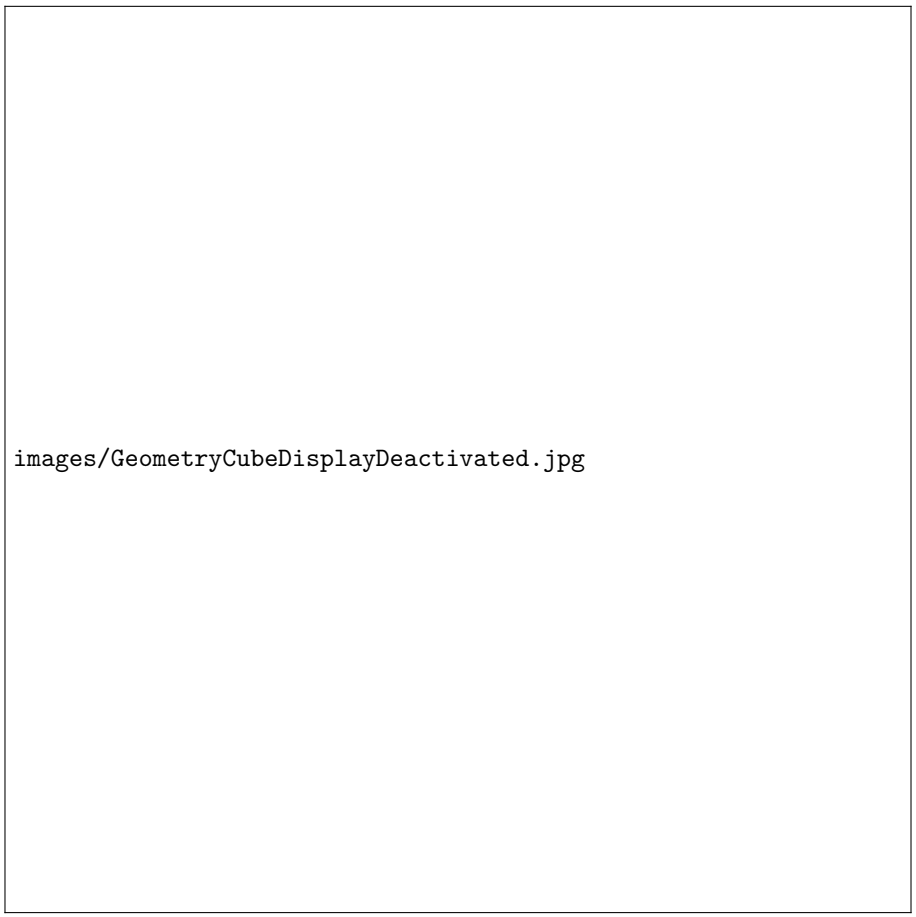
Color - These controls allow for setting the shape's color.



Scale - Controls the magnification of the shape. For more complex shapes such as tubes or imported files, it is a simpler way to set the geometry's size.



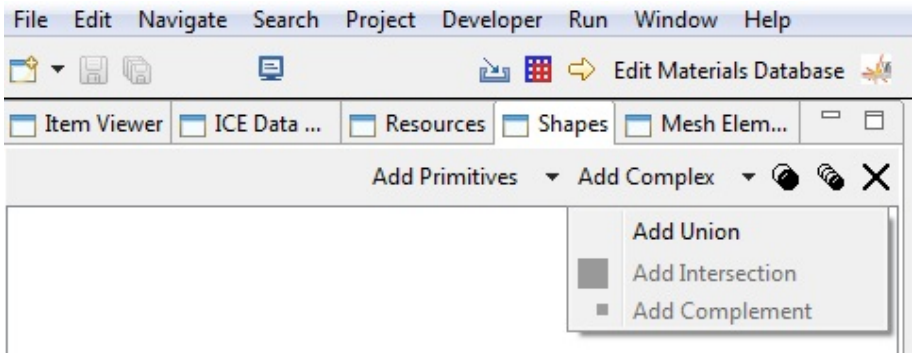
Opacity - This menu allows for a shape to be displayed as a wireframe or to be rendered transparent and removed from the scene.



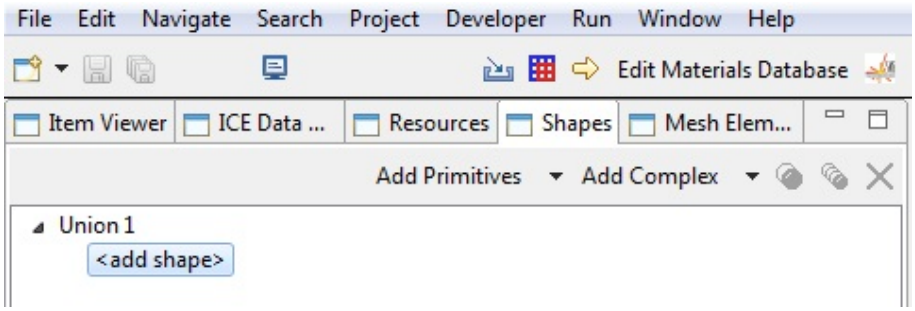
The check boxes by each display option allow that effect to be toggled, allowing you to quickly change a shape back and forth from its default display to your custom settings. Notice how the cube remained the same size, as it was edited to be larger, while the tube was kept the same size but had only been set to draw larger in the display menu.

Complex Shapes

Multiple primitive shapes can be combined within a single grouping for easier editing. First, create a complex shape using the Add Complex button.

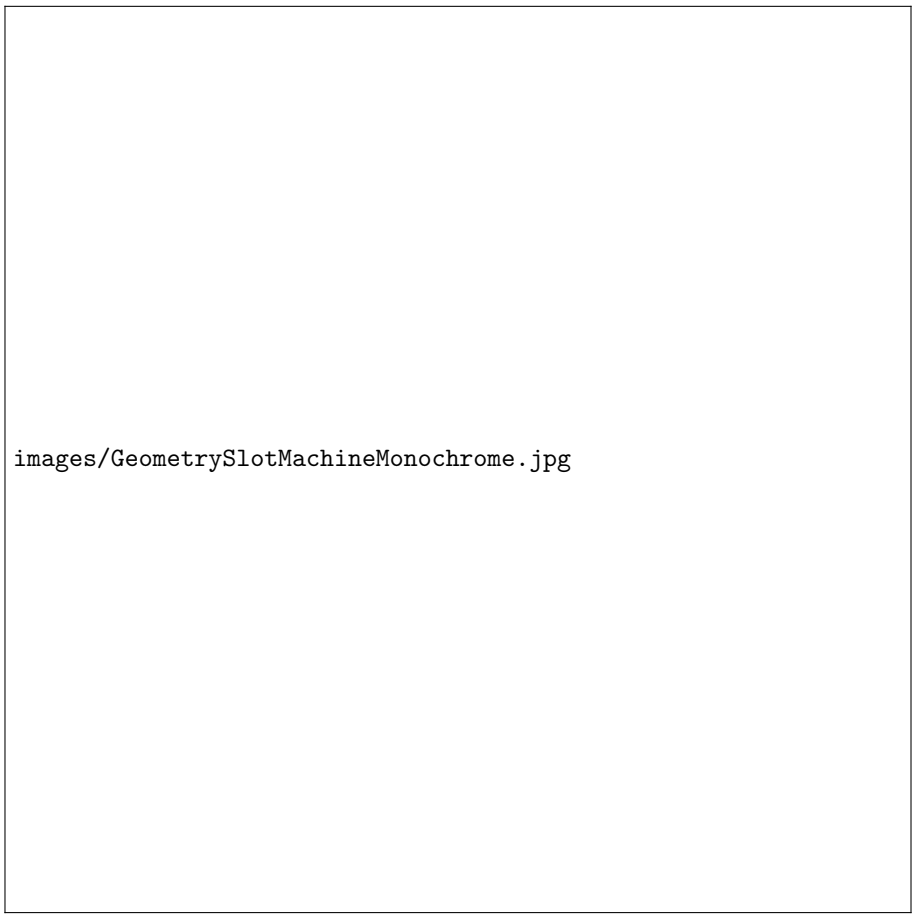


Currently, only unions are supported. The union will appear in the Shapes View with an empty spot for a child shape.

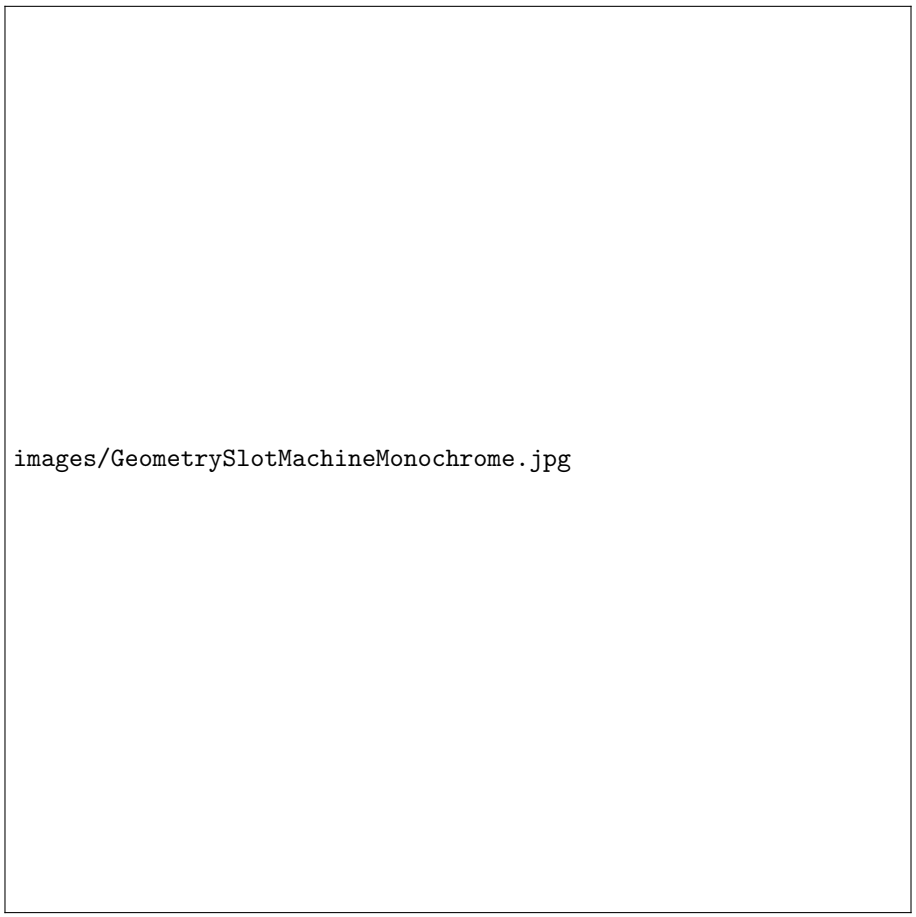


Select <Add Shape>and add a primitive shape as before to add the shape as a child of the union. To add additional shapes, select the child shape and add another as normal.

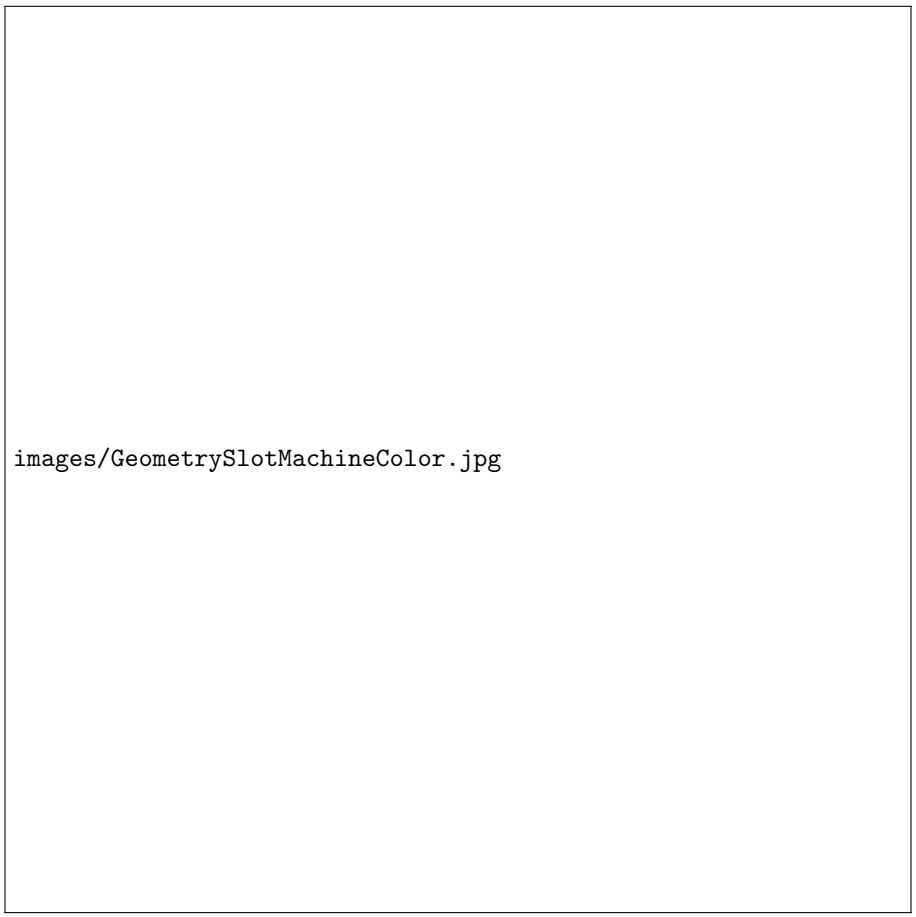
Selecting the union will select all of its child shapes, and changes to the Properties View will affect the entire complex shape. Individual shapes can still be selected for editing on their own, but you must deactivate all parent shape's display options if they conflict with the childrens.



For example, this slot machine is a union of shapes. While the union is set to a purple color, all child shapes are also set to purple.



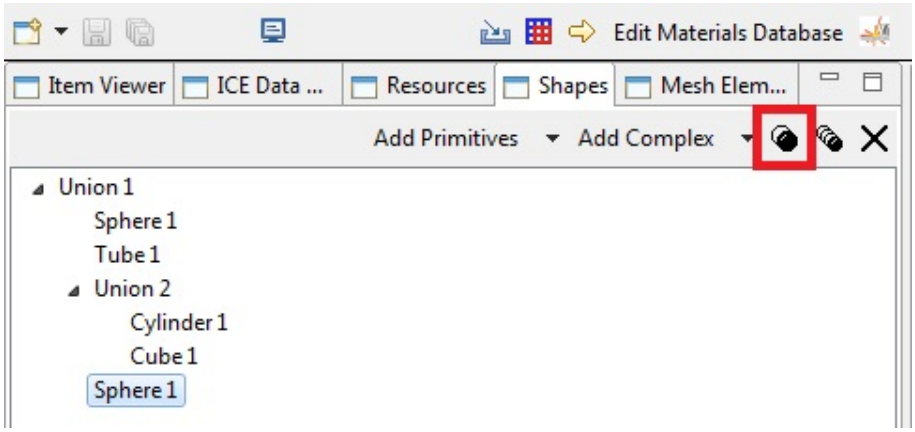
By deactivating the union's color display, the child shapes will each be able to display their own color.



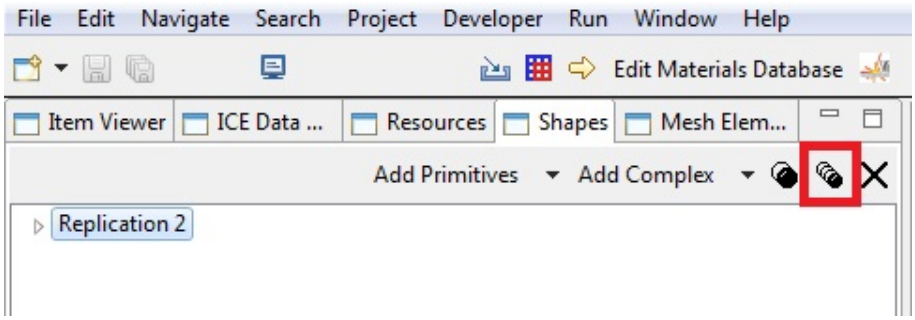
You can add a complex shape as a child to a complex shape in the same way as a primitive shape, allowing for nested unions

Copying

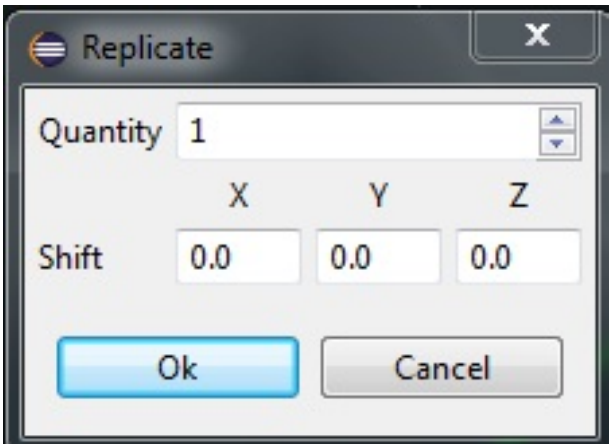
Once you have a shape created, you can automatically create copies of that shape.



Press the Duplicate Shape button, highlighted above, to create an exact copy of the selected primitive shape. The copy will appear as the child of the same complex shape as the original, if any, and can be modified independently after creation.



You can systematically create many copies of a shape with the Replicate Shape button highlighted above. This will open a new dialog.



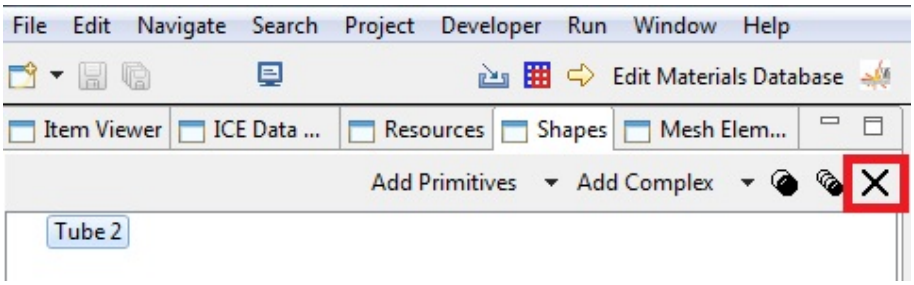
The quantity is the number of desired copies of the shape which should be in the replication. This includes the original (ie setting quantity to 2 will result in 2 shapes, the original and a copy).

The Shift boxes allow you to specify an offset to be placed between each copy. For example, if you set X to 100 and Y to 50, then each copy will be 100 units along the X axis and 50 along the Y away from the previous copy.

The original shape and all copies will be placed in a new complex shape, which will take the original shape's position in the tree.

Deletion

You may remove a shape and all its children by selecting it in the Shapes view and clicking the Delete button, highlighted below.



Importing

The Import File button next to Add Primitive will add import the contents of an .obj or .stl file. These will appear as a union containing other shapes inside.

Saving

You may save the contents of the Geometry Editor. This can be done as normal for an Eclipse file, using the Save button or `Ctrl + S`. The result will be a file named Geometry_Editor.xml in the ItemDB folder. The Geometry Editor can be reponed by double clicking on this file.

Chapter 8

Mesh Editor

8.1 Editing Meshes

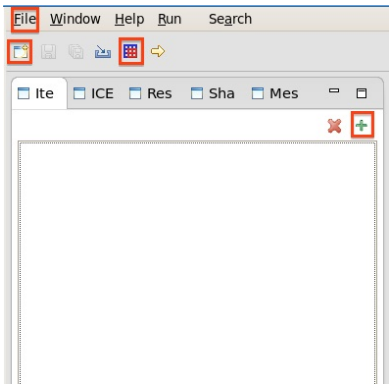
This document is designed to outline the basic user controls of the Mesh Editor plugin in ICE.

8.1.1 Getting Started

Once ICE is installed on your system, there are no addional dependencies or preparation required to use the Mesh Editor.

Opening a Mesh Editor

To open a Mesh Editor in ICE, you have four options:



The UI elements which can be used to open a Mesh Editor are highlighted. Instruction for how to use each one, from top to bottom, left to right, are given below.

1) Click the File menu, then New, then Other... and select the Create Item Wizard in the new dialog and press Next. Then, select Mesh Editor from the list and press Finish.

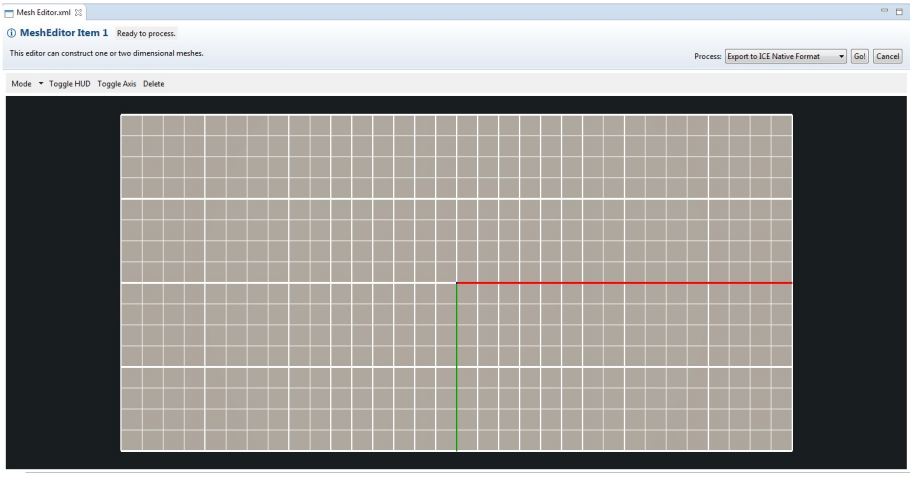
2) Click the New button, select Create Item Wizard in the new dialog and press Next. Then, select Mesh Editor from the list and press Finish.

3) Click the Mesh Editor button.

4) Enter the ICE Perspective by clicking the Open Perspective button in the upper right corner of the screen, select ICE from the dialog that pops up, and click OK. Afterwards, click the Create an Item button, select Mesh Editor, and click OK.

8.1.2 Working With the Mesh Editor

The mesh editor will initially open to an empty grid, as shown below. There is currently no way to import pre-existing meshes from another source directly into the Mesh Editor.



Navigation

Meshes are constructed on the background grid. Gridlines are spaced one unit apart from each other. The origin is the initial center of the screen with the x axis in red and the y axis in green coming out from it in the positive x and y directions, respectively.

Camera Controls

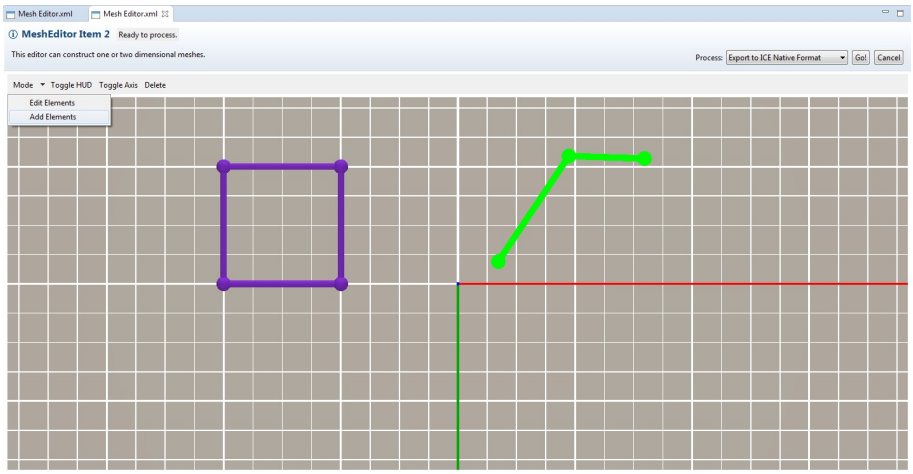
The camera is controlled with keyboard and mouse commands. The W, A, S, and D keys are used to move the camera around the editor's area, while scrolling the mouse wheel is used to zoom the camera in and out.

If the controls are not working, ensure that the Mesh Editor has focus by clicking inside of it.

Camera Controls	
Action	**Key(s)**
Scroll Up	W or Up Arrow
Scroll Down	S or Left Arrow
Scroll Left	A or Right Arrow
Scroll Right	D or Down Arrow
Zoom In	Scroll mouse wheel up
Zoom out	Scroll mouse wheel down

Adding Elements

2D meshes are constructed in the editor by specifying one quadrilateral at a time. To add a new polygon, the Mesh Editor must be in Add Elements Mode. This mode is the editor's default setting, and it can later be reset by clicking the Mode button in the top left corner.



Placing Vertices

In Add Elements Mode, clicking anywhere on the grid will place a new vertex at that location. These new, temporary vertices and the edges between them will be colored in green, to show that the polygon is still under construction.

Alternatively, you may select a vertex not already in the new polygon. This allows you to reuse vertices and/or edges already present in the mesh to form part of your new polygon.

Once the fourth vertex has been specified, the polygon will be displayed in full. Clicking one more time will change it to purple to show that it has been completed. At any time before this, you can press the backspace, delete, or escape buttons to cancel the new polygon, removing it from the editor and allowing you to start the process over.

Editing Elements

To edit an already present element of the mesh, you must switch to Edit Elements Mode. The Mode button in the upper left corner allows you to switch

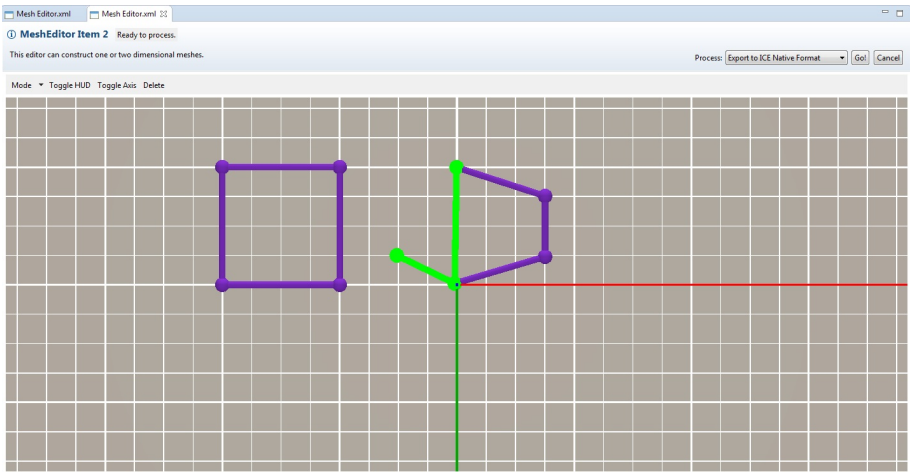
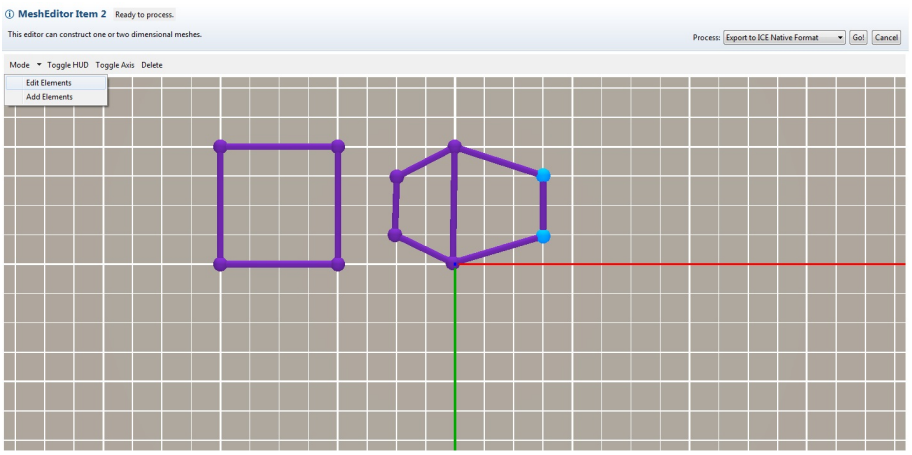


Figure 8.1: One of the edges from the trapezoid is being combined with a new edge in the creation of the current polygon.

between modes, as shown below.



A vertex can be selected by either clicking it in the Mesh Editor, or by selecting it from the tree in the ICE Perspective's Mesh Elements View. Holding down shift while clicking will allow multiple vertices to be selected at once, while holding down control during a click will toggle a vertex's state either into or out of the selection. Selected elements are displayed in blue. The current selection can be cleared by pressing the backspace, delete, or escape buttons.

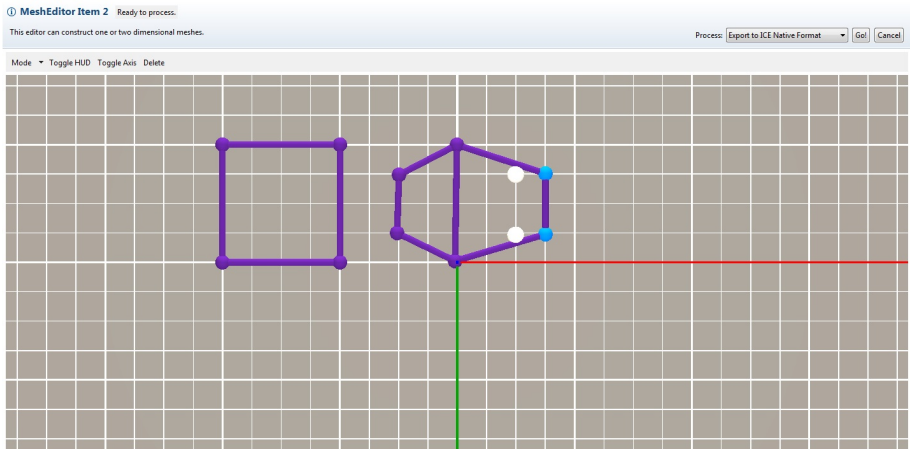
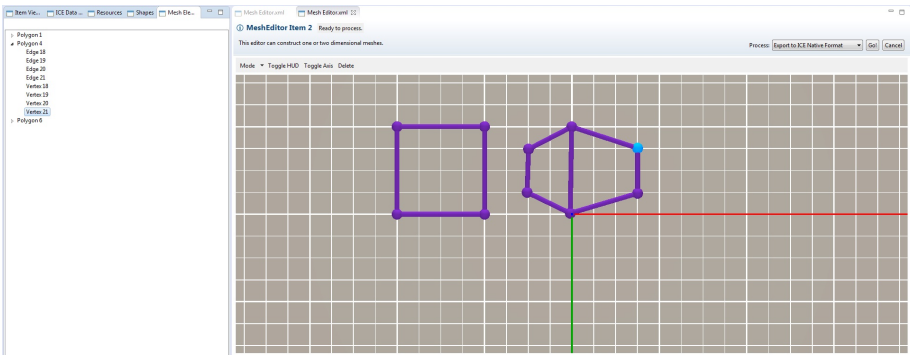


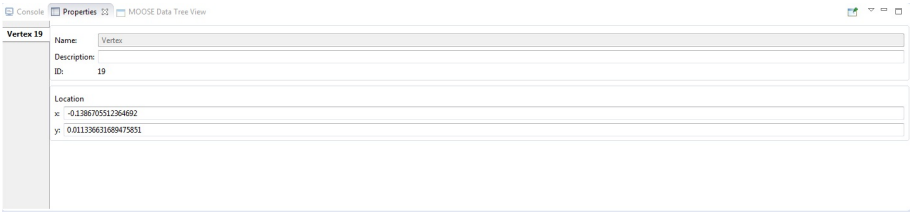
Figure 8.2: Two vertices are selected. When the bottom vertex is moved left, the top vertex is moved left by an equal amount.



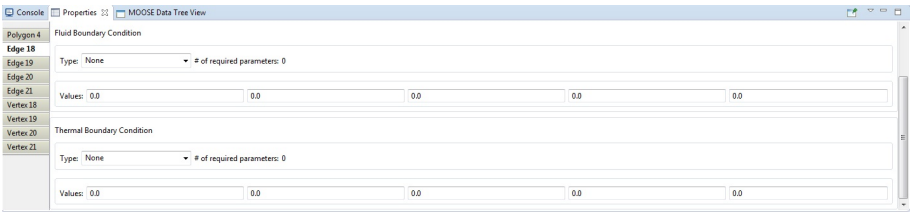
Click on a selected vertex and drag it to change its location. The clicked vertex will stay beneath your mouse cursor, while all other selected vertices will be moved as well, keeping their relative position to the dragged vertex. White circles are displayed to show each vertex's new location. Pressing backspace, delete, or escape during the drag will deselect the vertices and cancel the movement.

Edit Mode Controls	
Action	**Key(s)**
Select vertex	Left click
Add to selection	Shift + left click
Toggle (Add/Remove) Selection	Ctrl + left click
Move selected vertices	Left click on vertex and drag mouse
Clear selection/Cancel move	Backspace, Delete, or Escape

Alternatively, you can edit a vertex more precisely by setting its exact coordinates. First, select it in the Mesh Elements View, then open the Properties View. The vertex's x and y coordinates will be displayed in editable text boxes.



Selecting a polygon in the same way, then opening the tab for one of its edges in the Properties View, will display the edge's boundary conditions for that polygon. These are editable, just like the vertices are.

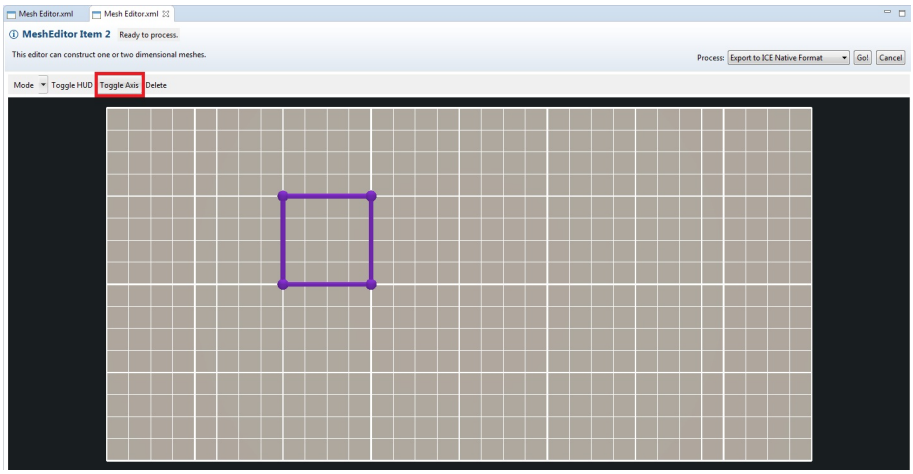


Deleting Elements

Polygons in the mesh can be deleted. All four vertices for the polygon must be selected, then press the Delete button on the toolbar. Vertices and edges which are still part of other, undeleted, polygons will remain, but all others will be removed from the editor.

Additional Controls

The Toggle Axis button on the toolbar will show/hide the axes in the editor.



The Toggle HUD button will remove or display the small info bar beneath the editor which shows the current cursor and camera positions.

Saving

You may save the contents of the Mesh Editor. This can be done as normal for an Eclipse file, using the Save button or Ctrl + S. The result will be a file named Mesh_Editor.xml in the ItemDB folder. The Mesh Editor can be reponed by double clicking on this file.

Chapter 9

Visualization

9.1 Visualization Tools

Currently, ICE features two plugins for visualizing and plotting simulation output data:

VisIt Tools - An interactive visualization tool for rendering data defined on 2D and 3D meshes.

CSV Plotting Tools - A customizable, 2D data plotting utility for data in CSV format.

The CSV Plotting Tools require no additional software or preparation before use. The VisIt Tools need both a VisIt installation and a connection between ICE and a VisIt session.

9.1.1 VisIt Installation

Before preparing ICE, VisIt must be downloaded and installed. The ICE development team recommends using the latest working version of VisIt (2.9.2) but acknowledges that any version greater than or equal to 2.8.2 should work. The latest VisIt version, 2.10, is not currently compatible with ICE. VisIt does not need to be installed on the same machine where ICE is installed since ICE is capable of launching a VisIt session on a remote machine. Regardless of the host, make note of the directory where VisIt is installed as it will be necessary in the next step.

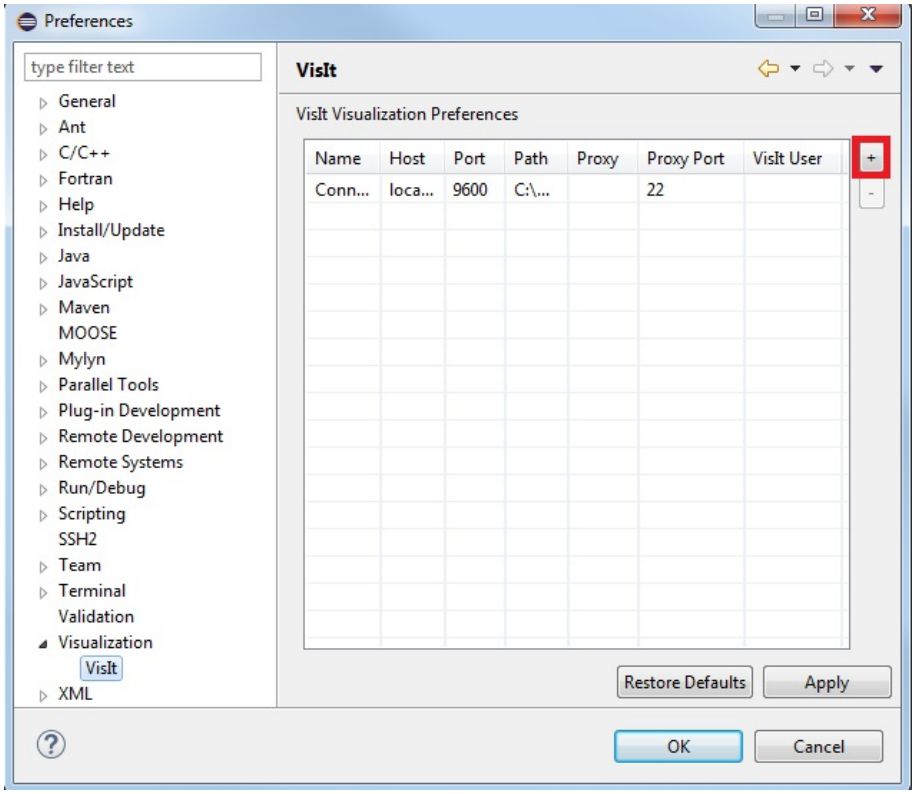
Configuring the VisIt Connection

Once VisIt is installed, ICE must connect to a running instance of VisIt in order to make use of its rendering capabilities. ICE provides two different tools that utilize VisIt, the Plot Editor and the Visualization Perspective. Both utilities provide slightly different functionality and are accessed through different means.

Connecting for the Plot Editor

The Plot Editor uses a default connection to VisIt established in the ICE Preferences page. This process only needs to be performed once. After initially creating the connection, ICE will launch and connect with VisIt upon each subsequent launch of ICE.

To set the connection, select Window → Preferences... in ICE's menu bar. (On Mac OS X, Preferences... is instead located under ICE in the menu bar.) Select Visualization → VisIt in the tree on the left side of the Preferences window.



Press the button with a "+" symbol in the upper right (highlighted in the image above) to add a new row to the table. Click on cells in the new row to edit their values. The default values automatically supplied by ICE should be appropriate for most users. However, two fields may need to be changed:

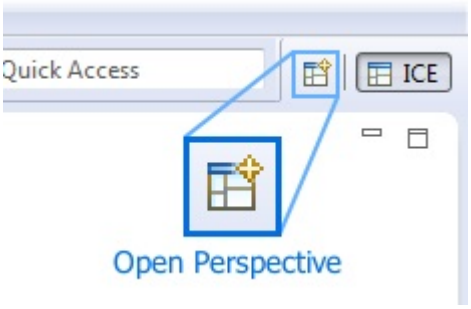
Host: The default value of "localhost" allows for connections to VisIt installations on the same machine where ICE is running. To launch a remote VisIt connection, change this to the hostname of that machine.

Path: Enter the full path to the directory containing the VisIt executable, not the executable itself. The VisIt executable is named *visit* on Linux and Mac OS X. On Windows, *VisIt.exe* is the appropriate file.

Once finished editing the cells in the new row, press Apply, then OK. ICE will then launch an instance of VisIt and connect to it.

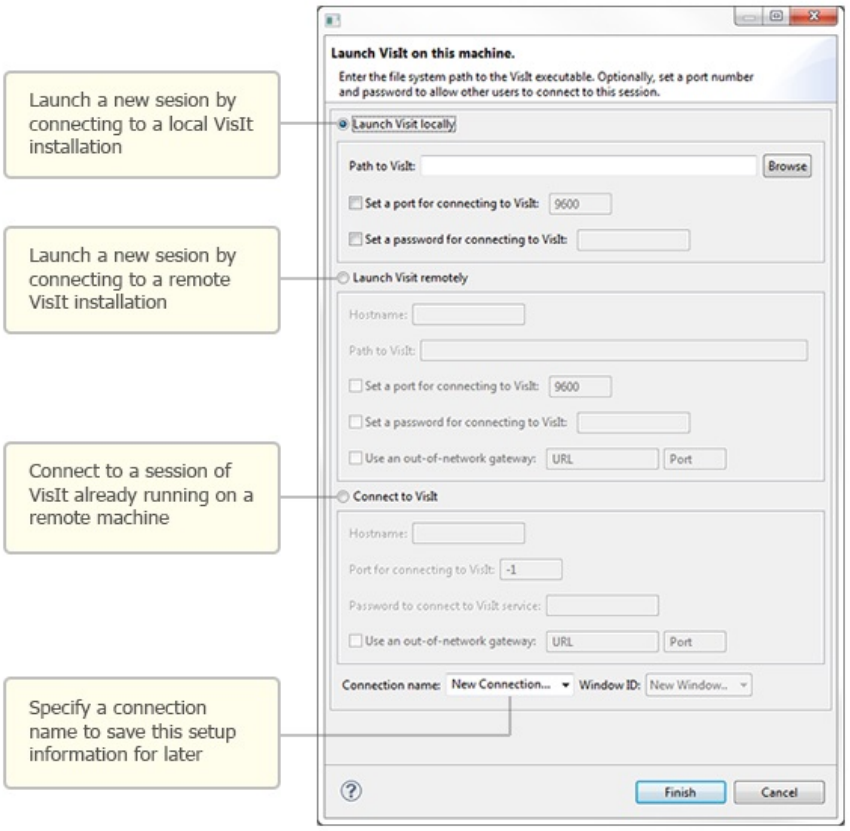
Connecting for the Visualization Perspective

To establish a VisIt connection in the Visualization perspective, begin by opening this utility. In the main menu bar at the top of the window, select Window → Perspective → Open Perspective → Other.... Select Visualization in the dialog that appears and click OK. Alternatively, the same dialog may be accessed by clicking the Open Perspective button in the toolbar in the upper right-hand corner of the ICE workbench.



Click the Launch VisIt button in the tool bar to enter the VisIt connection parameters.

The resulting dialog offers three options for connecting to VisIt.



1) Launch VisIt locally - This connection option will launch a new VisIt session on the machine running ICE. If VisIt in installed on this machine, use the Browse button to enter the directory containing the VisIt executable into the

Path to VisIt field. Optionally, set a port number (default 9600) that VisIt will use to serve data to ICE, and if this VisIt session will be shared with multiple users, set a password.

2) Launch VisIt remotely - Using this method of connecting will launch a new VisIt session on a machine other than the one used to run ICE. Specify the hostname and full path to the directory containing the VisIt executable. Optionally, enter a port number (default 9600), and if VisIt session will be shared with multiple users, enter a password. If access to the remote machine where VisIt is installed requires the use of an external gateway or proxy, specify its URL and port number, as well.

3) Connect to VisIt - To connect to a previously launched VisIt session, specify the hostname, port number, and password set on the lanuch of that session. This information will need to be obtained from the person who initially launched the VisIt session. If access to the machine hosting the VisIt session requires the use of an external gateway or proxy, enter its URL and port number, as well.

For a reminder of where VisIt is installed on Windows, find a shortcut to VisIt on the desktop or in the start menu. Right-click the shortcut and open its Properties. The path to the VisIt executable's directoy will be shown next to Target.

Regardless of the method used to connect to VisIt, enter a Connection name at the bottom of the dialog.

When connecting to an existing session, specify a Window ID between 1 and 16. The Window ID used determines how ICE will connect to VisIt. If multiple users connect using the same Window ID, they will all see and be able to interact with the same VisIt view. However, if multiple users wish to each have their own unique session with its own controls, assign a unique Window ID to each user. The VisIt installation can support up to 16 unique window IDs at a time.

Once the required fields are complete, click the Finish button at the bottom, and ICE should begin connecting to VisIt.

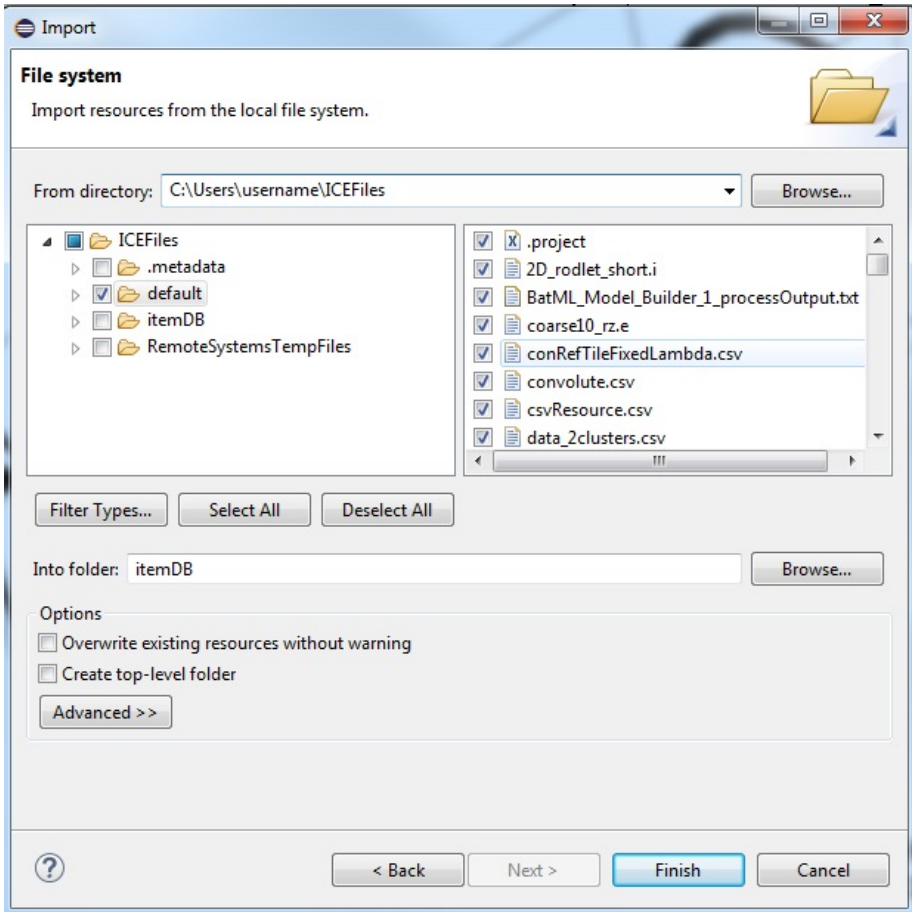
9.2 Using VisIt

9.2.1 Opening a Plot Editor

To open a Plot Editor, a file that uses this editor must first be placed in the Project Explorer. This view lists files imported into ICE. To access the Project

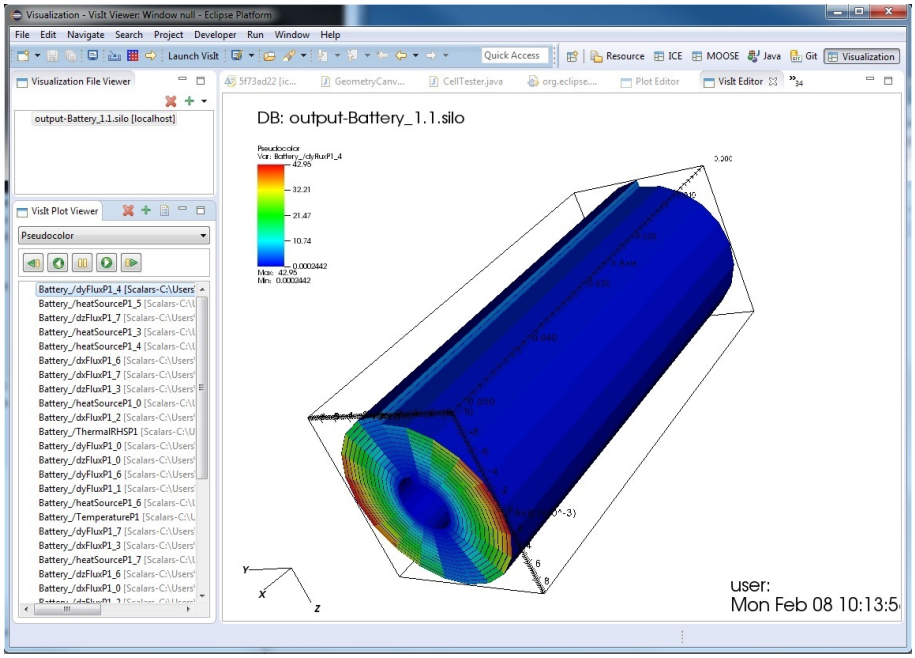
Explorer, use the menu bar at the top of the window and navigate to Window → Show View → Project Explorer. Depending on the active Eclipse perspective, opening this view may require selecting Other... and finding the Project Explorer in the dialog under the General category in the tree.

By default, the Project Explorer should automatically import the ICEFiles/default and ICEFiles/itemDB folders. If it does not, or if you want to import a different folder into ICE, right click in the Project Explorer and select Import... from the context menu. Then, select General → File System from the tree, and press the Next button. Select directories and/or files to import into the Project Explorer, and enter which folder they should be imported into, as shown below.

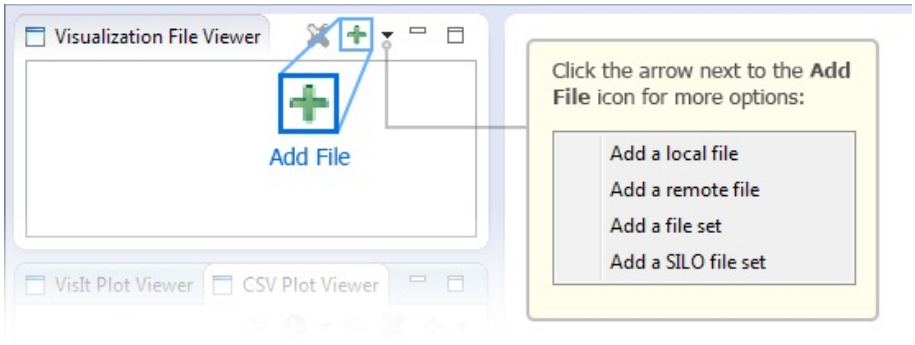


Once a file is in the Project Explorer, simply double click on it to open it in VisIt.

9.2.2 Opening a file in the Visualization Perspective

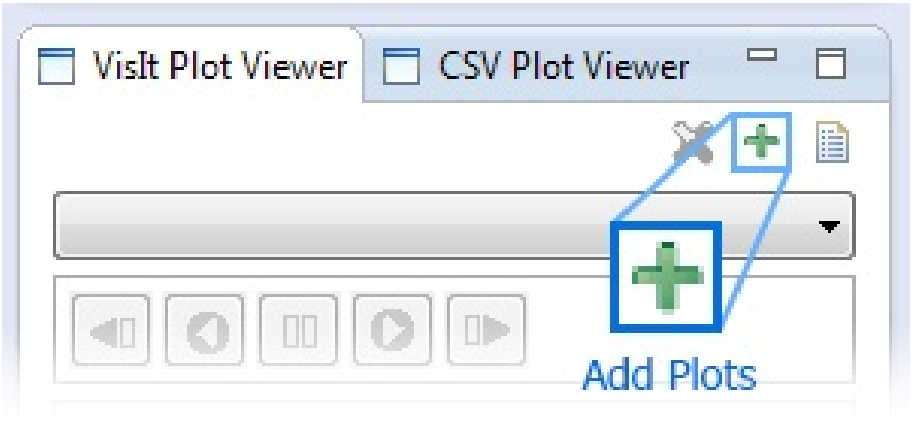


To open a file in the Visualization Perspective, first import the file into the Visualization File Viewer, located in the upper left of the screen. Click the Open a File button (green plus sign icon). The resulting dialog allows for the selection of local files to import.

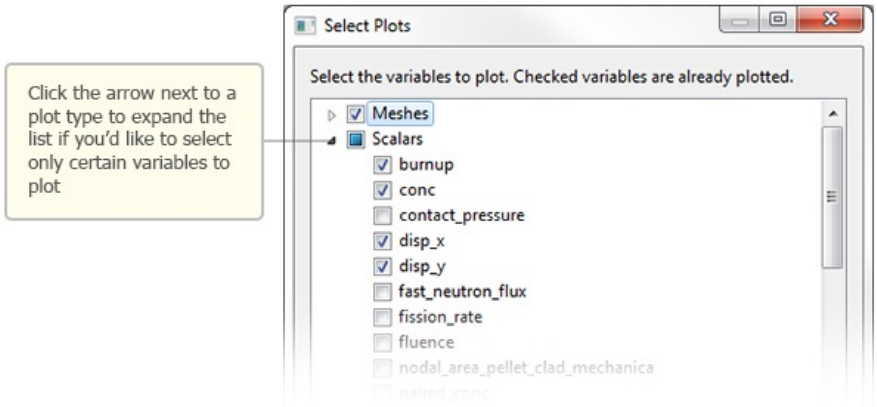


Currently, ICE can only open files in VisIt on the same machine that is running the VisIt session. For a connection to a remote VisIt installation, click the arrow beside the Open a File button, and select "Add a remote file" from

the drop down menu. The resulting dialog box allows the user to browse the file system of the remote machine hosting the VisIt session.



Once a file is in the Visualization File Viewer, create a plot by selecting the file, and then clicking the Add a Plot to the List button located in the VisIt Plot Viewer in the lower part of the column on the left side of the workbench. The resulting dialog allows the user to select plots from the file to view. After making a selection or selctions and pressing OK, these plots will be placed in the VisIt Plot Viewer.

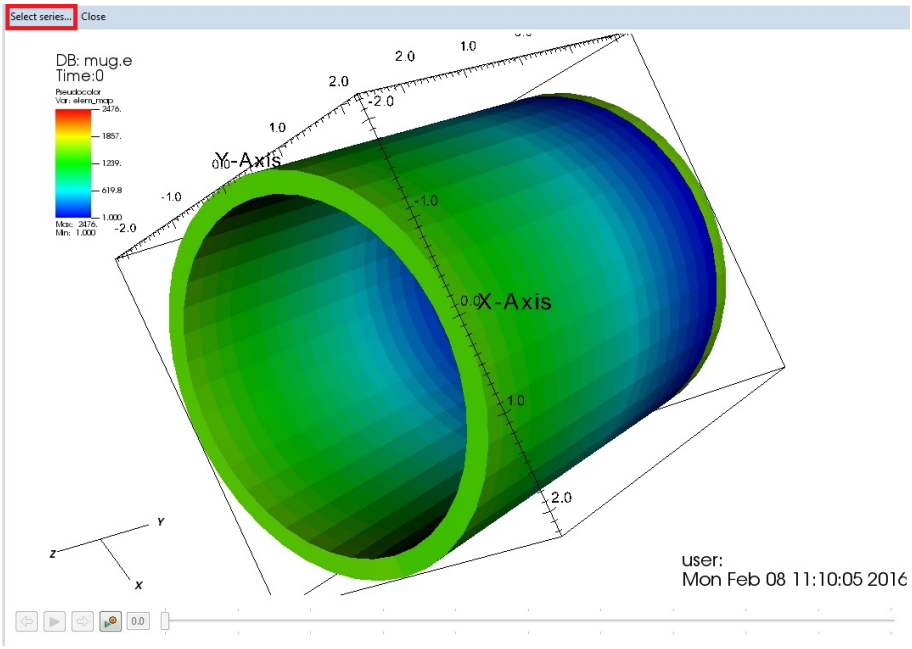


Finally, double click a plot in the VisIt Plot Viewer to render the data to the screen.

Both the Plot Editor and the Visualization Perspective allow the user to rotate the model by clicking and dragging inside the display area or adjust the zoom by scrolling the mouse wheel. Other commands vary slightly between the two utilities.

9.2.3 Selecting the Plot

In the Plot Editor, pressing the Select Series... button will open a dialog which lists the various plots in the opened file. Simply select one and click OK to open it.



In the Visualization Perspective, the opened plots will be listed in the VisIt Plot Viewer. Double click on one to open it.

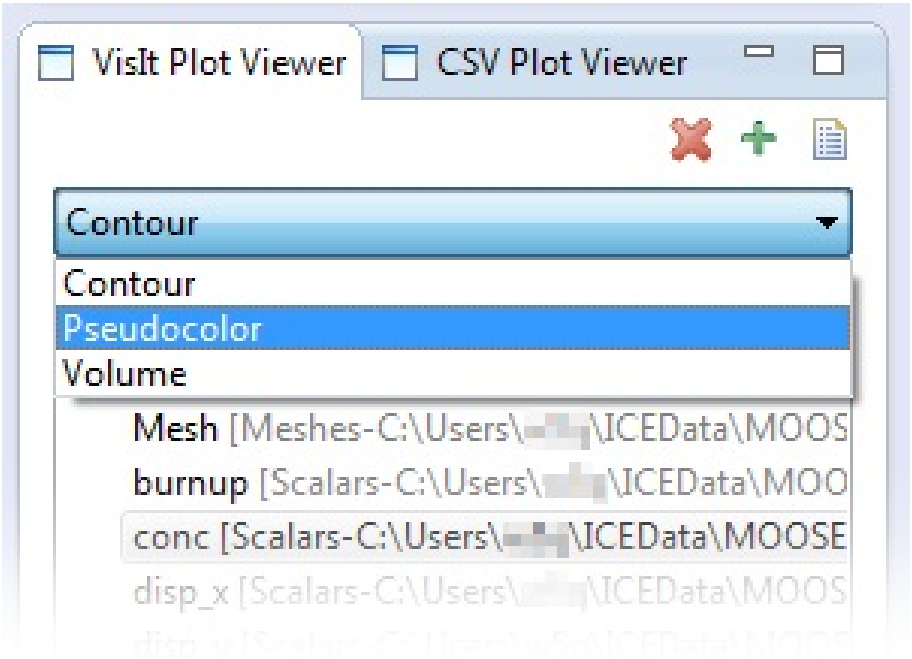
9.2.4 Setting the Plot Representation

VisIt is capable of displaying plots in several different representations, such as pseudocolor, contour, or volume.

To switch between plot types in the Plot Editor, right click inside the display

area and select one of the listed options under the Representation category in the context menu.

The Visualization Perspective features a drop down menu at the top of the VisIt Plot Viewer which allows for switching between the available representations.



9.2.5 Animation and Time Data

The Plot Editor features a time slider widget at the bottom of the screen.

The controls, in order of left to right, are:

1) Return to the previous time step.

2) Automatically play the plot as an animation by displaying the time steps sequentially.

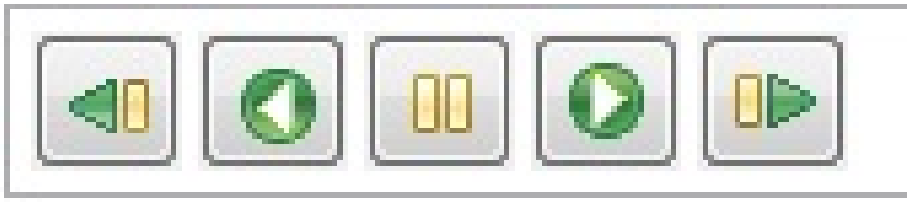
3) Advance to the next time step.

4) Opens an options menu that allows the user to set the playback speed, toggle whether the animation should loop when it reaches the end, and set the plot to the first or last time step.

5) A display for the current time step. It can be edited to set the plot to an arbitrary time step.

6) A slider that shows the current time step's position on the timeline. The slider can be dragged around the timeline, setting the plot's time step accordingly.

In the Visualization Perspective, the Visit Plot Viewer has a similiar set of controls.



The buttons, in order of left to right, are:

1) Return to the previous time step.

2) Automatically play the plot as an animation backwards, going through the time steps in reverse order.

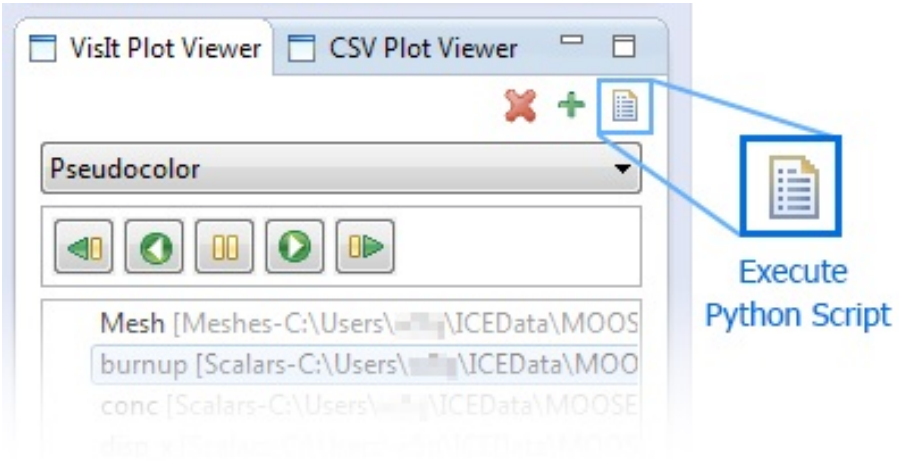
3) Pause the animation.

4) Automatically play the plot as an animation by displaying the time steps sequentially.

5) Advance to the next time step.

9.2.6 Sending VisIt Python Commands

This functionality is only available in the Visualization Perspective.



The Execute Python Script button in the upper right of the VisIt Plot Viewer will open a new window with a Python shell. Commands entered into this shell will be sent to the instance of VisIt.

Python code can be written into the shell directly, but the Load from File button will import an existing .py script into the console. Once done, hit the Execute button to send the python command(s) to VisIt.

Writing Python scripts for VisIt is beyond the scope of this tutorial. Please refer to the VisIt Python Interface Manual provided by the VisIt development team at Lawrence Livermore National Laboratory for information on Python commands for VisIt.

9.3 Using the CSV Plot Viewer

ICE offers functionality for the viewing of CSV files.

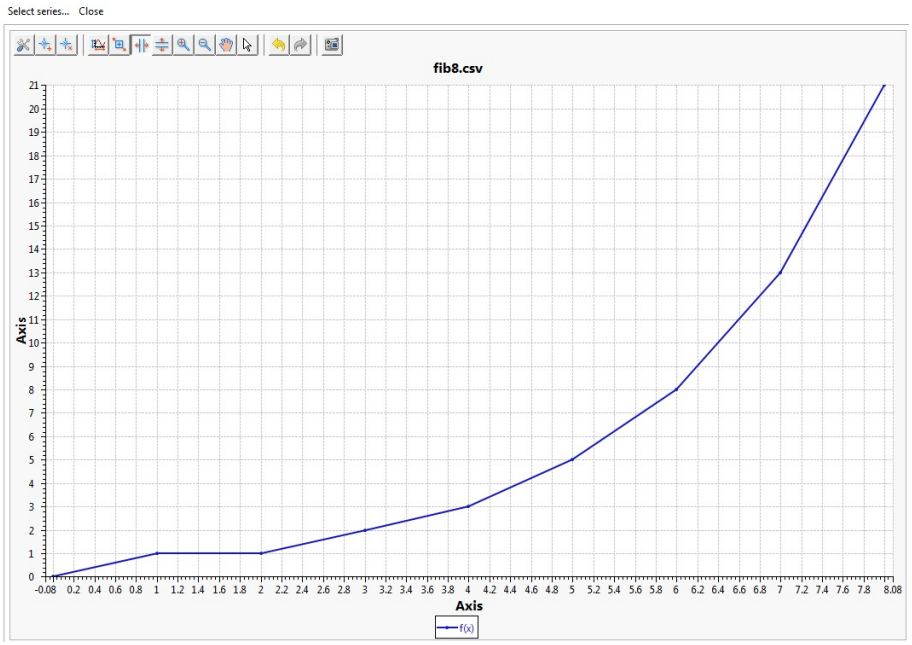
9.3.1 Opening a CSV File

CSV files are opened in Plot Editors through the Project Explorer. Open the Project Explorer with Window → Show View → Project Explorer and import the file by right clicking and selecting import. Once the file is in the Project Explorer, double click to open it.

ICE expects CSV files to be in an [m x n] format, with no row holding empty

values. The first row in the file will be used to name a series, with the series data being specified by the column of values beneath it.

9.3.2 Using the CSV Plot Viewer



Controlling the Graphics

The row of buttons at the top of the viewer provide basic graph editing capabilities.

The first button allows the user to edit a variety of basic graph attributes, such as font, axis titles and scales, line colors, etc.

The next two buttons allow for the addition and removal of annotations for specific data points.

The central grouping of buttons allow the user to zoom and pan the camera in a variety of ways.

The two yellow arrow buttons will undo/redo changes made to the graph.

The final button will export a screenshot of the graph.

Setting the Independent Series

By default, the first column in the file will serve as the independent series setting the graph's x axis. By right clicking the Plot Viewer and selecting Set Independent Series..., the user can set the independent series to any other series in the file.

Setting the Dependent Series

When first opened, only the series defined by the second column in the file will be displayed. There are several ways to change this.

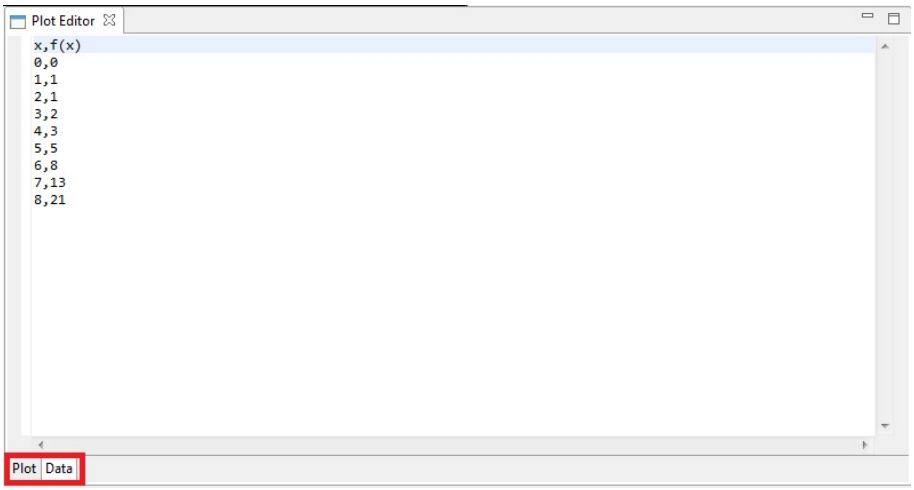
The Select series... button in the upper left hand corner will display a list of all the series in the file. Selecting one and pressing OK will graph that series and removing all others.

Right clicking and choosing Select series... from the context menu will open a dialog in which any of the available series may be selected. All selected series will be graphed at once with deselected series removed.

Finally, the Remove all series option in the context menu will completely clear the graph.

Viewing the Data

At the bottom of the editor is a series of tabs.



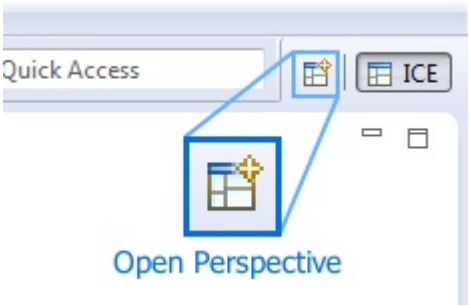
The Plot tab contains the graph described thus far. The Data tab will display the raw numerical source data in a text editor.

9.4 Using the MOOSE Embedded Visualization

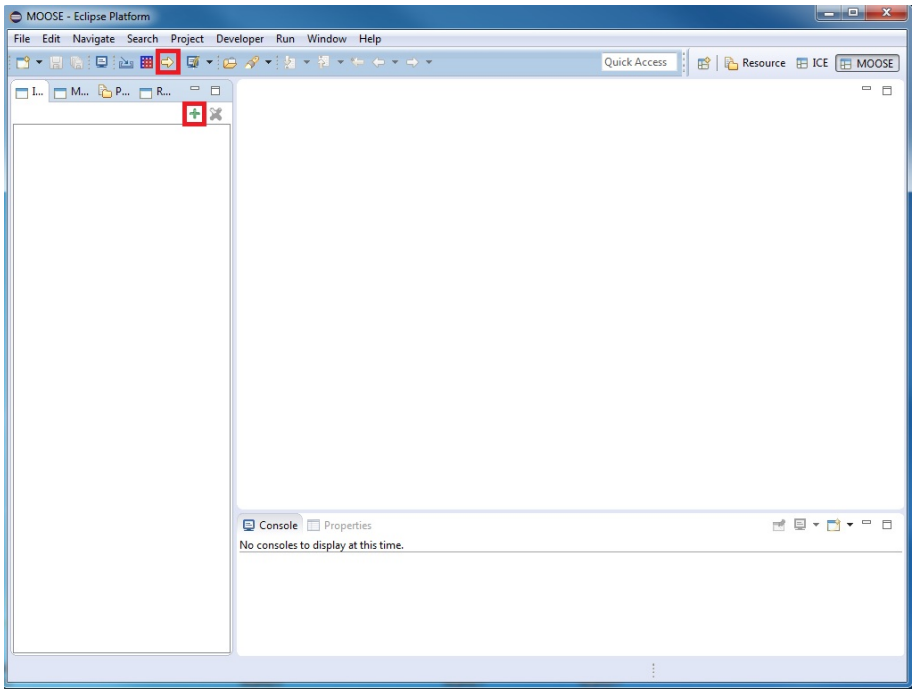
MOOSE Workflow Items allow for easy visualization of their associated files, making use of the VisIt and CSV Plot Editors described previously.

9.4.1 Creating a MOOSE Workflow

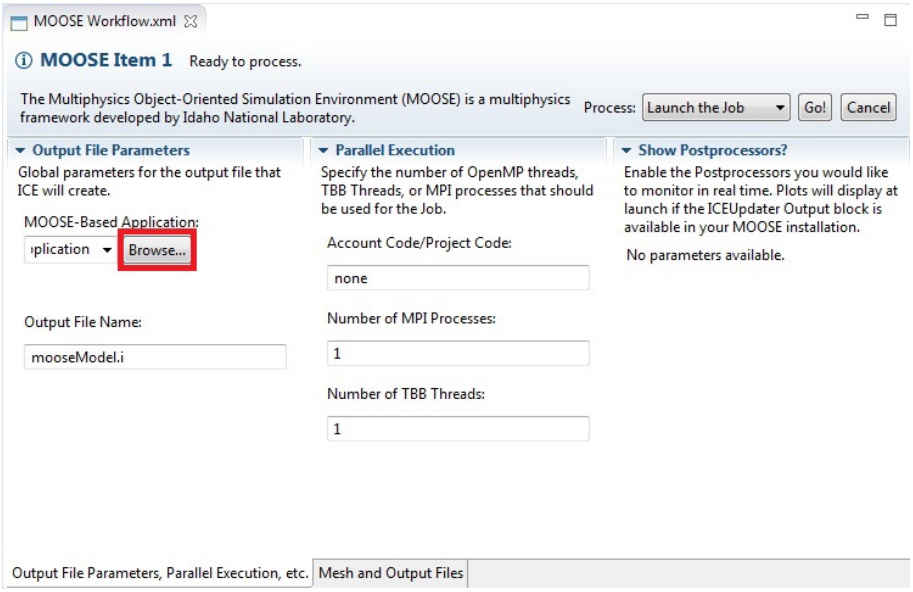
First open the MOOSE Perspective, either through Window → Perspective → Open Perspective → Other... and selecting MOOSE, or by clicking the Open Perspective button in the top right corner and selecting MOOSE.



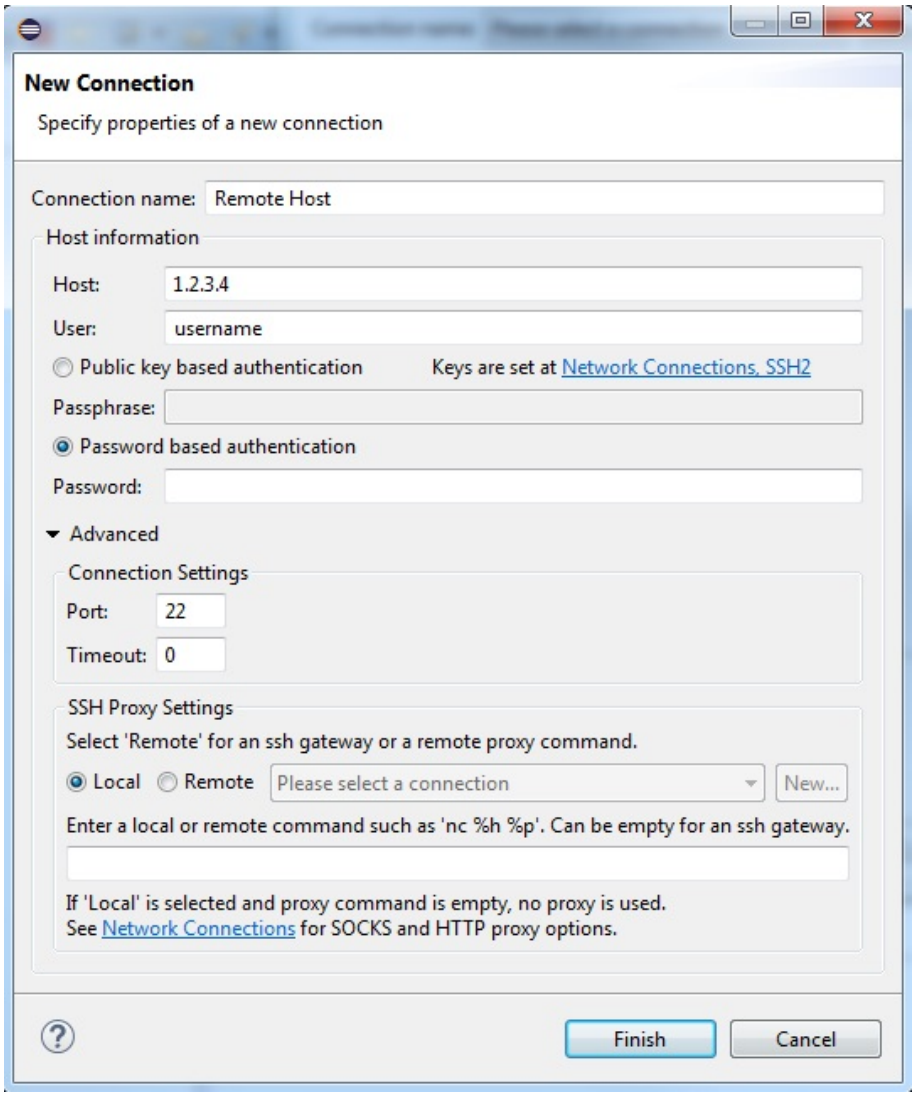
Next, you must open a MOOSE Workflow. The two buttons outlined in red below will create Items. The upper one will import a .i file, while the bottom button will create a new workflow. In either case, select MOOSE Workflow from the menu and hit OK.



You must now choose the MOOSE application which will run the file. The Browse... button under MOOSE-Based Application in the upper left hand corner will open a menu asking if the application is on the local machine or on a remote one. For local applications, you will simply need to select it from a dialog.



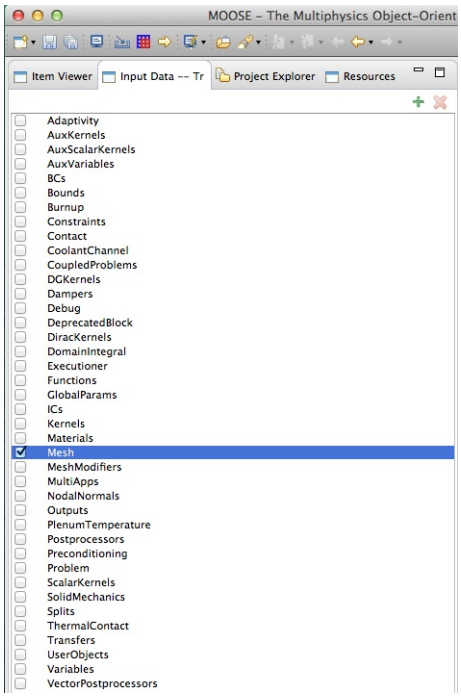
If the application is remote, a new dialog will open. At the top, press the New button to set up your connection.



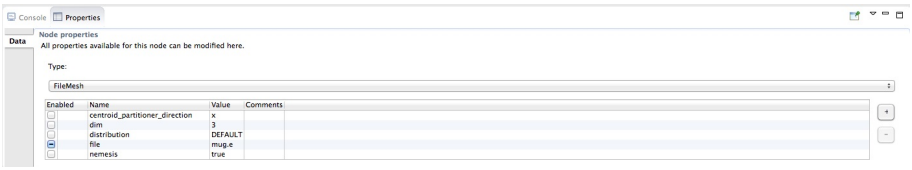
You must set the Host to the computer containing the MOOSE application, and User and Password to the username and password for your account on that machine. You may also optionally set the Port number, timemout, and proxy settings. Once done, click Finish. You will automatically be shown the file system for your new connection. Select the MOOSE application from it and click OK.

Save the form. If you imported a file, there will still be unsaved changes after a single save. You can save again and skip the next section.

Configuring the MOOSE Input Data



If you did not import a pre-written MOOSE file, some setup will be required before the job can create visualizations. Switch to the Input Data – Tree View tab on the left of the screen. Here you will see a list of MOOSE data categories. Click Mesh and switch to the Properties tab at the bottom.



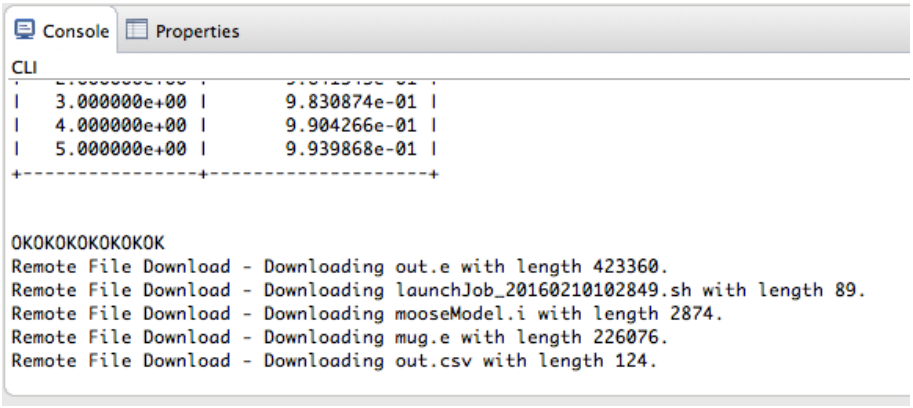
The Properties tab will display the properties of the Mesh variable. At the top will be drop down menu labeled Type. Select FileMesh from it and the table below it will be populated. In the table, select the file line and place the file containing the mesh you want to work with in it, as mug.e is in the example above.

If there are any variables you want to graph from the job, you will have to specify them as Variables and create PostProcessors which point to them,

similar to how you just set the Mesh.

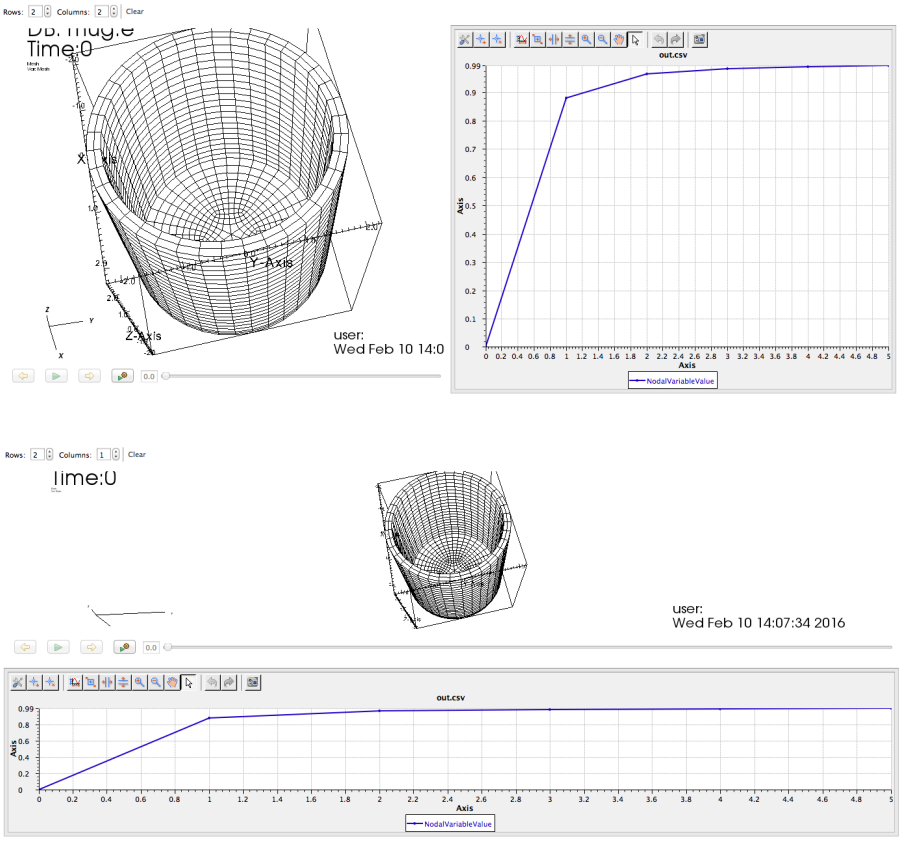
9.4.2 Viewing the Embedded Visualizations

Once your MOOSE Workflow is set up, make sure the Process control is set to "Launch the Job", and click the Go! button in the upper right hand corner. Wait until the console shows that the output files are finished downloading, similar to what is seen here:



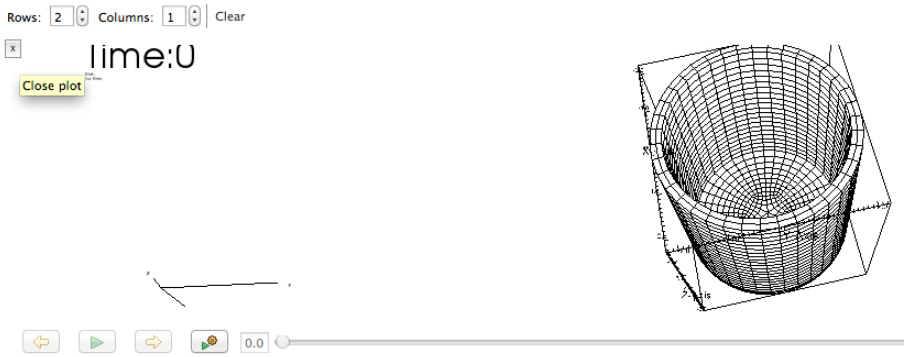
You can now change to the Resources tab on the left. This will list all the output files produced by the MOOSE job. Double clicking on one will open a Plot Editor inside the MOOSE Item's Mesh and Output Files tab. The visualization can be controlled normally, as described in previous sections.

Multiple resources can be opened at once. At the top of the screen are controls for the number of rows and columns making up the grid. These can be changed to set the layout for the various plots, as demonstrated below.



Be careful when reducing the number of rows or columns, as any plots which no longer fit in the grid will be closed.

If you hover over a plot, a button will appear in the upper left hand corner. Clicking it will close that plot.



Alternatively, the Clear button next to the controls for the number of rows and columns will close all the plots.

Chapter 10

Paraview

ICE features functionality for visualizing models using ParaView.

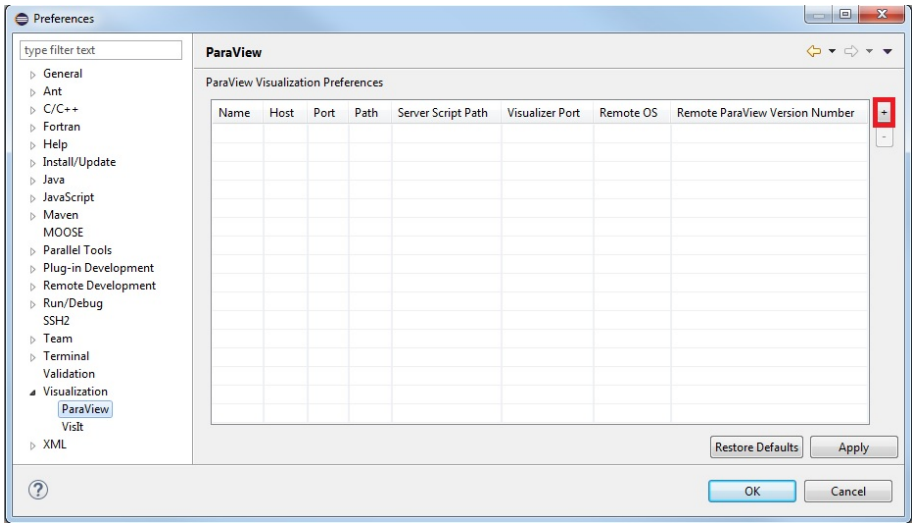
10.1 Installation and Configuration

ParaView use for ICE requires a Mac OS or Linux operating system, as ICE does not currently support ParaView connections to Windows hosts. However, it will still be possible to connect from ICE on Windows to a Mac or Linux machine that has ParaView installed. You will also need an installation of ParaView on your local machine. ParaView can be downloaded from its official website. The ICE development team recommends using the latest available version of ParaView, currently 5.0 at the time of this writing. You will further need a custom Python HTTP web server implementation, which can be downloaded from the Oak Ridge website.

10.1.1 Configuring the ParaView Connection

Once ParaView is running, ICE must be configured to connect to the server. This is done through specifying a default connection in the ICE Preferences page. This process only needs to be performed once. After initially creating the connection, ICE will attempt to connect to ParaView on that port each time it is launched.

To set the connection, select Window → Preferences... in ICE's menu bar. (On Mac OS X, Preferences... is located under ICE instead of Windows.) Select Visualization → ParaView in the tree on the left side of the Preferences window.



Press the button with a "+" symbol in the upper right (highlighted in the image above) to add a new row to the table. Click on cells in the new row to edit their values.

Name: The connection's name. The default value will be fine.

Host: The hostname for the machine that will run ParaView. Use "localhost" if the machine running ICE will also be used to run ParaView.

Port: The port number on which the ParaView server will run. The default value will be fine, but if you change it, it must different from the Visualizer Port.

Path: The path to your ParaView installation.

On Linux, the path will end with the top level folder into which ParaView was unzipped. For example, if you have a folder named ParaView on your desktop that contains the bin, doc, lib, and share folders, then your path would be /home/username/Desktop/ParaView.

On Mac, the path will end with the folder containing your ParaView.app. For example, if you have installed ParaView to your Applications folder, the path will simply be /Applications

Server Script Path: The full path to the http_pvw_server.py file, ending with the folder containing it. For example if the file is on your desktop, the path might be /home/username/Desktop.

Visualizer Port: The port number for the ParaView web visualizer. The

default value will be fine, but if you change it, then it must be different from the port number you provide for Port.

Remote OS: The operating system of the remote machine on which ParaView will be launched. If you want to launch ParaView on your local machine, ignore this cell. Otherwise, specify either "Linux" or "OSx".

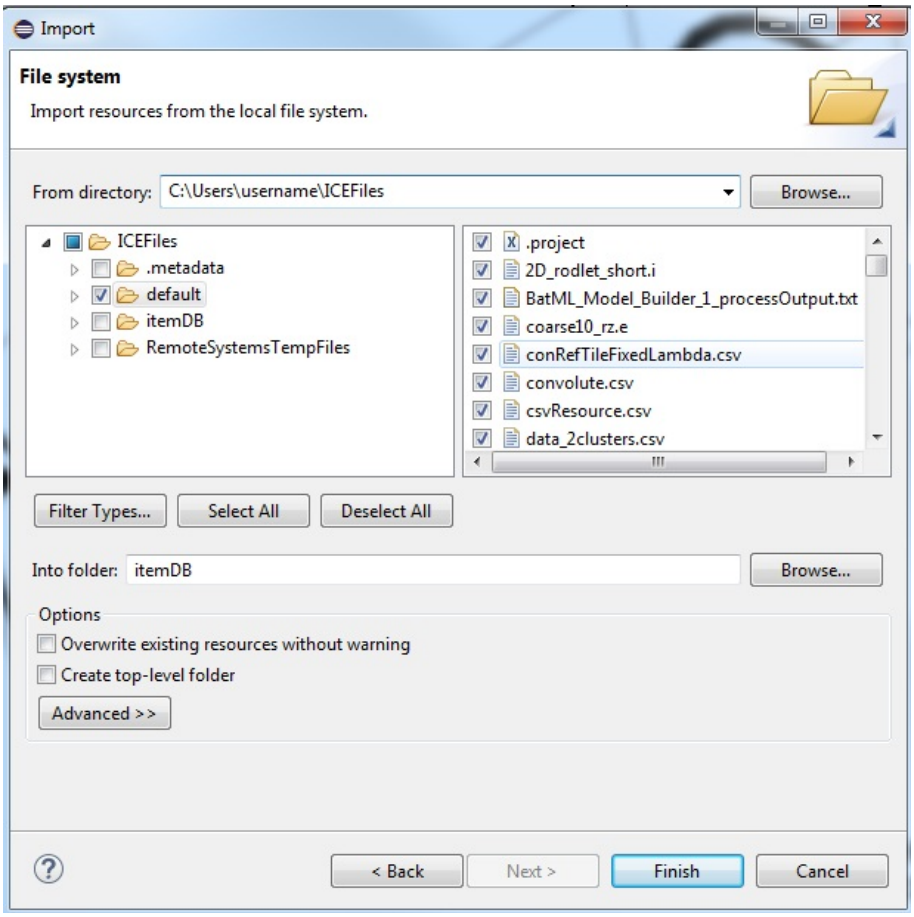
Remote ParaView Version Number: The version of ParaView you are using. This may be ignored unless you are launching a remote ParaView session on a Linux machine. You can check your installation's version number by looking inside the top level **lib** folder. It will contain a folder named paraview- followed by the version number.

Once finished editing the cells in the new row, press Apply, then OK. ICE will then launch the ParaView server and connect to it. If you are connecting to a remote machine, you will be prompted for permission to make the remote connection and asked for a password.

10.2 Opening a ParaView File

To open a ParaView Plot Editor, a file that uses this editor must first be placed in the Project Explorer. This view lists files imported into ICE. To access the Project Explorer, use the the menu bar at the top of the window and navigate to Window → Show View → Project Explorer. Depending on the active Eclipse perspective, opening this view may require selecting Other... and finding the Project Explorer in the dialog under the General category in the tree.

By default, the Project Explorer should automatically import the ICEFiles/default and ICEFiles/itemDB folders. If it does not, or if you want to import a different folder into ICE, right click in the Project Explorer and select Import... from the context menu. Then, select General → File System from the tree, and press the Next button. Select directories and/or files to import into the Project Explorer, and enter which folder they should be imported into, as shown below.

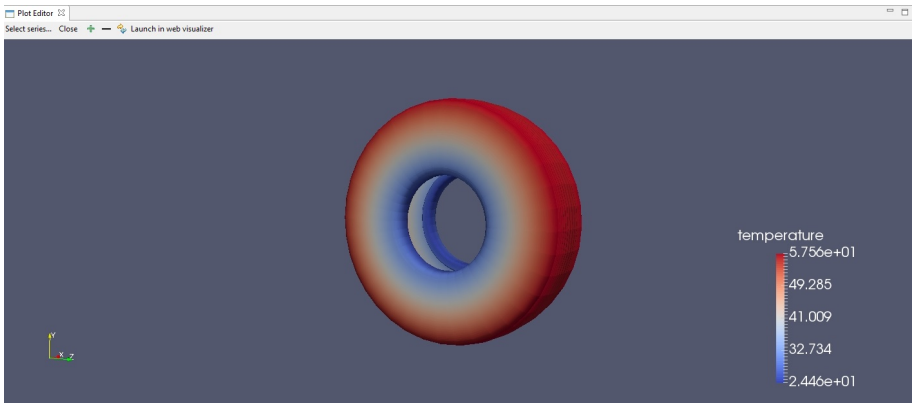


Once a file is in the Project Explorer, simply double click on it to open it in ParaView. Local ParaView connections can only open local files, while remote ParaView connections can only open files on the remote host.

10.2.1 Using ParaView

Camera Controls

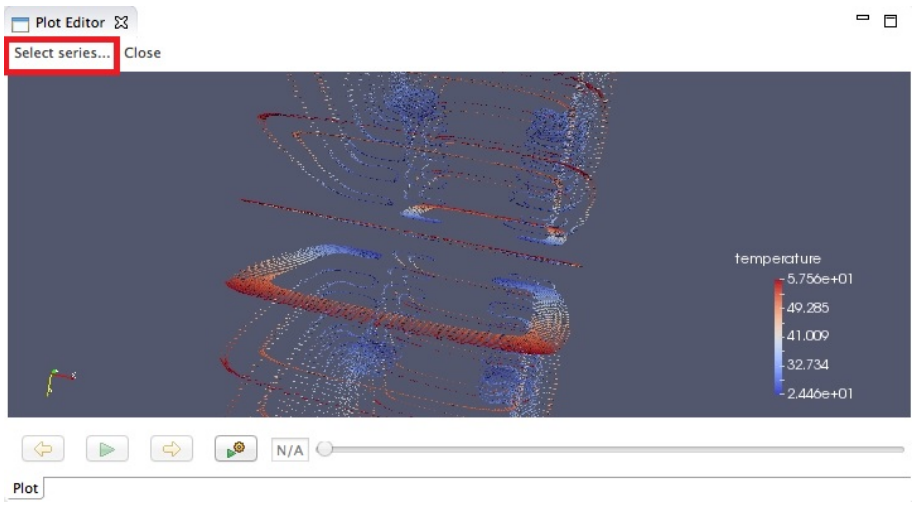
The Plot Editor allows the user to rotate the model by clicking and dragging inside the display area or adjust the zoom by scrolling the mouse wheel.



The buttons in the upper left can also be used to manipulate the camera. The green plus sign will zoom in, while the black minus sign will zoom out. The yellow and blue circular arrow button will reset the camera to its default position.

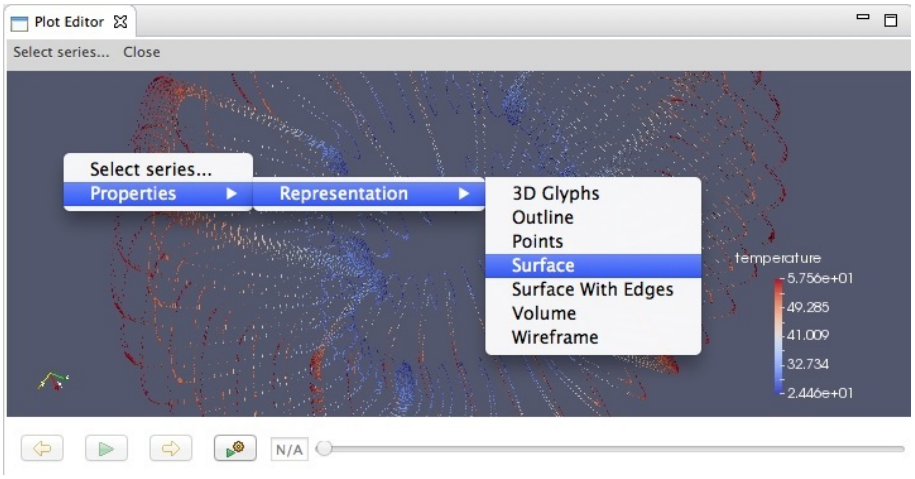
Selecting the Plot

Pressing the Select Series... button will open a dialog which lists the various plots in the opened file. Simply select one and click OK to open it.



Setting the Plot Representation

ParaView is capable of displaying plots in several different representations, such as points or surfaces. To switch between plot type, right click inside the Plot Editor's display area and select one of the listed options under the Representation category in the context menu.



Animation and Time Data

The Plot Editor features a time slider widget at the bottom of the screen.

The controls, in order of left to right, are:

1) Return to the previous time step.

2) Automatically play the plot as an animation by displaying the time steps sequentially.

3) Advance to the next time step.

4) Opens an options menu that allows the user to set the playback speed,

toggle whether the animation should loop when it reaches the end, and set the plot to the first or last time step.

5) A display for the current time step. It can be edited to set the plot to an arbitrary time step.

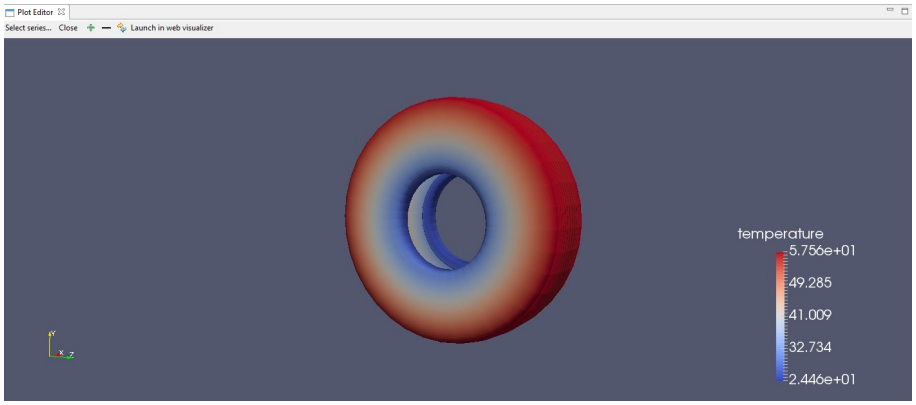
6) A slider that shows the current time step's position on the timeline. The slider can be dragged around the timeline, setting the plot's time step accordingly.

10.3 Accessing a ParaView Web Server

It is also possible to access the full ParaView web viewer application inside of ICE. This allows for full access to all the web viewer's features.

10.3.1 Accessing the Visualizer

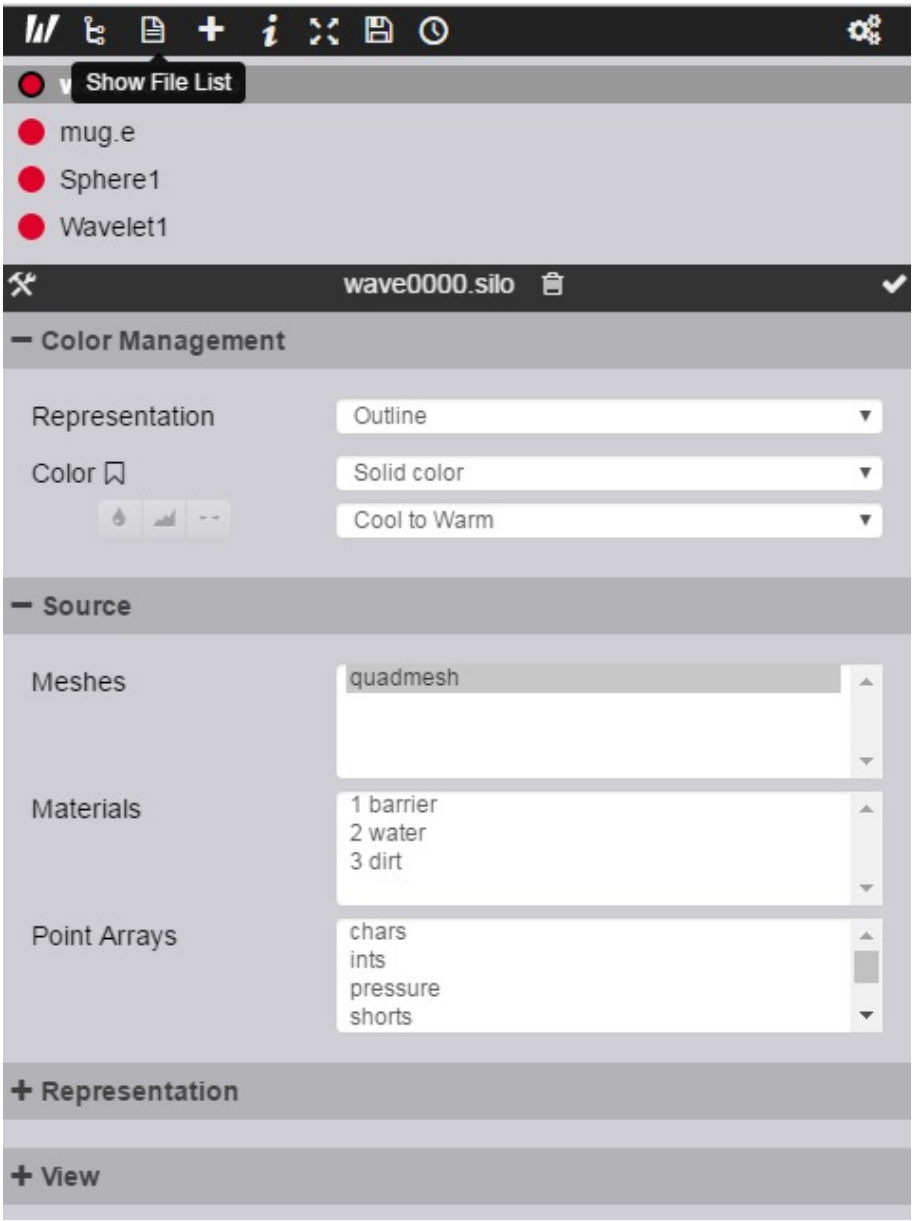
First, you must open a Plot Editor as described above.



Press the "Open in Web Visualizer" button in the top left to launch the web visualizer server and automatically open an internal web browser to view it.

10.3.2 Using the Visualizer

In order to load a data file, click the Show File List button as seen below.



This will place the contents of the data directory into the side bar. For sessions, the data directory will be your ICE workspace. For a remote connection, this will be the folder containing the remote file you are visualizing. You may then double click on a file to load it into the model.

You can click and drag the mouse to rotate the camera and right click fol-

lowed by dragging up or down to zoom. A full description of the visualizer's features is beyond the scope of this tutorial, but see the official documentation here.