

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).

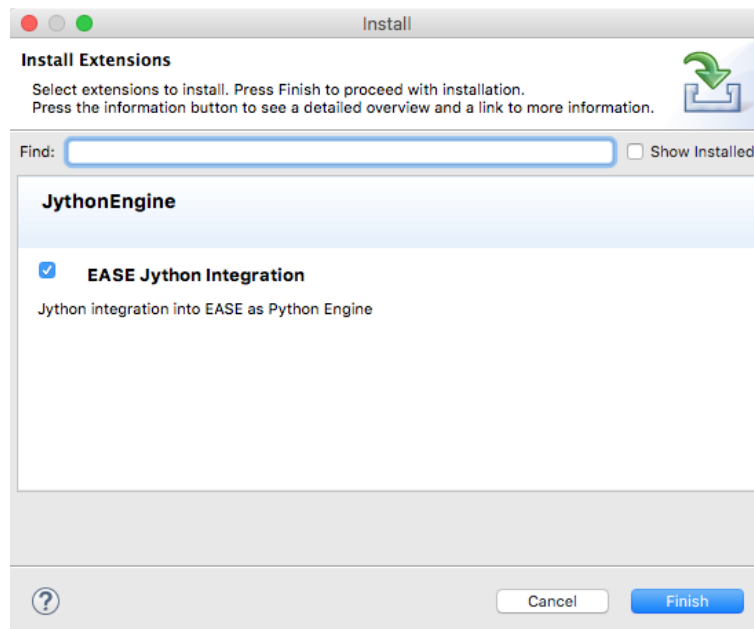
1 Installation and Configuration

Although EASE is pre-installed in the ICE application, there are a few additional components that need to be installed in order to provide a Python scripting engine.

1.1 EASE Jython Installation

The first step is to install the EASE Jython engine. This can be done via the official EASE repository¹ using the Eclipse Update Manager, but is more simply achieved using the Install EASE Components menu.

To installed the Jython engine, select **Help** → **Install EASE Components** from the ICE menu bar. Check the box next to the EASE Jython Integration entry and click Finish. Follow the prompts to install the component, and restart Eclipse when asked.

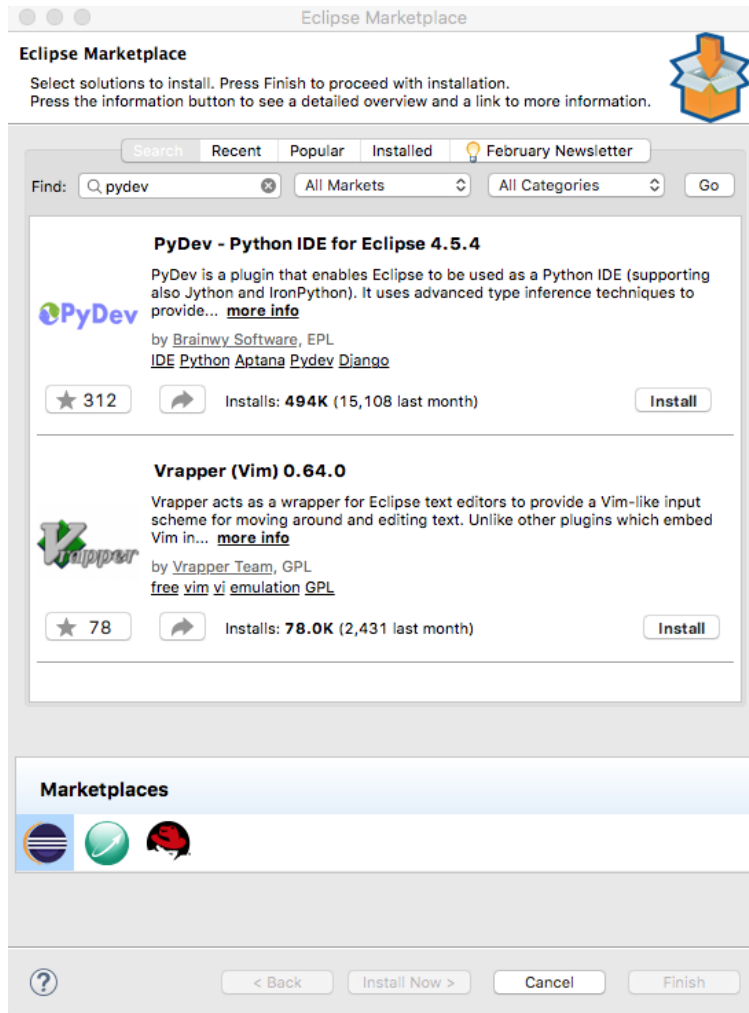


¹<https://dl.bintray.com/pontesegger/ease-jython>

1.2 PyDev Installation (optional)

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. 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.

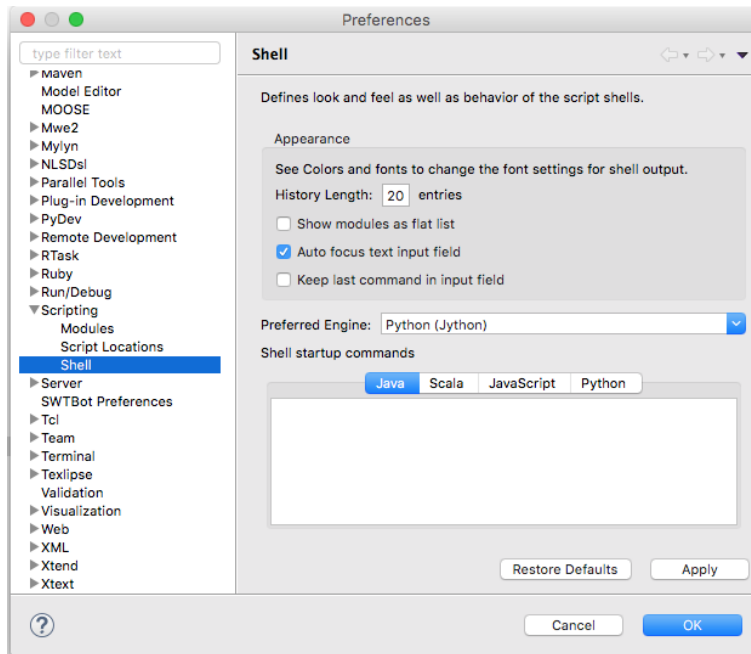
PyDev can be easily installed from using the Eclipse Marketplace client as follows. From the ICE menu bar, select **Help** → **Eclipse Marketplace...**, type “pydev” in the Find field, then click the **Go** button. After a few seconds, you should see an entry for PyDev. Simply click the **Install** button 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.



1.3 EASE Configuration

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. This is done through the Scripting preferences.

To set the script engine default, 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, then select **Shell**. Finally, select **Python (Jython)** from the **Preferred Engine:** dropdown, then click on **OK**.



2 Creating and Running Scripts

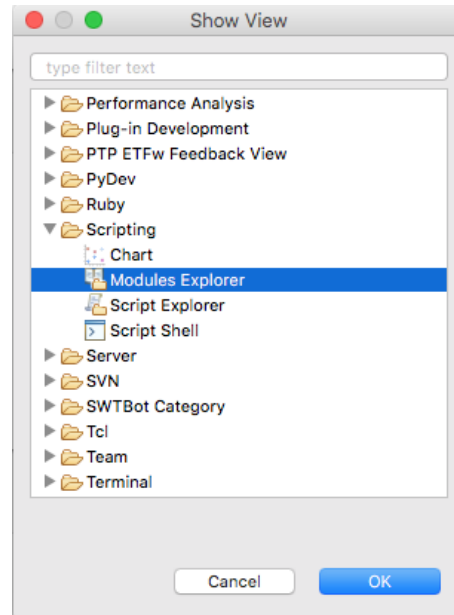
There is nothing special about creating and running EASE scripts. They can be created in a variety of ways using the development tools available in ICE, and then run later when needed. For this tutorial, we will be creating the scripts using PyDev (or any text editor) and running them directly using 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. For additional information on the **Scripting** perspective and other EASE features, see the EASE documentation².

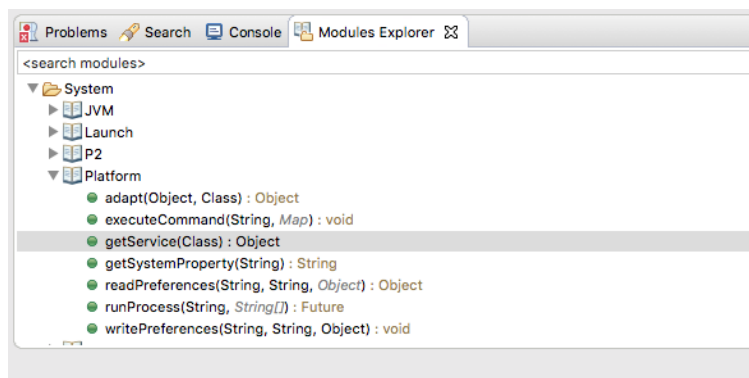
EASE hides many of the details that would normally be required to manipulate Java objects and perform actions in the Eclipse IDE. It does this by encapsulating typical actions into simple script commands that can be easily invoked from scripts that you write. These commands are collected together into “modules”. You can see which modules are available, and the commands that they contain, using the **Modules Explorer** view. This view can be opened by selecting **Window** → **Show View** → **Other...** to open the **Show View** dialog,

²<http://eclipse.org/ease/documentation>

as shown below.



Form this dialog, open the `Scripting` folder and select the `Modules Explorer`, then click on `OK`. This will open the `Modules Explorer` view which can be used to explore the modules and commands that are available for use in EASE scripts.

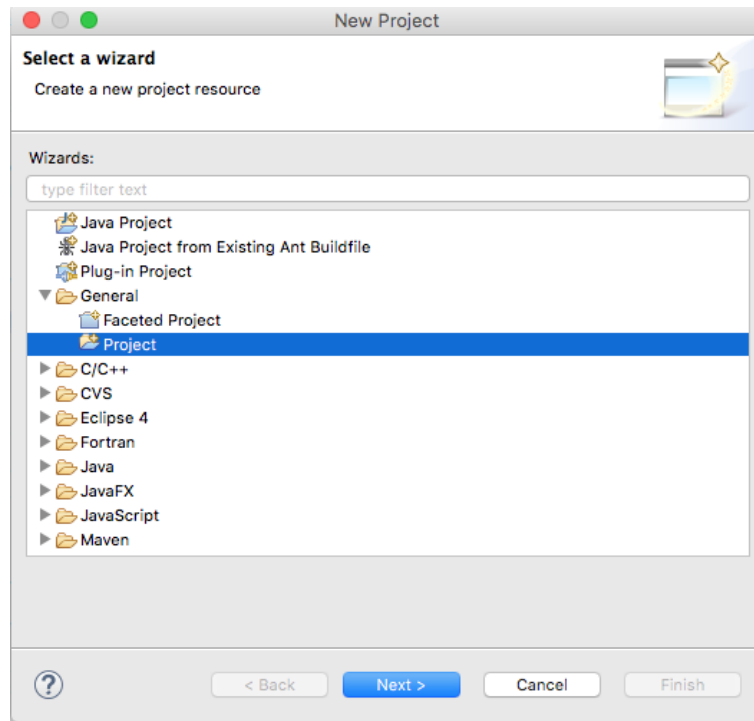


2.1 Creating a Python Script

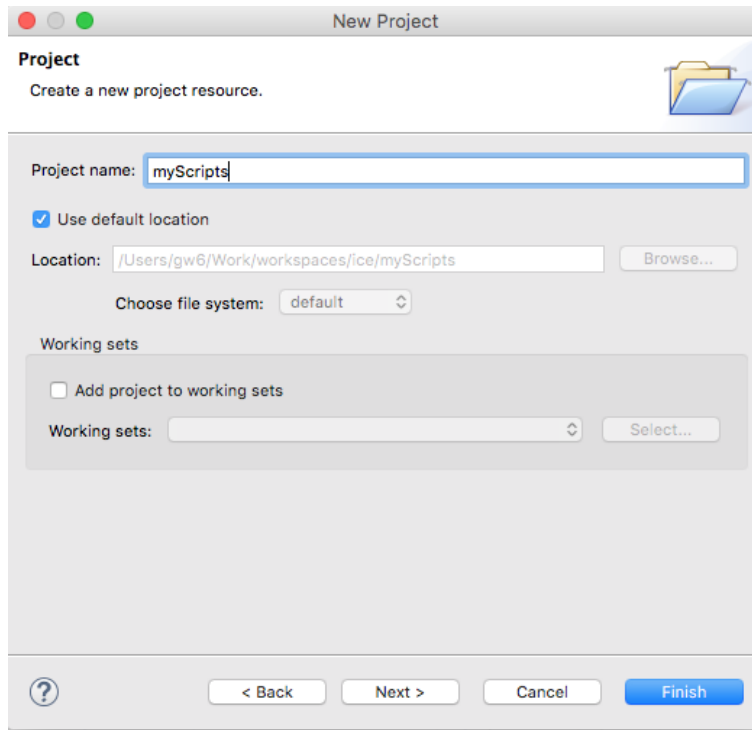
The easiest way to create a Python script is to simply create a text file ending in “.py” in an existing project. If your scripts will be used with stand-alone Python

programs, you can use PyDev to create a Python project, but in general, any kind of project will suffice.

First, let's create a project to hold the scripts. To do this, right click anywhere in the **Project Explorer** view and select **New** → **Project...** or click on the **New** button in the toolbar. Once you do this, you should see the **Select a wizard** dialog like the one shown below.

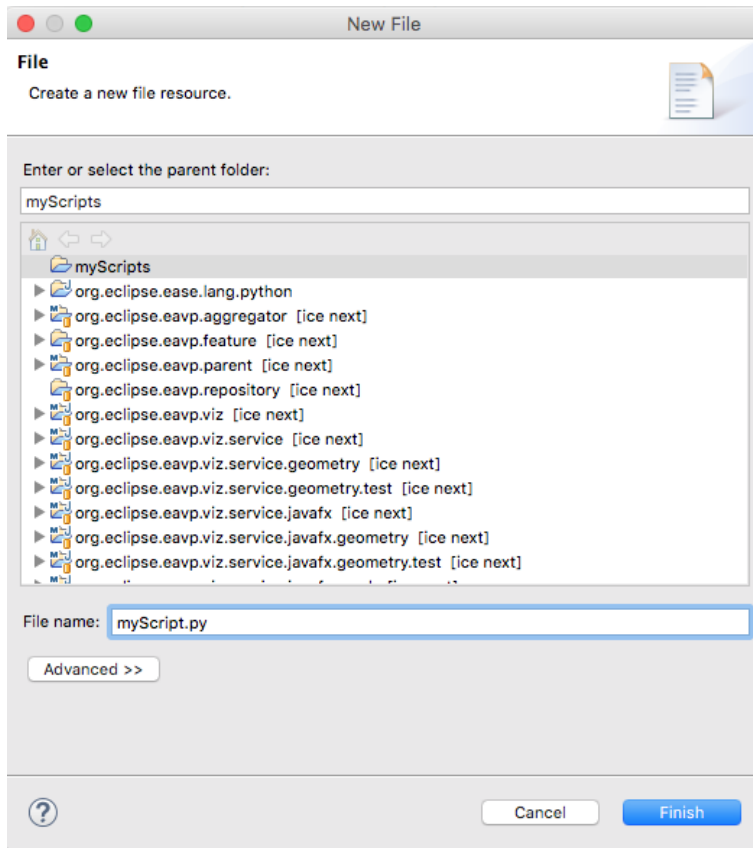


Open the **General** folder in the list of wizards, and select the **Project** wizard as shown. Then click on the **Next >** button which should open the **New Project** wizard. This is shown below.

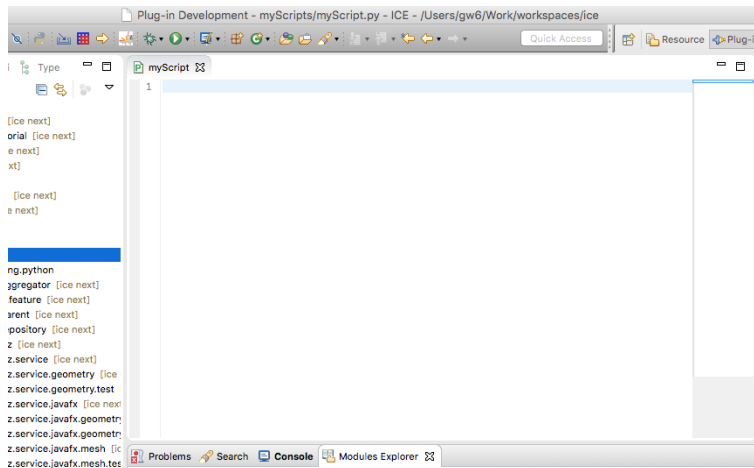


Enter a name for the project in the **Project name:** field and then click the **Finish** button. You can leave all the other options set to their default values.

You should now see a folder with the name you specified appear 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.



2.2 Writing a Python Script

For the purposes of this tutorial, we will just be using the `Platform` module. In order to load a module, we use the `loadModule()` function in Python. The argument to this function is a string representation of the module path, which in this case will be `/System/Platform`.

Enter the following command as the first line of the script file:

```
loadModule("/System/Platform")
```

Once this module has 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. This is done by adding the following line:

```
coreService = getService(org.eclipse.ice.core.iCore.ICore)
```

Once a reference to the core services has been obtained, we can use this to obtain a reference to the Reflectivity Model. This is done by adding the following line:

```
reflectModel = coreService.getItem(int(coreService.  
createItem("Reflectivity_Model")))
```

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()`.

In this example, we're just going to accept the default inputs for the model, so the only thing left to do is process the model. This is done using the core service `processItem()` as follows:

```
res = coreService.processItem(reflectModel.getId(), "
    Calculate_Reflectivity", 1)
```

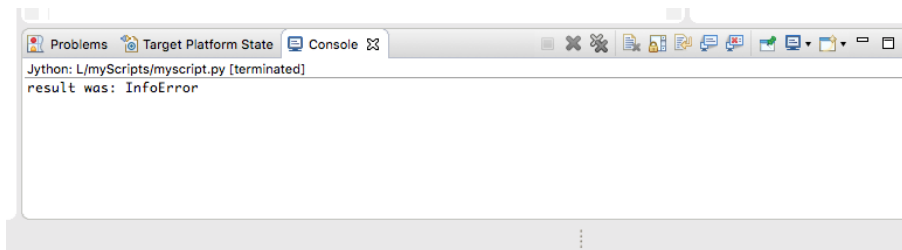
Finally, let's print out the result of processing the model to see if it was successful.


```
print "result was: %s" % res
```

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

2.3 Running a Python Script

Once you have created a Python script, it can be easily launched using the `Run As` context menu. Simply right-click on the Python file, then select `Run As` → `EASE Script`. EASE will automatically recognize the file type and run the script with the appropriate engine. Any (textual) output generated by the script will be displayed in the Console view shown below³.



Re-running the script is simple as ICE has automatically added the most recently executed run configuration to the run button. Just click on the  icon to re-run the last script. You can also click on the small triangle next to the button to see a list of recent launches and re-run them if desired.

³ In this case the `processItem()` function returns `InfoError` because it hasn't been correctly configured before the model was processed. We will see how to fix this later in the tutorial.

3 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 repository⁴, 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.

3.1 `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.

```
# *****  
# 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  
loadModel('/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 in  
#   t() to  
# convert it to an integer.  
reflectModel = coreService.getItem(int(coreService.createItem("Reflectivity_M  
odel")))  
  
# This is usually where you would do your own customization and automation re  
#   garding  
# the reflectivity model you just created. Maybe change the layers, or do som  
#   e custom  
# calculations.
```

⁴<http://github.com/eclipse/ice.git>

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

3.2 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
loadModel('/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 in
# t() to
# convert it to an integer.
reflectModel = coreService.getItem(int(coreService.createItem("ReflectivityModel")))

# 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(), "CalculateReflectivity", 1);
```

3.3 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 paramete  
# r  
# to show the effect that creates.  
# *****  
  
# Load the Platform module for accessing OSGi services  
loadModel('/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 dou  
# bled  
# 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 cre  
    # ate item  
    # method will return a string representing the number of that item, so us  
    # e int() to  
    # convert it to an integer.  
    reflectModel = coreService.getItem(int(coreService.createItem("Reflectivi  
    ty_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);
```

3.4 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 in  
#   t() to  
# convert it to an integer.  
reflectModel = coreService.getItem(int(coreService.createItem("Reflectivity_M  
odel")))  
  
# 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 i  
#   t. Clearing  
# the list requires the locks as multiple operations are happening and we nee  
#   d to  
# protect the list from multiple threads trying to access it at the same tim  
#   e.  
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(), "CalculateReflectivity", 1);

```