Chapter 1

The Eclipse ICE Item Project Generator

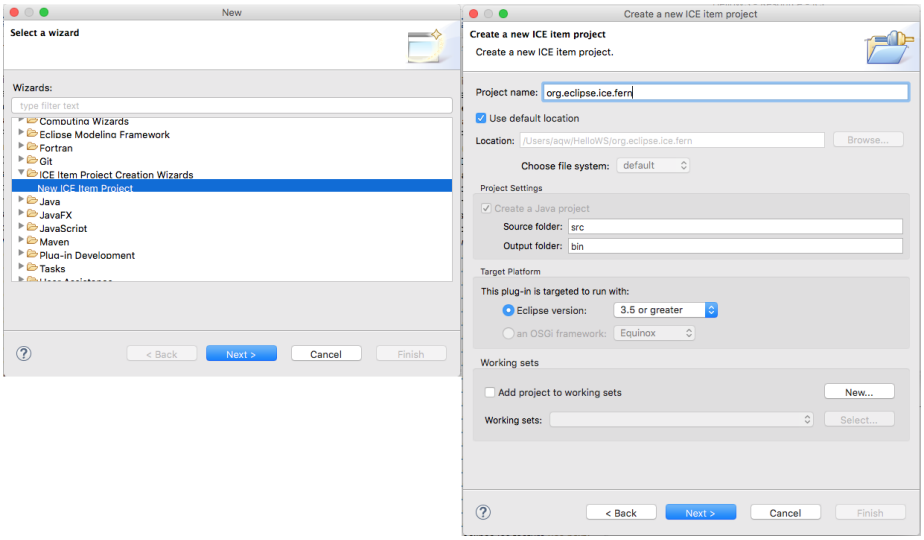
1.1 Overview

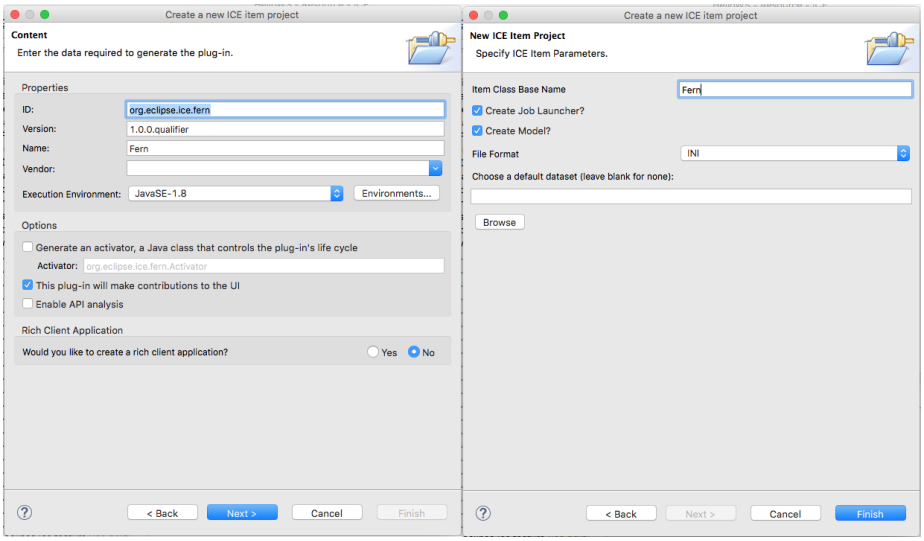
This tutorial will teach you how to create your own ICE Items via the built in tools within ICE. To demonstrate these tools, we will walk through the development of an ICE Item project for the FERN code, a fast, efficient nuclear reaction network solver.

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.

1.2 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.

1.3 Adding Functionality to the New Items

1.3.1 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.

1.3.2 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.

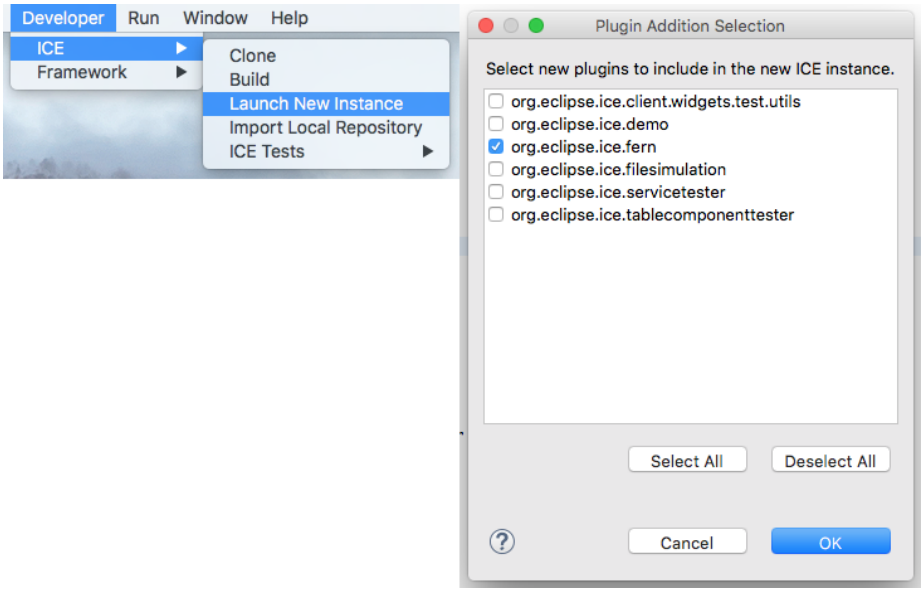
We also need to inform the JobLauncher what other files are involved in this execution. To do that, the JobLauncher provides an addInputType() method. Add the following to setupForm():

```
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.

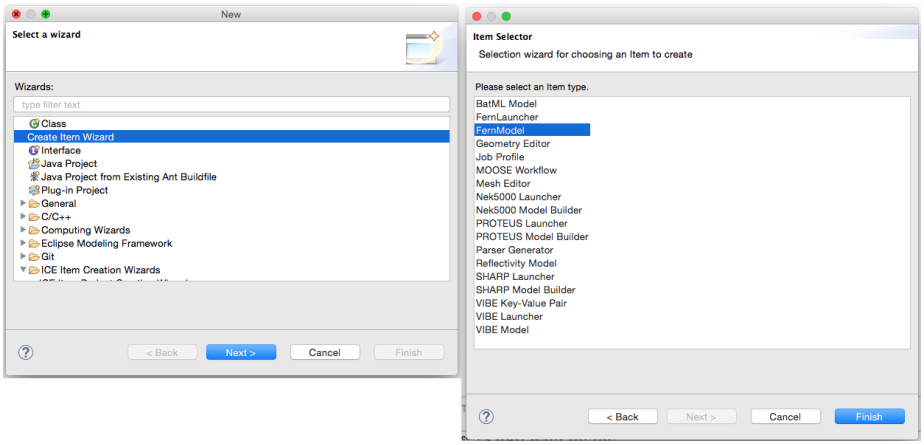
1.4 Using the New Fern Items

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.

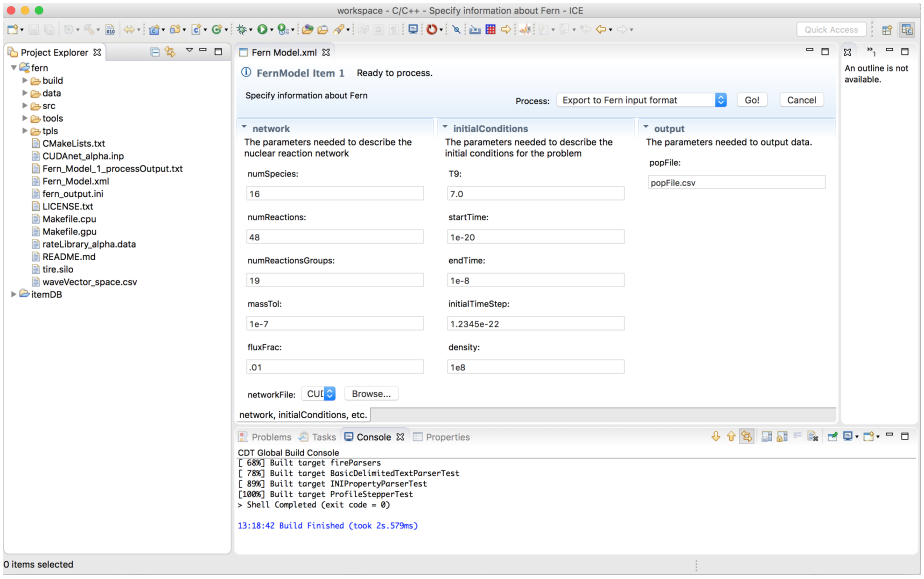


Select `org.eclipse.ice.fern` and click Ok. This will create and launch a new instance of ICE that includes your custom Item plugin.

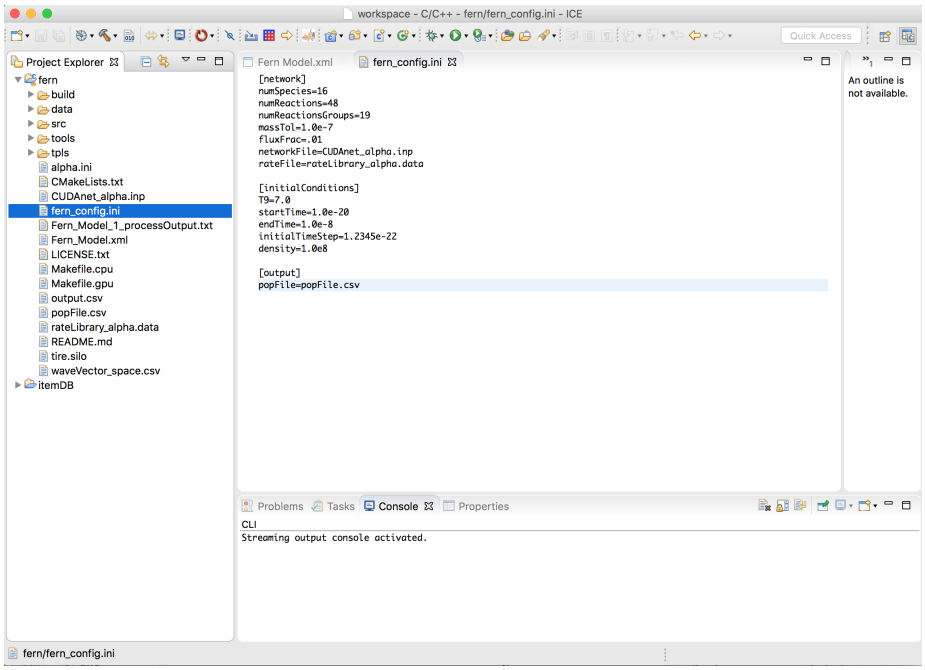
With a new ICE instance open, close the Welcome view if necessary and go to `File` > `New` > `Other` and select the Create Item Wizard.



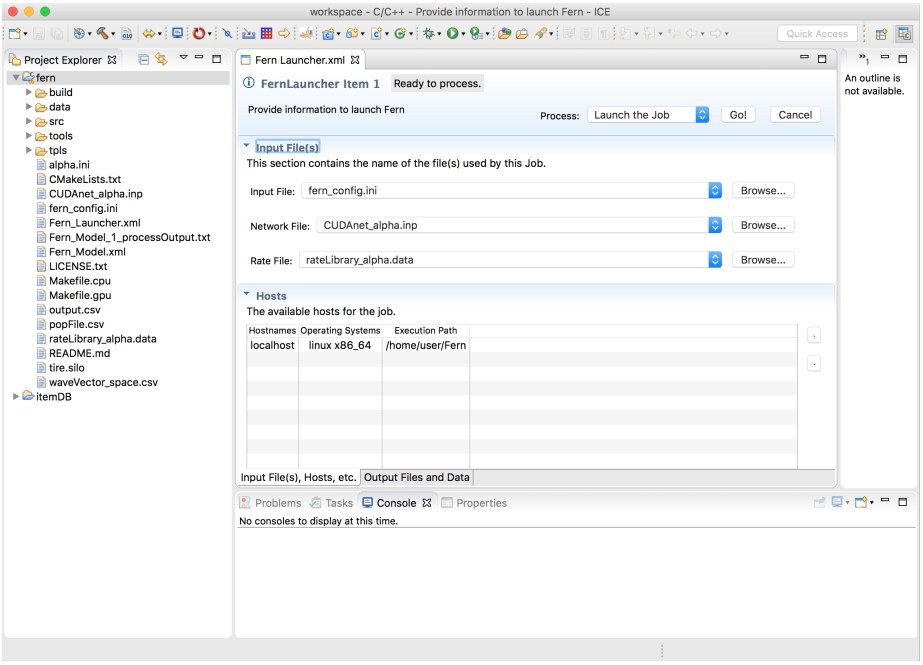
After selecting the FernModel Item, you will be presented with the view in the figure below.



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.



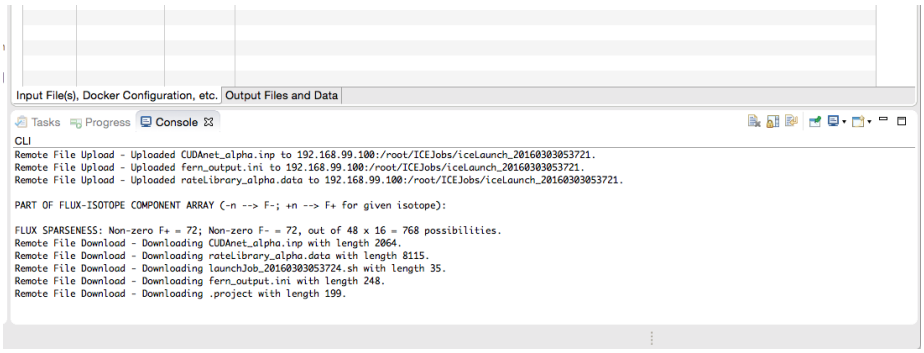
Now you can similarly create a new Fern Launcher. After creating the Launcher, you should see a view like below.



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.