Populus Guide for Developers

Lars Roe

August 2, 2014

Contents

Part I
Overview

Part II
Java Source Code

1 Models

1.1 Files

By convention, each end model (not meant to be inherited from) should be in the package `edu.umn.ecology.populus.model.`*`ModelName`*.

1.1.1 Model

A `Model` holds together the basic parts of a model.

1.1.2 ModelPacket

A `ModelPacket` is a simple wrapper for a model so we can refer to one class at a time, and used in making the menus. The menus are created in `initializeMenuPackets()`, and this is manually updated to add or remove models.

1.1.3 ModelPanel

The `ModelPanel` (input window) base files are in `edu.umn.ecology.populus.edwin` (short for editor window, from the Pascal DOS program's naming conventions).

`registerChildren()` looks at all of the components, and sets event listeners where appropriate. Read Events for more information.

1.1.4 ModelOutputPanel

The `OutputPanel` (output window) base files are in `edu.umn.ecology.populus.resultwindow`

1.2 Events

When changes in the input panel occur, events - or messages - are sent to the output. The `ModelPanel` will call `fireModelPanelEvent()` whenever a change occurs, with a constant such as `CHANGE_PLOT`. If this warrants a new output, `ModelPanel` will be queried for, in the case of Basic Plot, new plot info.

Do not assume that `getPlotInfo()` will be called whenever you call `fireModelPanelEvent`. For example, if changing the value of a radio button should disable another parameter, that should be done separately from `getPlotInfo()`. See the method `modelPanelChanged()` to see which events are ignored and which events create a new plot.

Inherited models should not have to worry about when to show the output screen. `registerChildren()` is called after the initialize of the front panel, and this routine looks at all of the components and adds listeners to the ones that should through events. There is a setting in the User Settings so that users can change when to automatically update the output and making decisions on a model-by-model basis will not work with this.

1.3 Adding a Model to the Menus

To add a model to the menu, add a ModelPacket in PopPreferences.

I dreamed of one day being able to dynamically modify these models. Maybe we could load a file `Model` class on the fly and it would be included in the top-level menu for that session. Or even store it in the preferences. But we haven't had much of a need, and Don would've preferred the simpler one-size-fits-all approach.

TODO - What is SelectModelDialog??

1.4 Basic Model

Most models will derive from `BasicPlotModel`, in the plot directory.

1.5 Common Variants

Most models extend from `edu.umn.ecology.populus.plot.BasicPlotModel`, which does basic graphing. But you don't have to do this. See `Woozleology` for an example of one that does not extend from this.

2 Main

`main` is found in `edu.umn.ecology.populus.core.PopRun`. The `DesktopWindow` is the primary GUI background to the application.

3 Help

When we click the Help button on a model or the main DesktopWindow, we call `HelpUtilities::displayHelp()`.

When we click on the Help button within a model, it's very similar, but we use the `getHelpId()` from the model to get a Named Destination into the PDF file.

The help system was changed dramatically in 5.5, by modifying the local help file to use the language specified by the user's configuration.

TODO - I suspect that getModelHelpText() doesn't really do anything these days. Maybe we can gut that.

4 Preferences

4.1 PreferencesFile

The file for keeping state is stored as `userpref.po` in the user's home directory (as of Populus 5.4). It is loaded during initialization. By default, it is in the user's home directory – not in Populus's – because we aren't guaranteed write permission for all systems. This can be overrided by the startup command - see README.config.

Almost all of the code is in `PopPreferences`.

5 GUI Widgets

5.1 ParameterField

The ParameterField was originally concocted as a spinner. But then we added the variable name, and variable information to the parameter. I like to use this with WindowBuilder (more details later).

5.2 JClass

JClass includes the chart software for Java that we use. The Manifest file in the JAR file they included has some bogus `dependson` lines that give warnings when you try to run. I manually deleted these, and just keep this new version around. JClass keeps switching companies. We have an old version of their product, and I don't have any reason for upgrading.

6 Javadoc

I wish the code were better documented. But you can still use `javadoc` to generate documentation for the files.

Part III
Installer

Populus Splash Screen. We have a file called Populus*.*.psd which is a photoshop file describing the title screen. For a new release, we probably want a change in version number, so make a new .psd file with the new version, and then export it to gif format (calling it `PopulusSplashScreen.gif`) and replace the one in edu/.../core/ with the new gif.

Part IV
Web Page

This should all be handled by the UMN Web team these days. They now use Drupal (a content management system). For 5.5, I just gave them a new JAR file.

Part V
Test and Verification

7 Release Checklist

Check that help works on all different platforms.

current issues for troubleshooting help file: on mac os x: the populus parameter field arrows are dim screen resolution can cause windows to be smaller than they should be - just resize on pc:

8 Platform

It's a good idea to test on different platforms.

8.1 Linux

LiveCD SLAX can boot up Linux on an otherwise Windows computer. There are other options now too.

8.2 MacOS

You really just need a Mac for this. The UofM computer team have testers to help with this.

Part VI
Setting Up New Development Machine

9 Development Software

9.1 Java Development Kit

Download and install Java SE (Standard Edition) from oracle.com. Please use JDK 1.7.

9.2 Git

Git on the command line should be default for OS X and Linux. You don't have to install more, but `http://git-scm.com/downloads/guis` has some nice GUIs. I used GitX-dev (rowanj) for OS X, which seems good.

For Windows, I like git for Windows: `http://msysgit.github.io/`, which includes the command line tools and GUI.

9.3 Eclipse

Download the Eclipse Standard from `https://www.eclipse.org/downloads/`. (You can actually install any version that has Java support.) The "installer" is just a zip file that you extract somewhere. You'll run it by running the executable in there.

9.4 TeX

We don't use LaTeX for any externally-facing file, but it is used for modifying this document. I use MiKTeX for Windows. MacTeX and livetex are recommended for OS X and Linux respectively.

9.5 Photoshop

Use Photoshop to make the pictures for, say, the Web page. There are saved `.psd` files around that contain the source image to work from with its Layers.

10 Populus-specific Setup

10.1 Files

Here's how you can check out the files from git. Assume that the `.git` directory is at `C:/TEMP/pop.git` and you want to put the code into `workspace/pop` relative to your current directory.

```
mkdir workspace
cd workspace
git clone file:///C:/TEMP/pop.git pop
```

10.2 Running Eclipse

Now run Eclipse. For the workspace, choose the `pop` directory, or whatever you used to extract the files from git in the previous step.

Be sure that you are using an installed JDK for the workspace (`Windows` ⇒ `Preferences` ⇒ `Java` ⇒ `Installed JREs`)

Go to `File` ⇒ `New` ⇒ `Java Project`

For the project name, choose `PopulusE`. Eclipse should know that this is an existing project, and don't set any more options.

Click the green run button. You want to run this as a Java Application. The main class is `PopRun` (`edu.umn.ecology.populus.core.PopRun`).

10.3 WindowBuilder

You'll want to install the WindowBuilder plugin to Eclipse if you plan to edit any of the screens. Go to `http://www.eclipse.org/windowbuilder/download.php` for instructions.

To use WindowBuilder, right click on a Panel file in the Package Explorer, then choose Open With... and select WindowBuilder Editor.

Most special build steps are specified in the Ant file (`fullbuild.xml`). Right-click the file, and select `Run As` ⇒ `Ant Build...` and select the `bundle_populus` option, and run it.

Part VII
How to add a new model

11 Example of a new model: Fibonacci rabbits

We'll look at a simple model idea and the steps needed to incorporate it into a model.

11.1 Description of the model

Fibonacci once posed the following question:

> Suppose a newly-born pair of rabbits, one male, one female, are put in a field. Rabbits are able to mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits. Suppose that our rabbits never die and that the female always produces one new pair (one male, one female) every month from the second month on. How many pairs will there be in one year? (from `http://fibonacci.uni-bayreuth.de/project/fibonacci-and-the-rabbits/the-story.html`)

Now, let's code!

11.2 Create package

From the Explorer window, select `File` ⇒ `New` ⇒ `Package`. Use `edu.umn.ecology.populus.mode` for the package. By convention, models are in a package/directory just under `edu.umn.ecology.populus.model`.

11.3 FRParamInfo

I think it's easier to think of what data will be taken from input screen. In this case, we just need the number of months, or generations, to run.

Right-click on the new package and select `New` ⇒ `Class`. Type in `FRInfo` for the name Add in the `Interface` `edu.umn.ecology.populus.plot.BasicPlot`. Press `Finish`.

Create a constructor that takes as input the number of generations. You should implement code here that creates a new `BasicPlotInfo` as a field.

Implement `getBasicPlotInfo()`, which will return a `BasicPlotInfo` object.

If you are creating a more-complicated model, you will want to create a `FRData` class that aggregates the data that you need to pass from the panel.

Your code should look something like this:

```
package edu.umn.ecology.populus.model.fibrabbits;

import edu.umn.ecology.populus.plot.BasicPlot;
import edu.umn.ecology.populus.plot.BasicPlotInfo;

public class FRInfo implements BasicPlot {
    private BasicPlotInfo bpi;

    public FRInfo(int maxGens) {
        bpi = new BasicPlotInfo();
        bpi.setMainCaption("Fibonacci Rabbits");
        bpi.setXCaption("Generation");
        bpi.setYCaption("Pairs of Rabbits");
        bpi.setIsDiscrete(true);

        //Generate Data
        double data[][][] = new double[1][2][maxGens+1]; //1 line with 2 variables
        double newbornPairs = 1.0;
        double maturePairs = 0.0;
        for(int gen = 0; gen <= maxGens; gen++) {
            data[0][0][gen] = (double) gen;
            data[0][1][gen] = newbornPairs + maturePairs;
            double prevNewbornPairs = newbornPairs;
            newbornPairs = maturePairs;
            maturePairs += prevNewbornPairs;
        }
        bpi.setData(data);
    }

    @Override
    public BasicPlotInfo getBasicPlotInfo() {
        return bpi;
    }
}
```

11.4 FRPanel

This is the input screen. Right-click on the new package and select `New` ⇒ `Class`. Type in `FRPanel` for the name Type in `edu.umn.ecology.populus.plot.BasicPlotInputP` for the Superclass (or use the browse button) Press `Finish`

Close the tab, then re-open it with WindowBuilder. You don't have to use WindowBuilder, but it definitely makes it easier. Click on the `Design` tab for the WYSIWYG designer of the window. We'll want to use a `PopulusParameterField` here for selecting values. If it is not yet in the WindowBuilder field, right-click on your menu of choice in the Palette and select `Add Component...`. Choose a name of your choice (I use PPField) and use `edu.umn.ecology.populus.visual.ppfield.Populu` for the Component.

Now click on the `PPField` in the `Palette` then click into the panel to insert it there.

In the properties window, set the `currentValue` and `defaultValue` to `10.0`. Set `helpText` to a long description, like `Total number of months for rabbits to grow` (this is the hover text). Set `integersOnly` to `true`, since we only want to allow an integer value (even though the underlying model uses floating point). Set `parameterName` to `months`. Set `minValue` and `maxValue` to something reasonable like `1.0` and `200.0`, respectively.

Now switch back to the Source view tab.

At the end of the constructor, add the following line so that user inpuut events will trigger plot updates:

`this.registerChildren(this);`

Now implement `getPlotParamInfo()`, which should return an object of type `FRInfo`.

Implement `getOutputGraphName()`, which will return a string for the main title of the output window.

Your code should like this:

```
package edu.umn.ecology.populus.model.fibrabbits;

import edu.umn.ecology.populus.plot.BasicPlot;
import edu.umn.ecology.populus.plot.BasicPlotInputPanel;
import edu.umn.ecology.populus.visual.ppfield.PopulusParameterField;

public class FRPanel extends BasicPlotInputPanel {
    private static final long serialVersionUID = -982727645471238633L;

    private PopulusParameterField maxGenerations;

    public FRPanel() {
        maxGenerations = new PopulusParameterField();
        maxGenerations.setMinValue(1.0);
        maxGenerations.setMaxValue(200.0);
        maxGenerations.setHelpText("Total number of months for rabbits to grow");
        maxGenerations.setParameterName("months");
        maxGenerations.setIntegersOnly(true);
        maxGenerations.setDefaultValue(10.0);
        maxGenerations.setCurrentValue(10.0);
        add(maxGenerations);
        this.registerChildren(this);
    }

    @Override
    public BasicPlot getPlotParamInfo() {
        return new FRInfo(maxGenerations.getInt());
    }

    @Override
    public String getOutputGraphName() {
        return "Fibonacci Rabbits";
    }
}
```

11.5 FRModel

Now create `FRModel`. Its Superclass is `edu.umn.ecology.populus.plot.BasicPlotModel`.
Implement `FRModel()` to set the modelInput to a new `FRPanel`.
Implement `getModelName()` to return the model name.
Don't worry about implementing `getModelHelpText()` and `getHelpId()`

at this stage. These functions are so that users looking for help will go to the context-specific section of the help pdf.

Your code should look like this:

```
package edu.umn.ecology.populus.model.fibrabbits;
import edu.umn.ecology.populus.plot.BasicPlotModel;

public class FRModel extends BasicPlotModel {
    public FRModel() {
        this.setModelInput(new FRPanel());
    }

    public static String getModelName() {
        return ("Fibonacci Rabbits");
    }
}
```

11.6 Res

You may want to create a Res file that should be used for storing all of the String resources. See how other models use it.

11.7 Add model to the menu

If you can shoehorn this into an existing menu group, it's quite easy. Just go to `PopPreferences::initializeMenuPackets()` and add a single line with the new `ModelPacket` accordingly. If you wanted this near in the single-species dynamics menu, add this in the initialization list of `singleModels`:

```
new ModelPacket( edu.umn.ecology.populus.model.fibrabbits.FRModel.class ),
```

If you have to create a new `ModelPacket` array, you'll need to also add code to `DesktopWindow`.