

Notes on running SNAPSim (on a Linux system)

Michael Richmond

Feb 6, 2007

Contents:

- [Preparing to run Eclipse](#)
 - [Synchronizing source code with the repository](#)
 - [Running a simple piece of code](#)
 - [Finding datafiles written to disk](#)
 - [Using the SNAPSim data browser to examine datafiles](#)
 - [Modifying and running Java code](#)
 - [For more information](#)
-

Preparing to run Eclipse

- Start a new shell
- Use ssh-agent to give SSH a "memory" -- which will allow you to upload/download code via CVS without having to provide a password every single time

```
richmond@spiff:~  
[richmond@spiff ~]$ xterm &  
[1] 11800  
[richmond@spiff ~]$ █
```

```
richmond@spiff:~  
[richmond@spiff ~]$ ssh-agent bash  
[richmond@spiff ~]$ ssh-add  
Identity added: /home/richmond/.ssh/id_rsa (/home/richmond/.ssh/id.  
Enter passphrase for /home/richmond/.ssh/id_dsa:  
Identity added: /home/richmond/.ssh/id_dsa (/home/richmond/.ssh/id.  
[richmond@spiff ~]$ █
```

- Set shell environment variables so that your computer will be able to talk to the remote CVS repository

```
[richmond@spiff ~]$  
[richmond@spiff ~]$  
[richmond@spiff ~]$ export CVS_RSH=ssh  
[richmond@spiff ~]$ export CVSROOT=cvsuser@build.snap.nersc.gov:/cvs  
[richmond@spiff ~]$  
[richmond@spiff ~]$  
[richmond@spiff ~]$
```

- Run the eclipse executable program. On my computer, I need to set the PATH explicitly so that the proper version of Java will be chosen to run eclipse.

```
[richmond@spiff ~]$  
[richmond@spiff ~]$ export PATH=/usr/java/latest/bin:$PATH  
[richmond@spiff ~]$ /home/richmond/eclipse/eclipse
```

At this point, the Eclipse environment should start and give you a big window with a number of panes.

File Edit Navigate Search Project Run Window Help



Package Explorer Hierarchy

- ▷ snap.jar.release [build.snap.nersc.gov]
- ▷ snap.snoop.framework3 [build.snap.nersc.gov]
- ▷ snap.snoop.physics.calibration [build.snap.nersc.gov]
- ▷ snap.snoop.physics.cosmologyfitter [build.snap.nersc.gov]
- ▷ snap.snoop.physics.lcfitter [build.snap.nersc.gov]
- ▷ snap.snoop.physics.lcfitterv2 [build.snap.nersc.gov]
- ▷ snap.snoop.physics.mission [build.snap.nersc.gov]
- ▷ snap.snoop.physics.mufitter [build.snap.nersc.gov]
- ▷ snap.snoop.physics.observatory [build.snap.nersc.gov]
- ▷ snap.snoop.physics.pixel [build.snap.nersc.gov]
- ▷ snap.snoop.physics.simulation [build.snap.nersc.gov]
 - ▷ snap.snoop.physics.simulation
 - ▷ domainBased
 - ▷ launch
 - ▷ nouns
 - ▷ parameters
 - ▷ runfiles
 - ▷ samples
 - user.properties 1.1 (ASCII -kqv)**
 - ▷ studies.aperture
 - ▷ targetBased
 - ▷ verbs
 - .classpath 1.2 (ASCII -kqv)
 - .project 1.1 (ASCII -kqv)
 - ▷ snap.snoop.physics.sourcedata [build.snap.nersc.gov]
 - ▷ snap.snoop.physics.universe [build.snap.nersc.gov]
 - ▷ snap.snoop.physics.util [build.snap.nersc.gov]
 - ▷ snap.snoop.runfile [build.snap.nersc.gov]

```

1###
2### Configuration for SNAP Default Parameters
3###
4SNAPDefaults.TargetSeed = 5000
5SNAPDefaults.SNSeed = 6000
6SNAPDefaults.DustSeed = 7000
7SNAPDefaults.MaxTargets = 10
8#
9#
10###
11### Configuration for GarySim Default Parameters
12###
13GarySim.MaxTargets = 15
14#
15#
16###
17### Configuration for Aperture
18###
19Aperture.ExposureTime = 310
20Aperture.MaxTargets = 20
21

```

Declaration Problems Javadoc Search Console

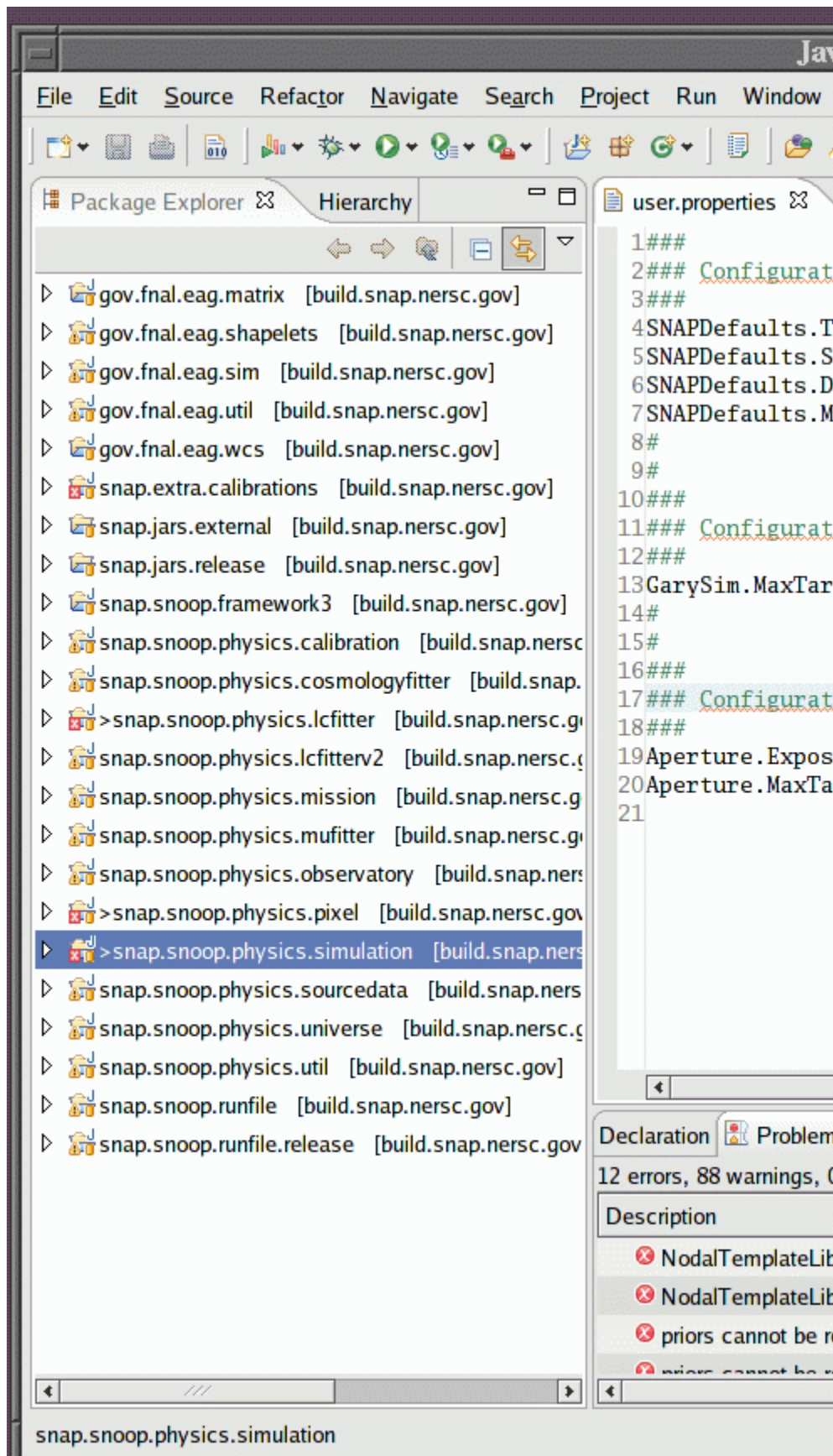
0 errors, 0 warnings, 0 infos (Filter matched 0 of 5034 items)

Description

Writable Insert 1 : 1

Synchronizing source code with the repository

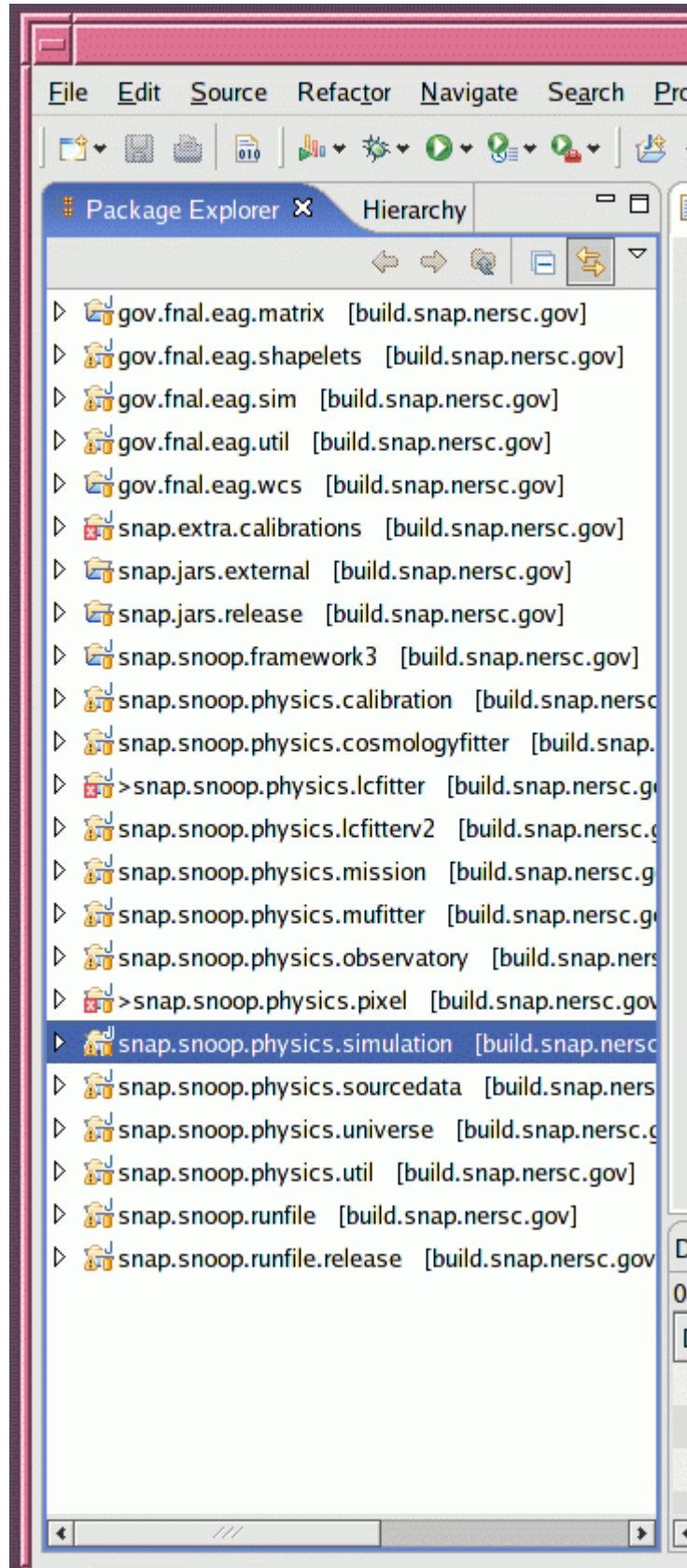
Other people may have modified source code since you last ran SNAPSIm. In order to grab the latest version of one package, go to the *Package Explorer* pane and highlight an item. I'll choose **snap.snoop.physics.simulation**



Right-click on the item, and choose **Replace With ...** and **Latest from HEAD**.

If this is the first time you have requested access to the remote code repository during this session, you'll be asked for a password. A window or two may pop up to show the status of the code transfer.

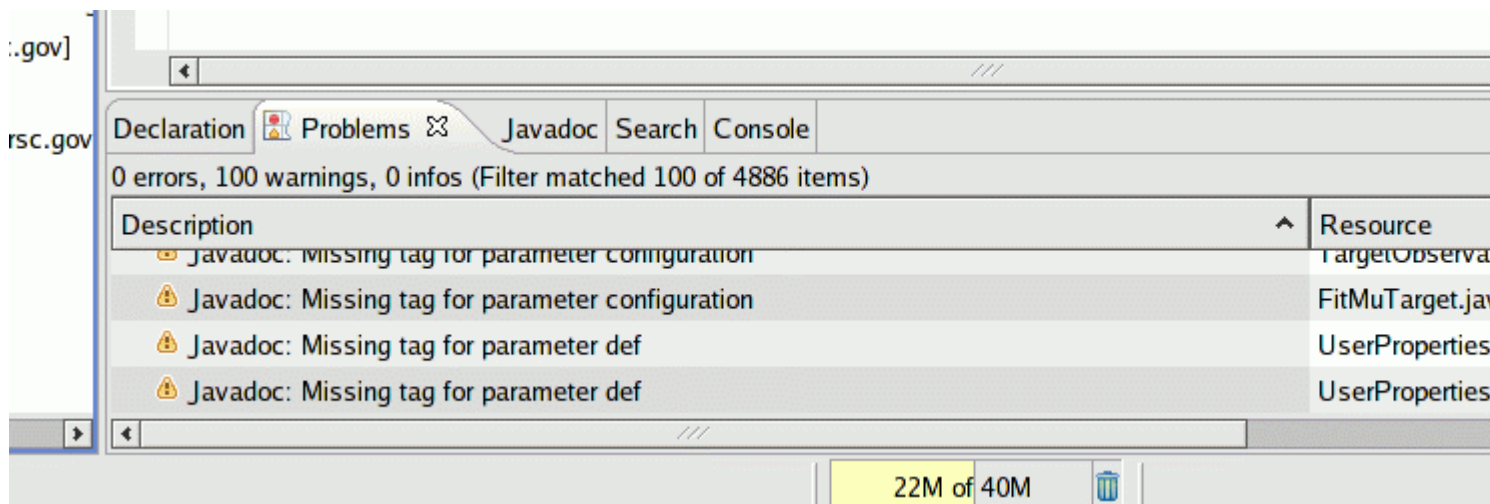
When the transfer has finished, the package should no longer show the little red "X" in the *Package Explorer* pane.



Right now, although the red "X" is gone, my pane shows a yellow "Warning" sign:



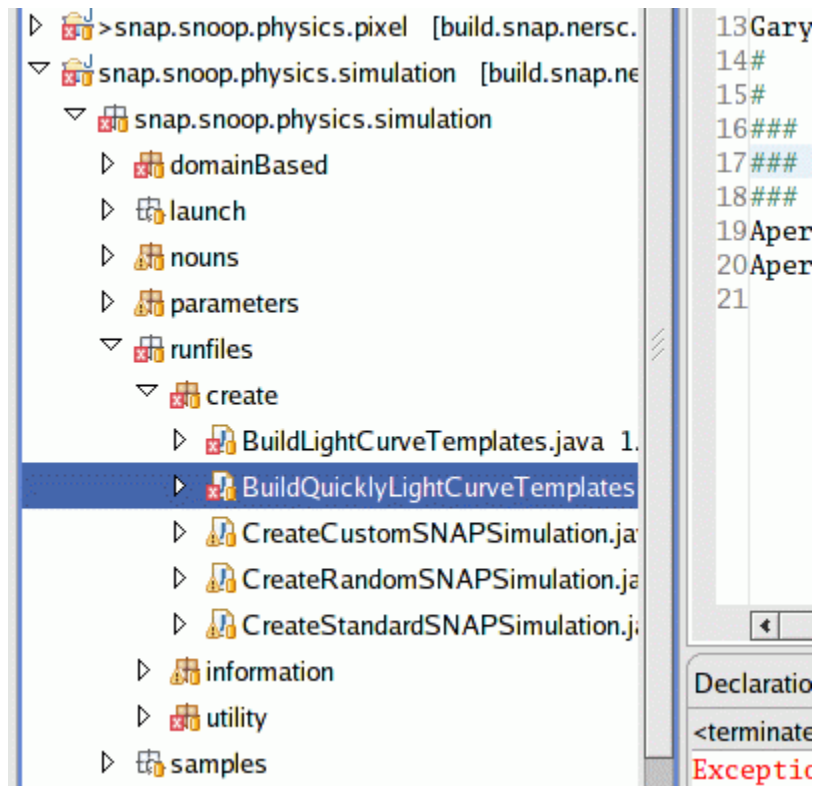
Looking down in the *Problems* pane of the Eclipse window, I see lots of warning messages about the code (but no errors).



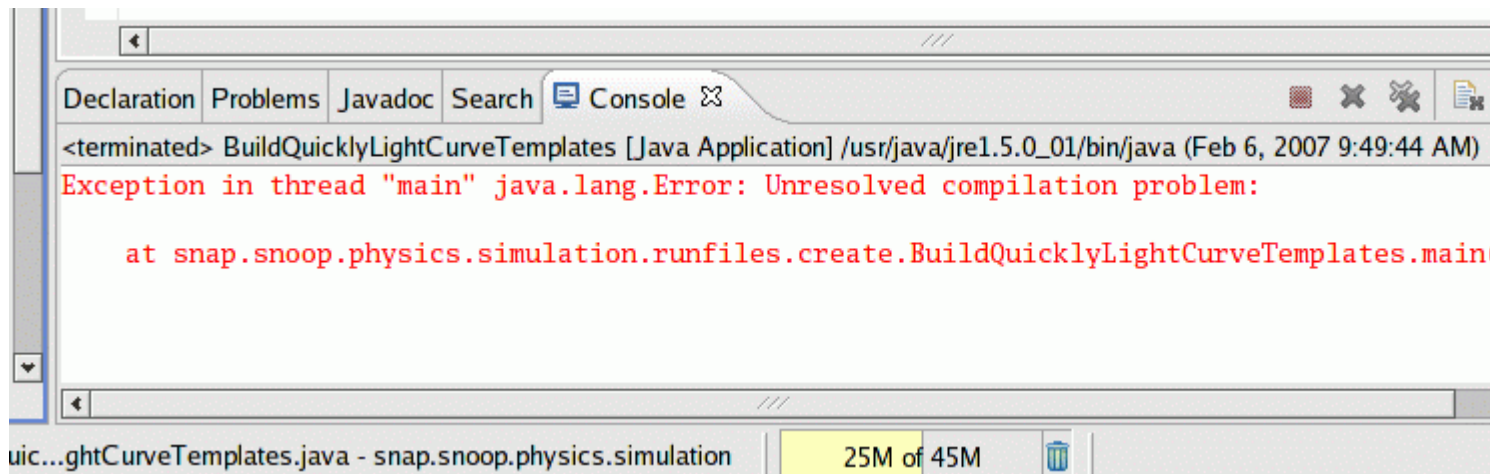
Running a simple piece of code

I will try to run one tiny little bit of code: a portion of the **snap.snoop.physics.simulation** project which builds light curve templates for supernovae. It's called **BuildQuicklyLightCurveTemplates.java**.

First, I go to the *Package Explorer* pane and, starting with **snap.snoop.physics.simulation**, click down through the choices to find this routine.



Whoops! The red "X" indicates that something is wrong with this routine. So, first I'll try to update the code ... nope, that didn't help. When I look in the *Console* pane, I see an error message in red:



By double-left-clicking on the name of this routine in the *Package Explorer* pane, I can open a new source code viewer/editor pane which shows the code for this routine:

```

user.properties  *BuildQuicklyLightCurveTemplates.java x
1 package snap.snoop.physics.simulation.runfiles.create;
2
3 import static snap.snoop.physics.simulation.parameters.SimulationParameterFactory
21
22
23 /**
24  * This runfile produces SN Templates for redshifts using SP and a set step
25  * size of .5 using the perfectly known SNAP transmission curves and using the
26  * PeterNugent2004Ia supernova model.
27  *
28  * The DatabaseTemplateLibrary is used to store the results. This class
29  * provides a simple way to retrieve the results. DatabaseTemplateLibrary
30  * used in conjunction with InterpolatedTemplateLibrary can be used when
31  * interpolating between redshifts is appropriate.
32  */
33 public class BuildQuicklyLightCurveTemplates implements RunFile
34 {
35     /**
36     * Get SNAP Global Logger.
37     */
38     static Logger log = LogUtil.getLogger();
39
40     /*****
41     * Run Control Parameters *
42     *****/
43
44     String yourName = "kushner";
45
46     String datasetName = NodalLightCurveTemplates;
47

```

Declaration Problems Javadoc Search Console

<terminated> BuildQuicklyLightCurveTemplates [Java Application] /usr/java/jre1.5.0_01/bin/java (Feb 6, 2007 9:49:44 A
Exception in thread "main" java.lang.Error: Unresolved compilation problem:

If I choose the *Problems* tab in the bottom pane of my Eclipse window, I see a short list of errors:

```

45
46 String datasetName = NodalLightCurveTemplates;
47

```

Declaration Problems Javadoc Search Console

2 errors, 2 warnings, 0 infos (Filter matched 4 of 5134 items)

Description	Resource
✘ NodalTemplateLibrary cannot be resolved to a type	BuildQuicklyLig
✘ The method run(SimulationParameters, double, double, double) is undefined for the type CreateN	BuildQuicklyLig
⚠ Javadoc: Missing comment for public declaration	BuildQuicklyLig
⚠ Javadoc: Missing comment for public declaration	BuildQuicklyLig

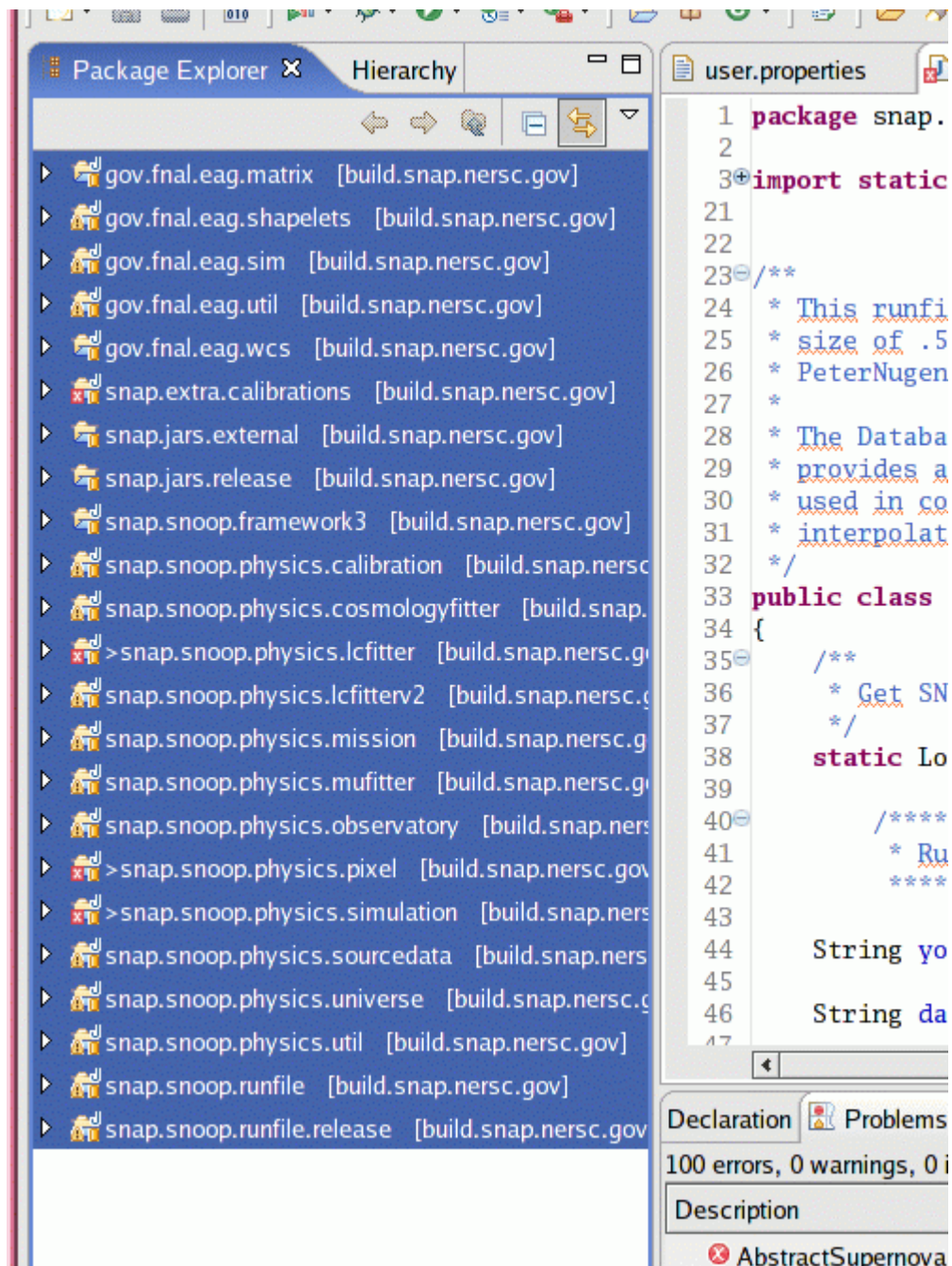
29M of 47M

When I scroll all the way to the right end of the first error message, I see that it occurs on line 73 of **BuildQuicklyLightCurveTemplates.java**. So, I move in the editor window to line 73, and find a little red "X" next to the lines with the error.

```
68
69     double zStart = sp.getPropertyAsDouble(LCTZStart);
70     double zEnd   = sp.getPropertyAsDouble(LCTZEnd);
71     double zStep  = .5;
72
73     NodalTemplateLibrary nLibrary =
74         CreateNodalTemplateLightCurves.run(sp, zStart, zEnd, zStep);
75
76     nLibrary.setIName(NodalLightCurveiName);
77     nLibrary.deepSave(cds);
78 //     cds.save(nLibrary);
```

Hmmm. We seem to be declaring a variable whose type, **NodalTemplateLibrary**, is not defined anywhere. There must be some other code, elsewhere, which does define this type. How can I track it down and make sure I include it?

I'll try updating ALL the code in the entire SNAPSim package. I go to the *Package Explorer* pane and highlight all the packages:



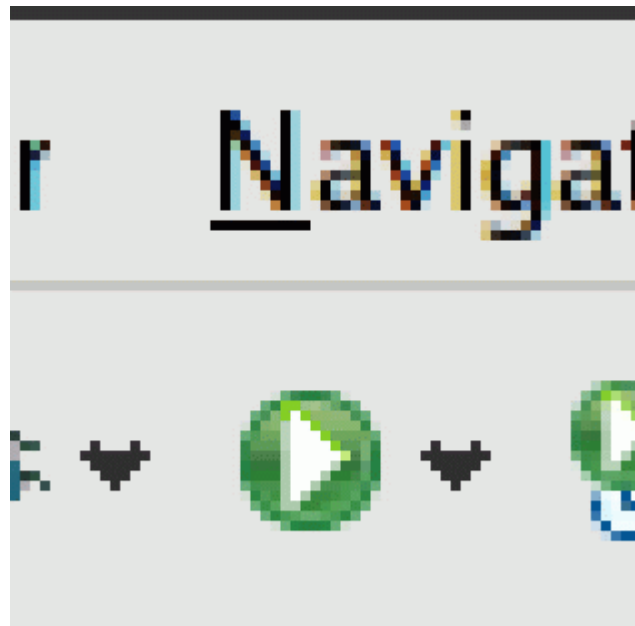
Now I right-click and choose **Replace With ...** and **Latest from HEAD**. A status window pops up and much code is transferred. When the transfer finishes, my source code file no longer has little red "X" markers next to the line with the **NodalTemplateLibrary** declaration:

```

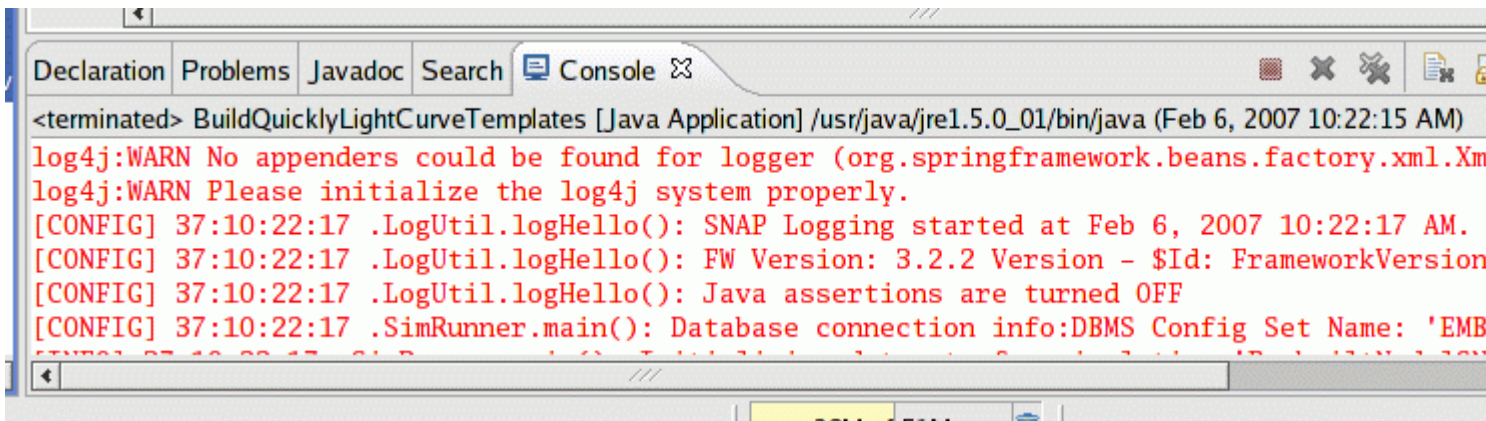
62
63     SimulationParameters
64         sp = InitializeSimulation.run(cds, SPiName, UPiName, SimName);
65
66     double zStart = sp.getPropertyAsDouble(LCTZStart);
67     double zEnd   = sp.getPropertyAsDouble(LCTZEnd);
68     double zStep  = .5;
69
70     NodalTemplateLibrary nLibrary =
71         CreateNodalTemplateLightCurves.run(sp, zStart, zEnd, zStep);
72
73     nLibrary.setIName(NodalLightCurveiName);
74     nLibrary.deepSave(cds);
75     // cds.save(nLibrary);
76 }
77
78 /*****
79  * Template Generated Code *
80  *****/

```

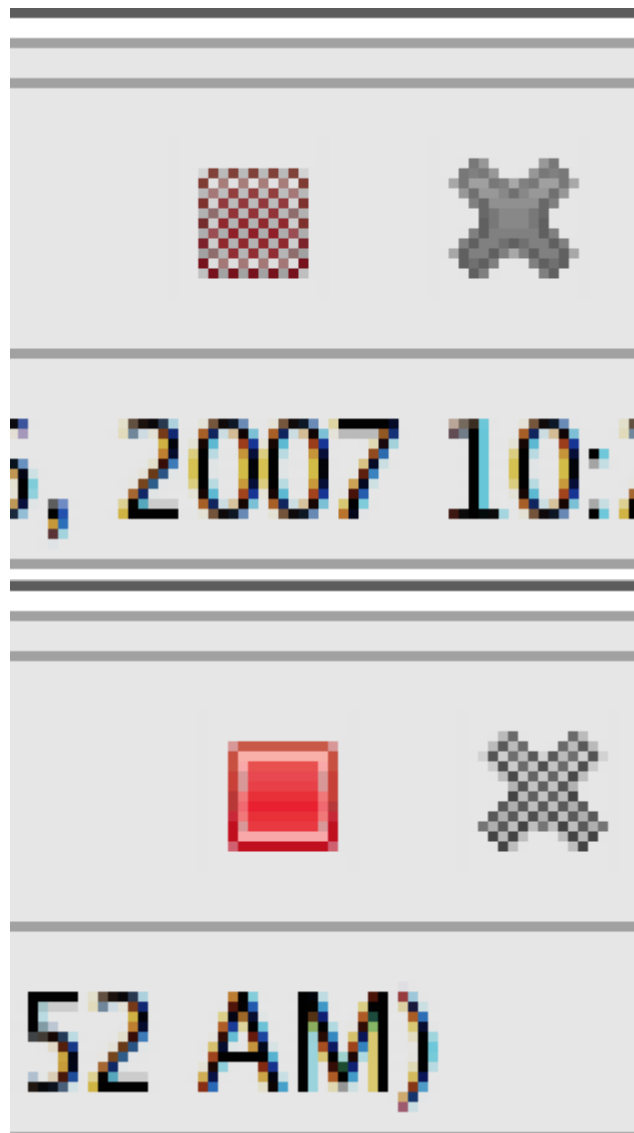
Hooray! Now I'll try running this routine again. I move my cursor to the *Run* icon in the toolbar



and left-click it. Several seconds pass, but stuff is happening -- I can see notification messages in the *Console* pane. Then the routine actually begins to run. I can follow its progress by looking at the *Console* window. Here's the start of the many, many messages it prints as this routine runs:

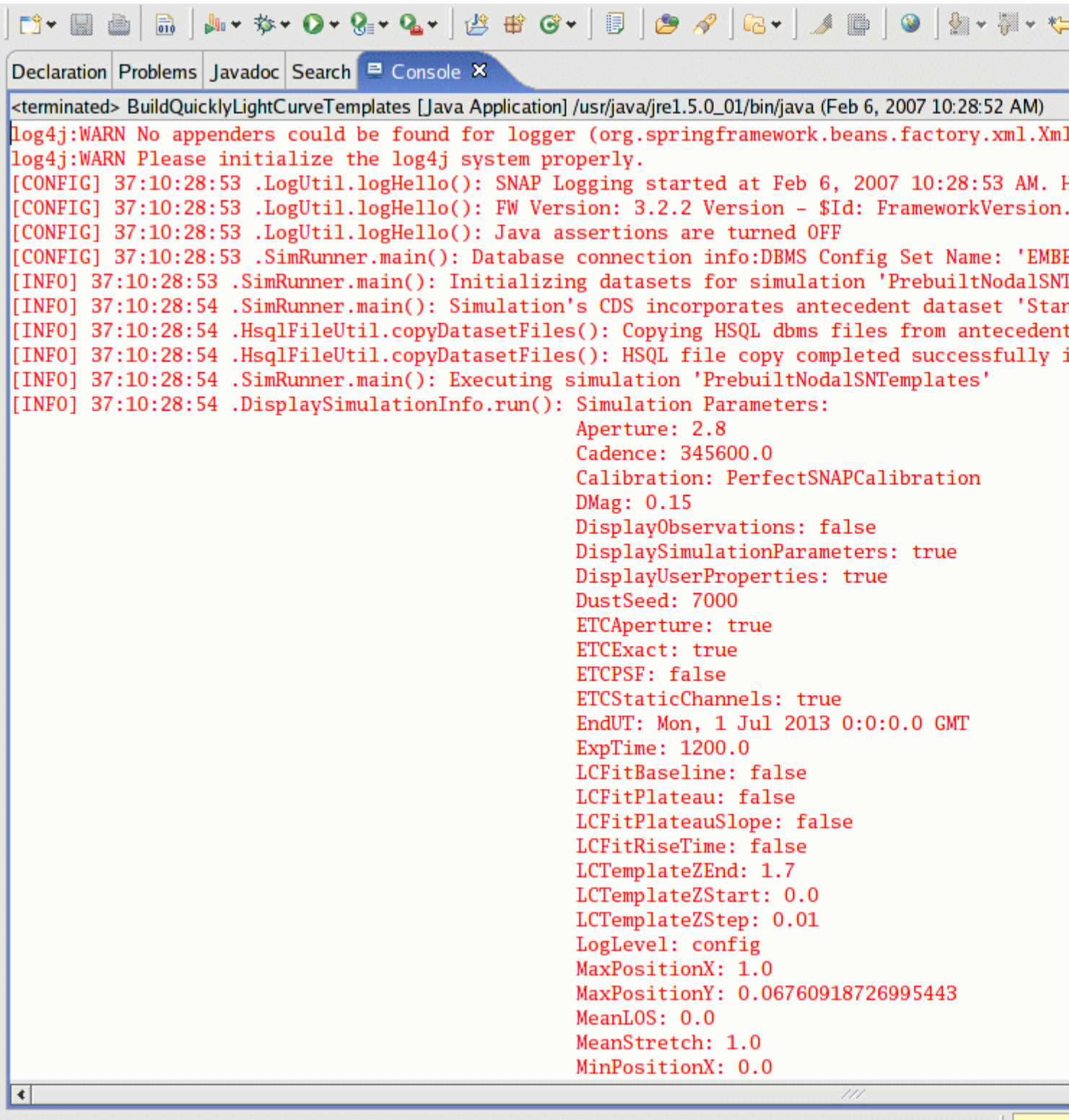


Look closely at the picture above. In the toolbar at the top of this pane, just above the date "Feb 6, 2007", there is a small square icon. The icon is normally a greyed-out red (left picture below), but when a program is running, it glows bright red (right picture below).



Let's look at the status messages in a bit more detail. I will enlarge the *Console* pane by clicking on the "Maximize" icon, at the far right end of the toolbar for this pane. Even though these messages are written in a red color, they don't (necessarily) indicate errors.

At first, I see a bit of bookkeeping information. Then, a long string of parameter values -- these might be useful if one is trying to debug some new routine.



The screenshot shows a Java IDE window titled "BuildQuicklyLightCurveTemplates [Java Application] /usr/java/jre1.5.0_01/bin/java (Feb 6, 2007 10:28:52 AM)". The console output is as follows:

```
<terminated> BuildQuicklyLightCurveTemplates [Java Application] /usr/java/jre1.5.0_01/bin/java (Feb 6, 2007 10:28:52 AM)
log4j:WARN No appenders could be found for logger (org.springframework.beans.factory.xml.Xml
log4j:WARN Please initialize the log4j system properly.
[CONFIG] 37:10:28:53 .LogUtil.logHello(): SNAP Logging started at Feb 6, 2007 10:28:53 AM. H
[CONFIG] 37:10:28:53 .LogUtil.logHello(): FW Version: 3.2.2 Version - $Id: FrameworkVersion.
[CONFIG] 37:10:28:53 .LogUtil.logHello(): Java assertions are turned OFF
[CONFIG] 37:10:28:53 .SimRunner.main(): Database connection info:DBMS Config Set Name: 'EMBE
[INFO] 37:10:28:53 .SimRunner.main(): Initializing datasets for simulation 'PrebuiltNodalSNT
[INFO] 37:10:28:54 .SimRunner.main(): Simulation's CDS incorporates antecedent dataset 'Star
[INFO] 37:10:28:54 .HsqlFileUtil.copyDatasetFiles(): Copying HSQL dbms files from antecedent
[INFO] 37:10:28:54 .HsqlFileUtil.copyDatasetFiles(): HSQL file copy completed successfully i
[INFO] 37:10:28:54 .SimRunner.main(): Executing simulation 'PrebuiltNodalSNTemplates'
[INFO] 37:10:28:54 .DisplaySimulationInfo.run(): Simulation Parameters:
    Aperture: 2.8
    Cadence: 345600.0
    Calibration: PerfectSNAPCalibration
    DMag: 0.15
    DisplayObservations: false
    DisplaySimulationParameters: true
    DisplayUserProperties: true
    DustSeed: 7000
    ETCAperture: true
    ETCEXact: true
    ETCPSF: false
    ETCStaticChannels: true
    EndUT: Mon, 1 Jul 2013 0:0:0.0 GMT
    ExpTime: 1200.0
    LCFitBaseline: false
    LCFitPlateau: false
    LCFitPlateauSlope: false
    LCFitRiseTime: false
    LCTemplateZEnd: 1.7
    LCTemplateZStart: 0.0
    LCTemplateZStep: 0.01
    LogLevel: config
    MaxPositionX: 1.0
    MaxPositionY: 0.06760918726995443
    MeanLOS: 0.0
    MeanStretch: 1.0
    MinPositionX: 0.0
```

Scrolling down through the messages, I reach a point at which the routine of interest really gets started:

```

[INFO] 37:10:28:54 .DisplaySimulationInfo.run(): User Properties Set: (BuildLCTemplates)
                                     User Properties Set: (--unqualified--)
                                     *end*
[INFO] 37:10:28:54 .GetSNAPStandardConfiguration.run(): Getting the Standard SNAP Configurati
[INFO] 37:10:28:55 .CreateNodalTemplateLightCurves.run(): Creating Supernova model to be use
[INFO] 37:10:29:01 .CreateNodalTemplateLightCurves.run(): Building Light Curves for channel
[INFO] 37:10:29:10 .CreateNodalTemplateLightCurves.run(): Remaining time 01:12 seconds
[INFO] 37:10:29:10 .CreateNodalTemplateLightCurves.run(): Building Light Curves for channel
[INFO] 37:10:29:18 .CreateNodalTemplateLightCurves.run(): Remaining time 59 seconds
[INFO] 37:10:29:18 .CreateNodalTemplateLightCurves.run(): Building Light Curves for channel
[INFO] 37:10:29:27 .CreateNodalTemplateLightCurves.run(): Remaining time 51 seconds
[INFO] 37:10:29:27 .CreateNodalTemplateLightCurves.run(): Building Light Curves for channel
[INFO] 37:10:29:34 .CreateNodalTemplateLightCurves.run(): Remaining time 40 seconds
[INFO] 37:10:29:34 .CreateNodalTemplateLightCurves.run(): Building Light Curves for channel
[INFO] 37:10:29:44 .CreateNodalTemplateLightCurves.run(): Remaining time 34 seconds
[INFO] 37:10:29:44 .CreateNodalTemplateLightCurves.run(): Building Light Curves for channel
[INFO] 37:10:29:55 .CreateNodalTemplateLightCurves.run(): Remaining time 26 seconds
[INFO] 37:10:29:55 .CreateNodalTemplateLightCurves.run(): Building Light Curves for channel
[INFO] 37:10:30:07 .CreateNodalTemplateLightCurves.run(): Remaining time 18 seconds
[INFO] 37:10:30:07 .CreateNodalTemplateLightCurves.run(): Building Light Curves for channel
[INFO] 37:10:30:17 .CreateNodalTemplateLightCurves.run(): Remaining time 9 seconds
[INFO] 37:10:30:17 .CreateNodalTemplateLightCurves.run(): Building Light Curves for channel
[INFO] 37:10:30:36 .CreateNodalTemplateLightCurves.run(): Remaining time 0 seconds
[INFO] 37:10:30:37 .SimRunner.main(): Closing simulation 'PrebuiltNodalSNTemplates'
[INFO] 37:10:30:38 .SimRunner.main(): Statistics:
                                     -----
                                     Number of nouns saved: 471
                                     Number of nouns imported : 0   (not including inlined
                                     Number of nouns read from dbms: 52
                                     Number reads from cache instead of dbms: 0
[INFO] 37:10:30:38 .SimRunner.main(): Simulation completed.
[CONFIG] 37:10:30:38 .LogUtil.logGoodbye(): Logging active for 00:01:44
[CONFIG] 37:10:30:38 .LogUtil.logGoodbye(): SNAP Logging stopped at Feb 6, 2007 10:30:38 AM.

```

The person who wrote this routine very kindly included special statements which print out the status and give an estimate of the time remaining. Finally, at the end of the routine, there are some statistics and a timestamp.

Finding datafiles written to disk

The routine I ran, **BuildQuicklyLightCurveTemplates**, presumably created some templates of light curves. Where are those light curves? Can I read them with some external program, to plot them or use them outside the SNAPSIm environment?

SNAPSIm writes data to disk in a directory which is specified in a file called *snoop.properties*. That file should have been placed into some specific directory when the SNAPSIm framework was installed on your computer. In my case, this file lives in a directory called `~/snoop`.


```
[richmond@spiff ~]$ cd
[richmond@spiff ~]$ cd .snoop
[richmond@spiff .snoop]$ ls
total 44
 8 ./      8 ../     12 snoop.properties    8 spring.xml    8 user.properties
[richmond@spiff .snoop]$ █
```

The *snoop.properties* contains simple ASCII text setting a number of global parameters for the SNAPSIm framework. A short distance into the file is a set of "Common application parameters".

```
#-----
# Common application parameters
#-----
help=false
siteId=29
logDirectory=/home/richmond/simdata/log
exportDirectory=/home/richmond/simdata
statisticsDirectory=/home/richmond/simdata/statistics
#
```

The *exportDirectory* line provides the location of the directory into which SNAPSIm will write (and read) datafiles. In my case, it is `/home/richmond/simdata`. Let me list the files in this directory.

```
richmond@spiff:~
[richmond@spiff ~]$ ls -l /home/richmond/simdata/
total 768
  8 drwxrwxr-x   4 richmond richmond   4096 Feb  6 10:30 ./
  8 drwx----- 86 richmond richmond   4096 Feb  6 09:26 ../
  8 drwxrwxr-x   2 richmond richmond   4096 Feb  6 10:30 log/
 96 -rw-rw-r--   1 richmond richmond  87743 Feb  6 10:23 PrebuiltNodalS
184 -r--r--r--   1 richmond richmond 1048576 Feb  6 10:23 PrebuiltNodalS
 12 -rw-rw-r--   1 richmond richmond   7652 Feb  6 10:23 PrebuiltNodalS
  8 -r--r--r--   1 richmond richmond    389 Feb  6 10:23 PrebuiltNodalS
  8 -r--r--r--   1 richmond richmond    783 Feb  6 10:23 PrebuiltNodalS
 96 -rw-rw-r--   1 richmond richmond  88178 Feb  6 10:30 PrebuiltNodalS
184 -r--r--r--   1 richmond richmond 1048576 Feb  6 10:30 PrebuiltNodalS
 12 -rw-rw-r--   1 richmond richmond   7652 Feb  6 10:30 PrebuiltNodalS
  8 -r--r--r--   1 richmond richmond    389 Feb  6 10:30 PrebuiltNodalS
  8 -r--r--r--   1 richmond richmond    783 Feb  6 10:30 PrebuiltNodalS
 16 -rw-rw-r--   1 richmond richmond   8890 Jan  5 15:32 StandardSimula
 20 -r--r--r--   1 richmond richmond 1048576 Jan  5 15:32 StandardSimula
  8 -rw-rw-r--   1 richmond richmond   2513 Jan  5 15:32 StandardSimula
  8 -r--r--r--   1 richmond richmond    389 Jan  5 15:32 StandardSimula
  8 -r--r--r--   1 richmond richmond    781 Jan  5 15:32 StandardSimula
 16 -rw-rw-r--   1 richmond richmond   8888 Jan  9 16:21 StandardSimula
 20 -r--r--r--   1 richmond richmond 1048576 Jan  9 16:21 StandardSimula
  8 -rw-rw-r--   1 richmond richmond   2512 Jan  9 16:21 StandardSimula
  8 -r--r--r--   1 richmond richmond    389 Jan  9 16:21 StandardSimula
  8 -r--r--r--   1 richmond richmond    781 Jan  9 16:21 StandardSimula
  8 drwxrwxr-x   2 richmond richmond   4096 Jan  5 15:31 statistics/
[richmond@spiff ~]$ █
```

Note the two sets of files with names that start `StandardSimulation-`. There are two sets because I ran the **BuildQuicklyLightCurveTemplates** twice, once at 15:32, and a second time at 16:21. The two sets are distinguished by an ID number, which is called the **datasetID** value. Each time one runs the framework, a unique **datasetID** will be generated and attached to the results.

A single dataset consists of 5 files, as you can see in the figure above. The information I want is stored in the largest of the files, the one with a suffix `.data`. Unfortunately, the file is not a simple ASCII text file; instead, the information within it is compressed. So, how can I examine the numbers?

Using the SNAPSIm data browser to examine datasets

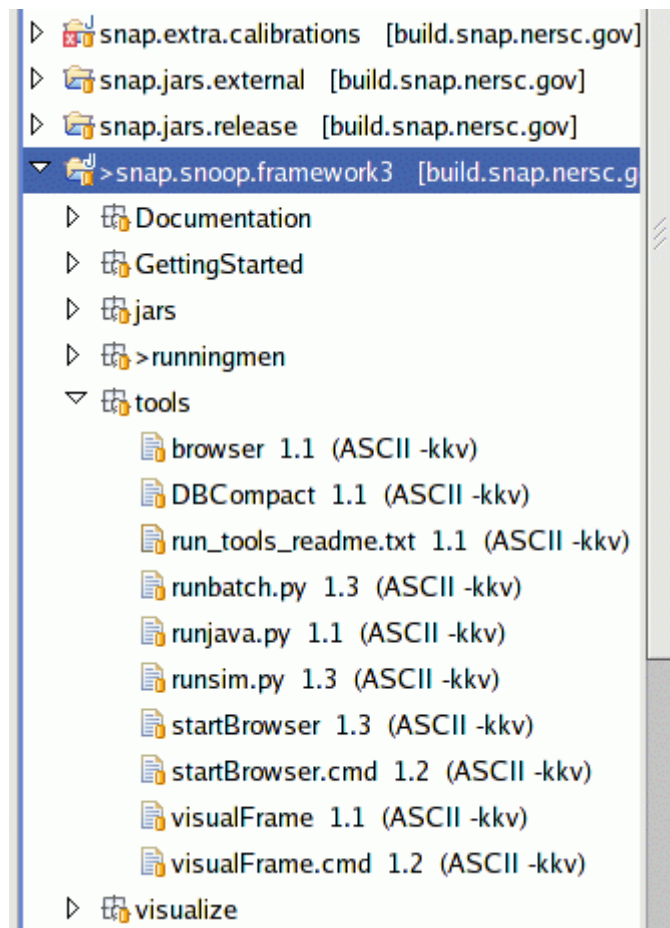
One way to examine datasets is to use the **data browser** built into SNAPSIm. This facility permits one to use any web browser (such as Firefox or Safari) to sift through the contents of a dataset, and save them in plain ASCII or HTML format.

Using the browser is a two-step procedure:

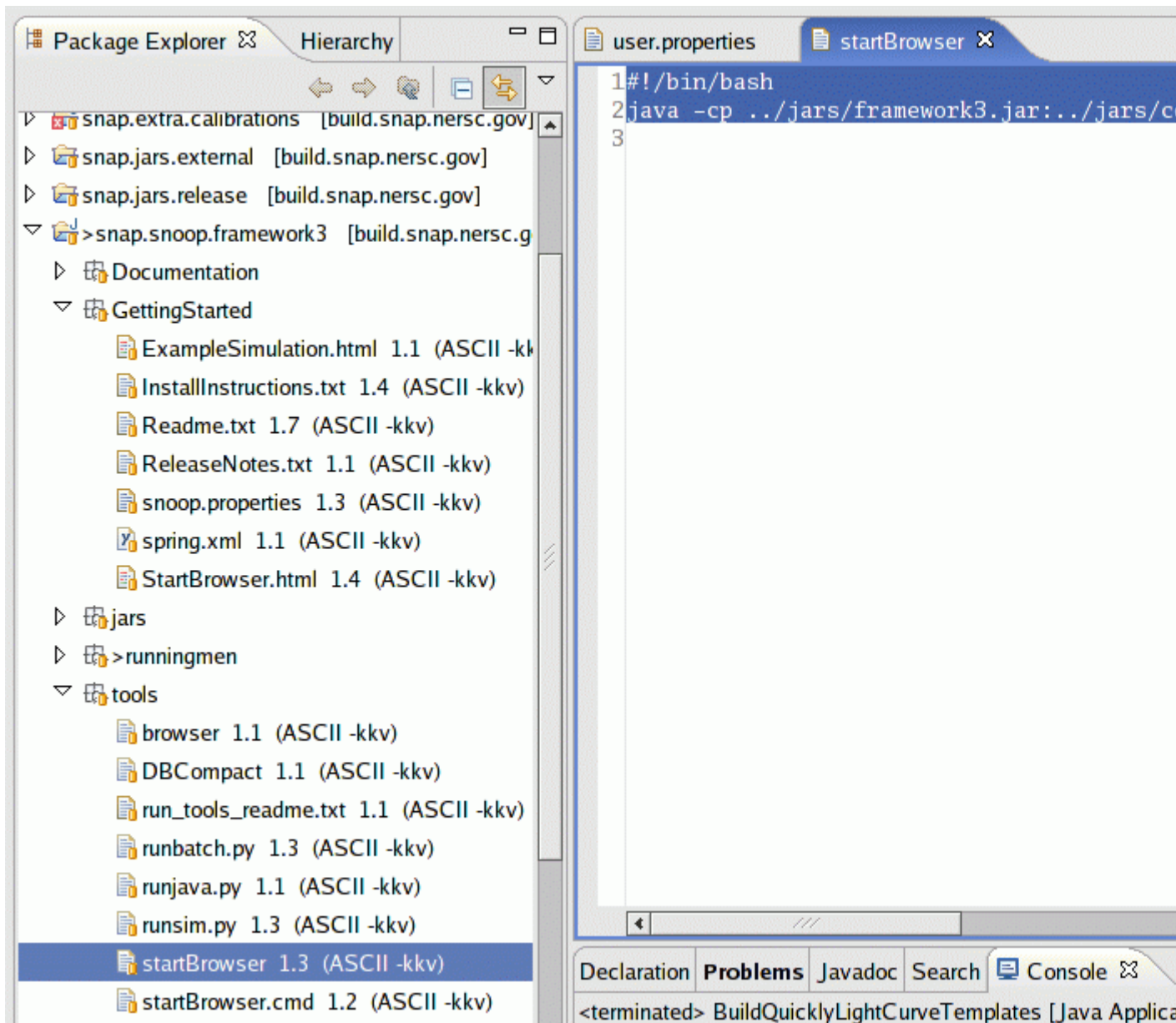
1. one must first start a special Server process running on one's computer
2. and then point a web browser to a very special address

Step One: start data server

Here's how I was able to use the browser on my Linux box. First, I opened the **snap.snoop.framework3** package in the *Package Navigator* pane and worked my way down to the "tools" section.



I then clicked on the *startBrowser* item to see the source code in the editor pane.



This item turns out to be a simple, two-line shell script. I reproduce it in full below -- note that the second line is very, very long.

```
#!/bin/bash
java -cp
../jars/framework3.jar:../jars/colt.jar:../jars/commons-cli-
1.0.jar:../jars/commons-lang-2.0.jar:../jars/commons-
logging.jar:../jars/hsqldb.jar:../jars/jasper-
compiler.jar:../jars/jasper-
runtime.jar:../jars/javax.jar:../jars/org.mortbay.jetty.jar:../j
ars/pg74.1jdbc.jar:../jars/spring.jar:../jars/javax.servlet.jar
snap.snoop.framework.web.browserapp.BrowserServer -war
../war/browser.war
```

I copied this text and pasted it into a text file on my computer, which I called "browser.sh".

- [browser.sh text file](#)

I copied this text file into one of the directories which contains the SNAPSim source code for the `snap.snoop.framework3` package. Specifically, the subdirectory called `runningmen`.

```
[richmond@spiff ~]$ cd
[richmond@spiff ~]$ cd /home/richmond/eclipse/workspace/snap.snoop.fr
[richmond@spiff snap.snoop.framework3]$ cd runningmen/
[richmond@spiff runningmen]$ pwd
/home/richmond/eclipse/workspace/snap.snoop.framework3/runningmen
[richmond@spiff runningmen]$ ls
total 56
8 ./      8 browser.sh  8 Dataset Browser.launch  8 Delete Dangling File
8 ../    8 CVS/       8 DBCompactor.launch
```

Now, what this shell script will do is to start running Java on a set of source code files. The Java program that results will read information saved by SNAPSim in its workspace directories, and make that information available to any other process which requests it. We'll call this the **server process**.

I run the shell script ...

```
[richmond@spiff runningmen]$ pwd
/home/richmond/eclipse/workspace/snap.snoop.framework3/runningmen
[richmond@spiff runningmen]$ ls
total 56
8 ./      8 browser.sh  8 Dataset Browser.launch  8 Delete Dangling File
8 ../    8 CVS/       8 DBCompactor.launch
[richmond@spiff runningmen]$ sh browser.sh
Starting BrowserServer...
[INFO] 37:14:18:55 .BrowserServer.run(): Using snap.war file: '../war
[INFO] 37:14:18:56 .ControllerServlet.init(): Invoking initializer...
[CONFIG] 37:14:18:57 .LogUtil.logHello(): SNAP Logging started at Feb
[CONFIG] 37:14:18:57 .LogUtil.logHello(): FW Version: 3.2.2 Version -
.v 1.4 2006/12/11 22:44:25 fwdev Exp $
[CONFIG] 37:14:18:57 .LogUtil.logHello(): Java assertions are turned
[INFO] 37:14:18:57 .ControllerServlet.init(): Initialization complete
```

If I open a new shell window on my machine and use the `ps` program to look at all the processes running on my computer, I'll see a new process; in fact, two processes, one the shell script, the other a new Java process (they are the final two processes shown in the list below)

```
12897 ?      S      0:00 /usr/sbin/httpd
12912 pts/10   S+     0:00 /home/richmond/eclipse/eclipse
12913 pts/10   Sl+    0:31 /usr/java/latest/bin/java -Xms40m -Xmx256m
13406 pts/4     S+     0:00 sh browser.sh
13407 pts/4     Sl+    0:01 java -cp ../jars/framework3.jar:../jars/col
```

Step Two: point a browser to the server's address

This step is easy. Once the server is running, we just point a web browser to a special URL. You can find this URL described in the **Readme.txt** item in the **snap.snoop.framework3** "Getting Started" package.

The server process listens on port 8080 of your computer. So, if we point to the URL, we should connect to it. You can try by clicking on the link below.

<http://localhost:8080/snap/servlet/SnapController?Action=ListAllDatasets>

On my computer, running Firefox under Linux, my browser window looks like this:

Once we have the server and browser talking to each other, things are easy. Let me look at the more recent of my two datasets of light curve templates. When I click on the link circled below

my browser opens up the dataset, showing all the variables which were saved:

Now, it turns out that when I ran this code to create Light Curve Templates, I didn't ask for the curves to be saved! None of the variables listed in the browser window contain the information I want. Rats.

Modifying and running Java code

So, let me try to modify the code so that these light curves ARE saved to disk. Can I do it?

I'll open the **BuildQuicklyLightCurveTemplates.java** item by double-clicking it in the *Package Explorer* pane. That will put a copy of the Java source code file into the editor pane.

Look carefully at line 75, which actually saves the data to disk via a call to the method `cds.save`. The line has been commented out with two slash characters, so that the light curves are never saved.

```
74 |  
75 | / /  
76 | :
```

I will delete these two slash characters in the editor pane, then save my changes using the *File -> Save* menu item at the top left of the Eclipse window. After I've performed these steps, my window looks like this:

Okay, I'll try running the code by moving my cursor to the editor pane, right-clicking to bring up a big menu, and choosing the *Run As -> Java Application* choice on the menu. The program runs, showing its progress via red text in the Console pane. I can maximize the Console pane to watch:

Aha! The console messages indicate that this time, the program saved 527 nouns to disk when it finished. If you go back and look at the messages in the console from [the first time I ran the code](#), you'll see that only 471 nouns were saved to disk. Good -- I was hoping that more information would be saved this time.

So, I'll point my browser to the latest dataset. Here's the list of all datasets, with the one I just generated circled in gold:

So, clicking on that link brings up the first of six pages of variables saved in the dataset:

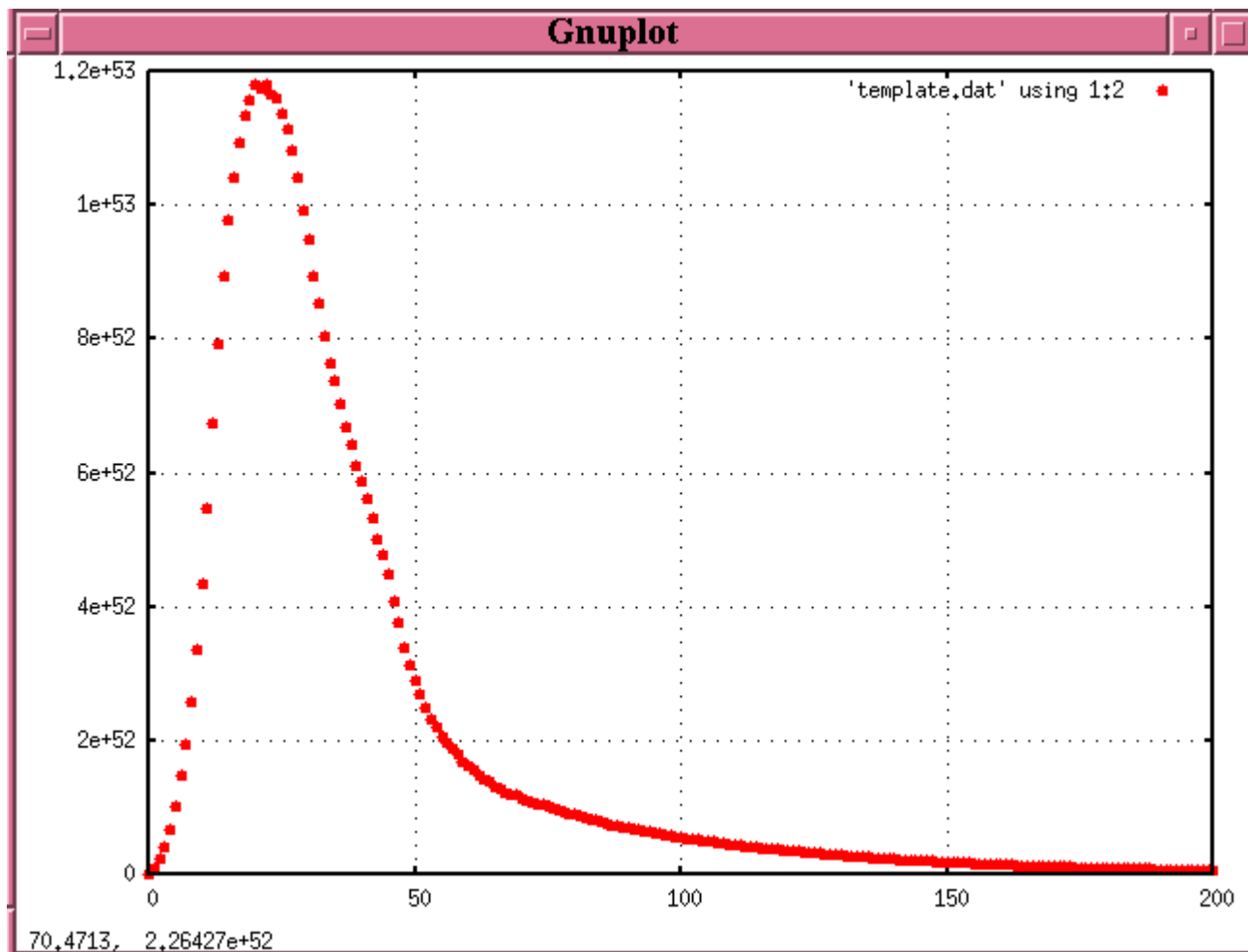
It's not completely clear which variable corresponds to which light curve, but one can figure some things out from the name (I could figure more out by reading the source code, I'm sure). Let me click on the very first item, *TabulatedSourceModel8TruthCalibration1.7* of Noun Class "Template".

Yes, this looks like a light curve. I see that this "Template" consists of one group of values labelled **flux**, and another called **date**. I can use my browser to save this page as plain ASCII text. With a little manipulation, I

can re-arrange the ASCII text into a simple form:

- [re-arranged "template.dat" file](#)

And I can then use my favorite tools to analyze the data, which appears to be the light curve of a supernova (in ergs per second) as a function of time (in days).



For more information

- [SNAPSim TWiki pages](#)
- [How to obtain CVS access for the SNAPsim framework](#)
- [Notes from the Nov 2006 SNAP Calibration team meeting](#)