Java Batch Job Framework

Basics Tutorial Guide

Author: Adym Lincoln, Java Batch Job Framework

Preamble

This tutorial assumes that you've read through the User Guide and more importantly, the Chapter on Essential Concepts. In this tutorial we will focus on the following concepts:

- ✔ Logging
- ✔ Email
- ✔ Task List

Glossary

Name	Description/Comments
JBJF	A document acronym for Java Batch Job Framework.
XML	Industry standard for Extensible Markup Language. A simple language for adding structure to data and documents.
XML Definition	A coding paradigm that combines Java's programming language with XML configuration files.
JBJF Batch Definition File	A specialized XML file that contains data and elements specific to a JBJF batch job.

References

- Title: jbjf-user-guide.pdf
  Location: http://jbjf.sourceforge.net/pdfs/jbjf-user-guide.pdf
  Author: Adym S. Lincoln
  Referenced Revision: 1.0.0

Basics Tutorials - Development

In this tutorial we'll focus on the basics of JBJF. This includes logging, email and especially the task list.

Prerequisites

The tutorial focuses on the essential services that JBJF supplies. Directories, Email and Logging are core services of JBJF. We will touch on all these core services in this first tutorial.

For this tutorial, I'll be using the following toolset:

- Some working Email recipients
- A working SMTP email host
- Windows 2000/XP/Vista
- Eclipse 3.2.1(+), Callisto or better
- JDK 1.6.0-12(+), Sun Microsystems mixed mode
- A bundled JBJF runtime package or the JBJF source code from the SourceForge JBJF File Release repository. This should be setup in the same Eclipse workspace where you'll be doing the tutorials.

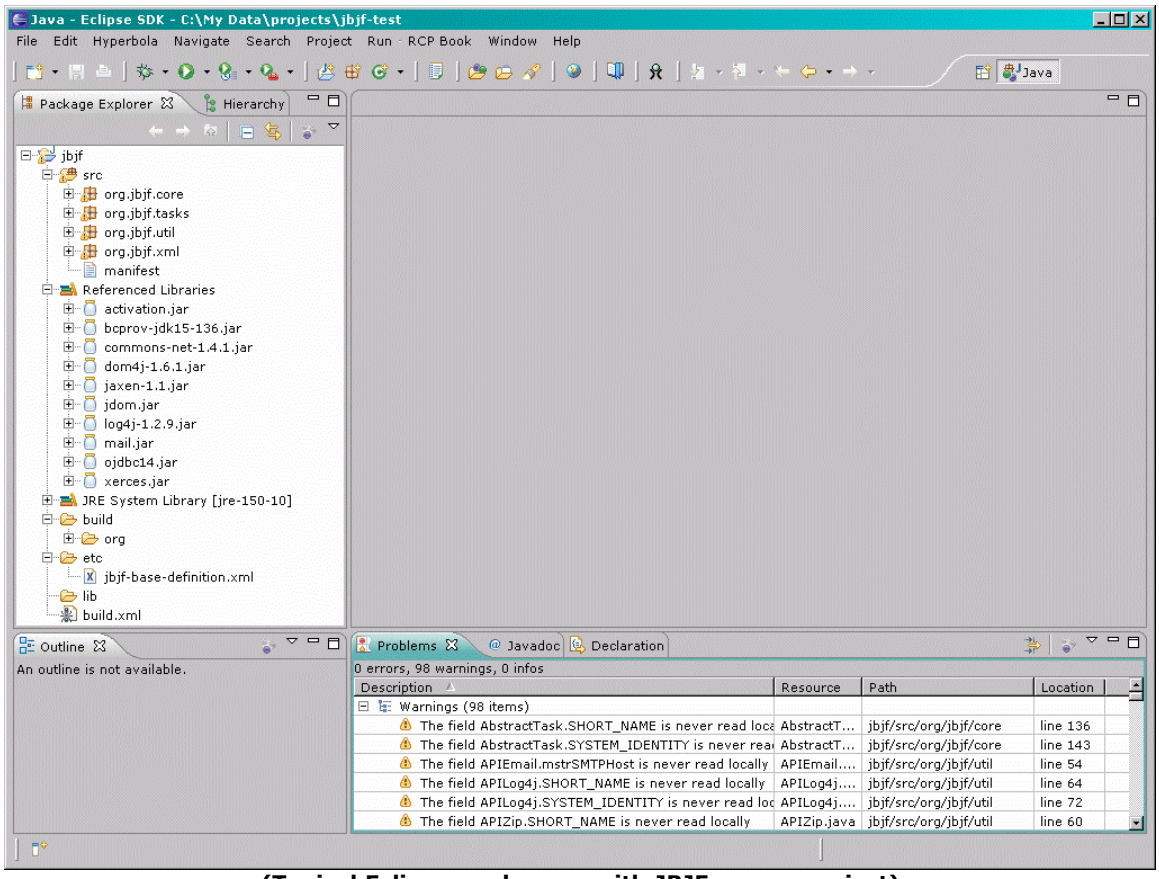
While the coding, builds and runtime will be done on a Windows desktop, any subtle differences to Unix or Linux can be easily adjusted.

Project Setup

Checkout the JBJF project or download a library/source package from the SourceForge repository. Unzip the library/source package into your desired folder location. For the purposes of this tutorial I'll refer to this location as ${jbjf-dir}. I'll be using a source code package for these tutorials, but you are free to use a binary package and just adjust your Eclipse environment accordingly.

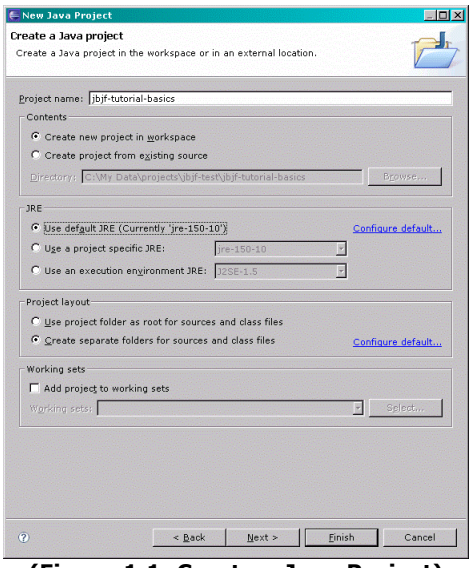
JBJF Downloads Page - http://sourceforge.net/projects/jbjf/

Open up your Eclipse and point it to the ${jbjf-dir}. For my examples, I'm using one Eclipse project to handle the JBJF source code and a separate project for the Tutorials. We'll actually work through the build process and deploy the JBJF to the tutorial project as a binary jarfile. So, while we're learning about how to code a JBJF batch job, we'll also be learning how to build and deploy the JBJF library.

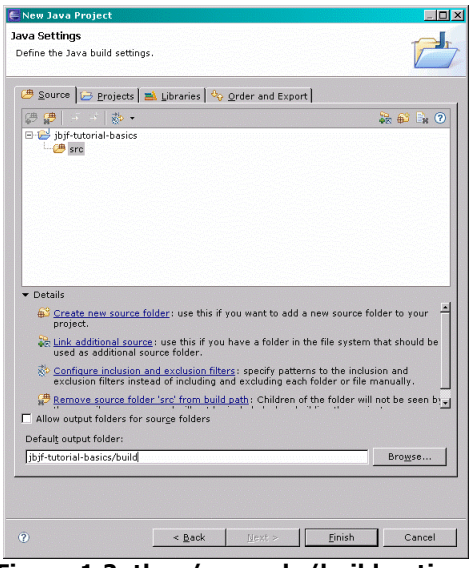


(Typical Eclipse workspace with JBJF source project)

Next, we need to create a tutorial project that we can build our batch job and tasks in. For this tutorial, I'll name my tutorial project jbjf-tutorial-basics. We'll refer to this as the ${tutorial-project-dir}. When we create the new project, set a source folder called src and set the Default output folder to jbjf-tutorial-basics/build...from the Source tab click the link Create new source folder. Then type in src and click the Finish button. By default, Eclipse uses the jbjf-tutorial-basics/bin for output, but we'll be using the jbjf-tutorial-basics/bin for console script files (*.bat, *.sh). See figures 1.1 and 1.2.

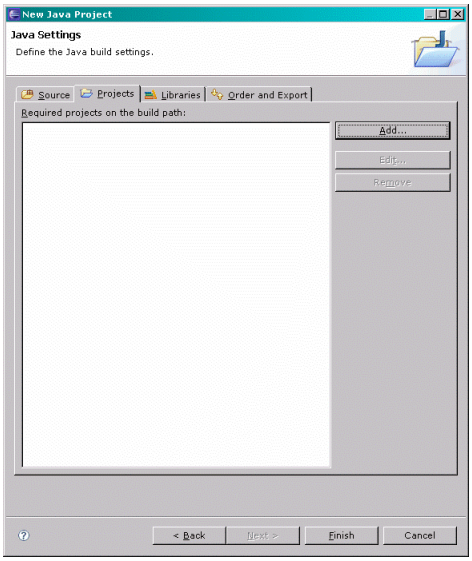


(Figure 1.1, Create a Java Project)

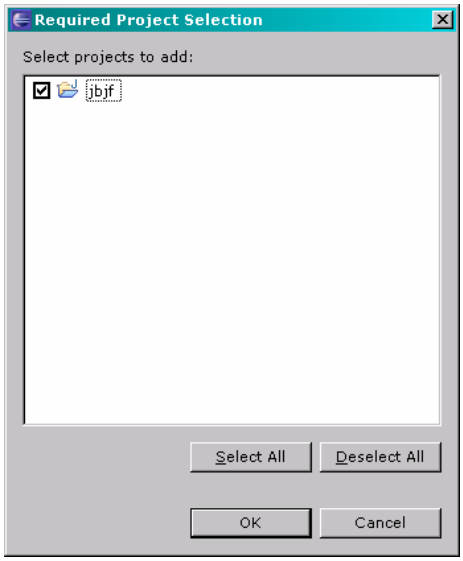


(Figure 1.2, the ./src and ./build options)

If you downloaded the source package for the JBJF then read through this section. If you are using a binary JBJF jarfile, skip ahead to the next step. For a source package, start by clicking on the Projects tab and then click the Add button. Locate the JBJF project from the list and click the checkbox. Then click the OK button to complete the addition. See figures 2.1 and 2.2.



(Figure 2.1, The Projects Tab)

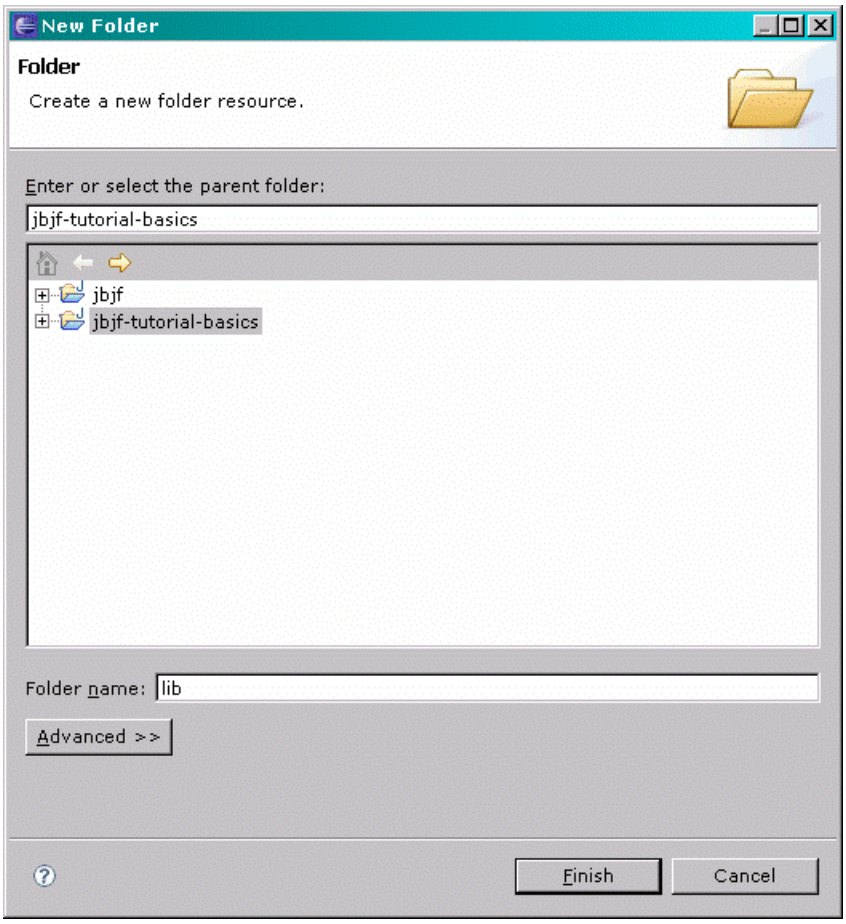


(Figure 2.2, adding the JBJF project)

For a binary jarfile JBJF package, the simplest way to add the JBJF framework after the project is created. For now, continue to create the project and we'll add the JBJF framework later. Once you've added the jbjf project, click the Finish button to create the project.

Once the project has been created, we also need to add the following sub-folders within the jbjf-tutorial-basics project. For each sub-folder to be added, right click on

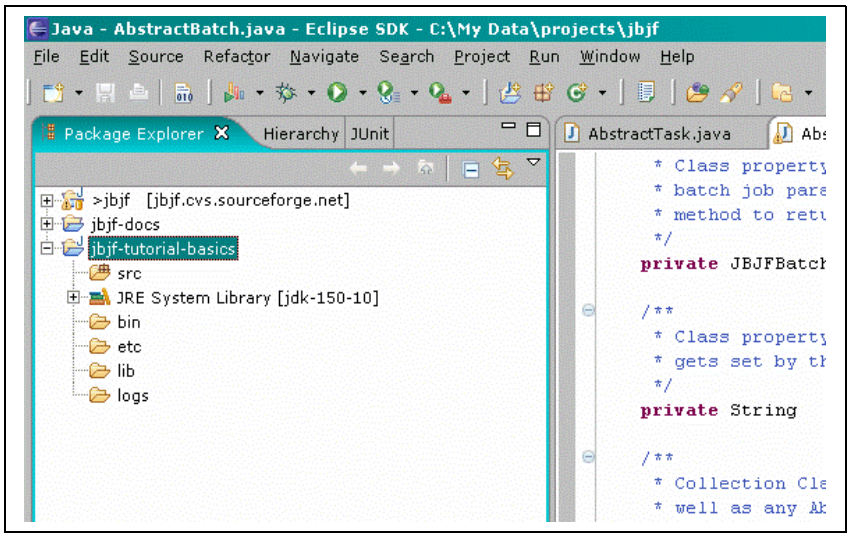
the jbjf-tutorial-basics project and select New -> Folder. Then type in the name and click the Finish button.



Create the following sub-folders:

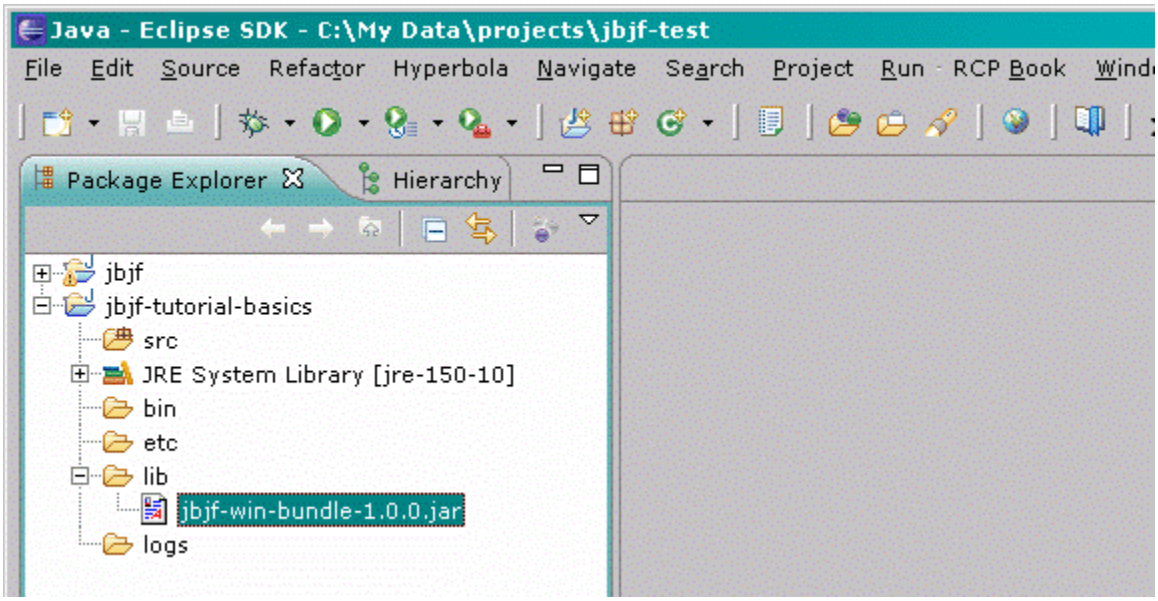
- lib
- etc
- logs

This completes the project setup. Your new tutorial project should be similar to the following:

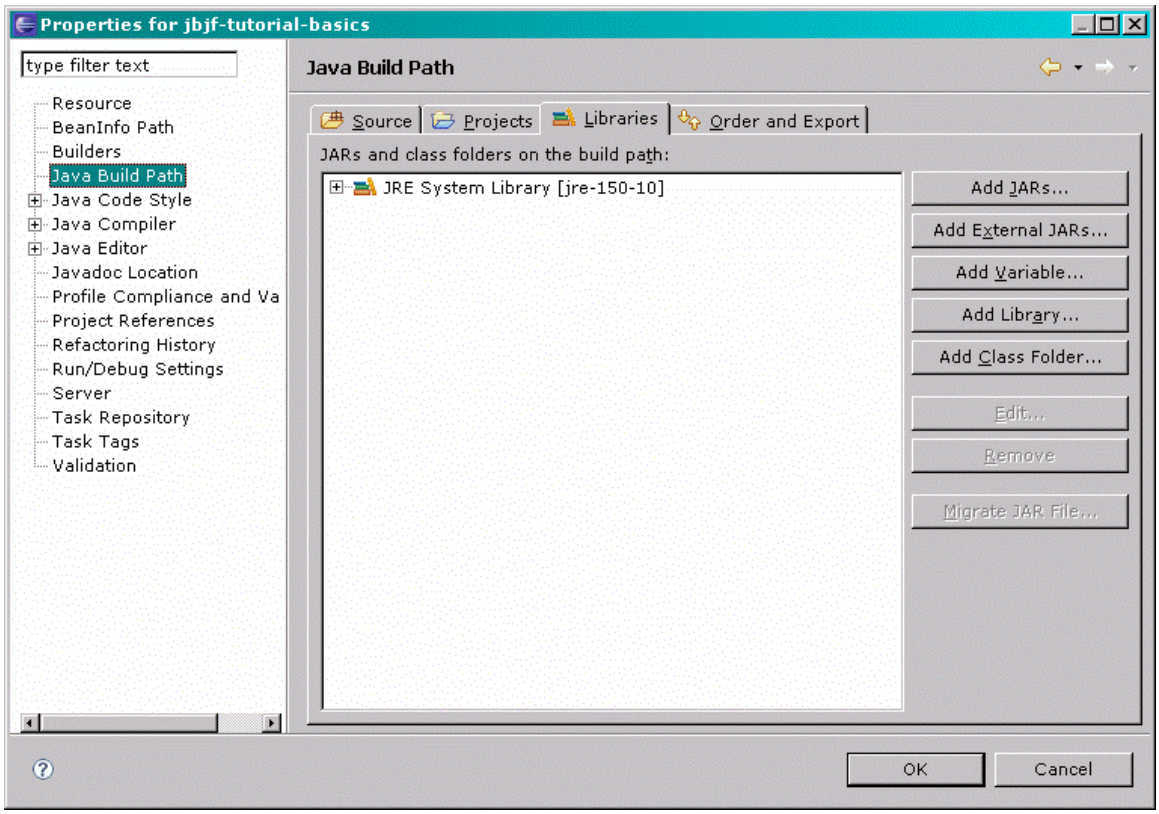


Adding the JBJF Jar file

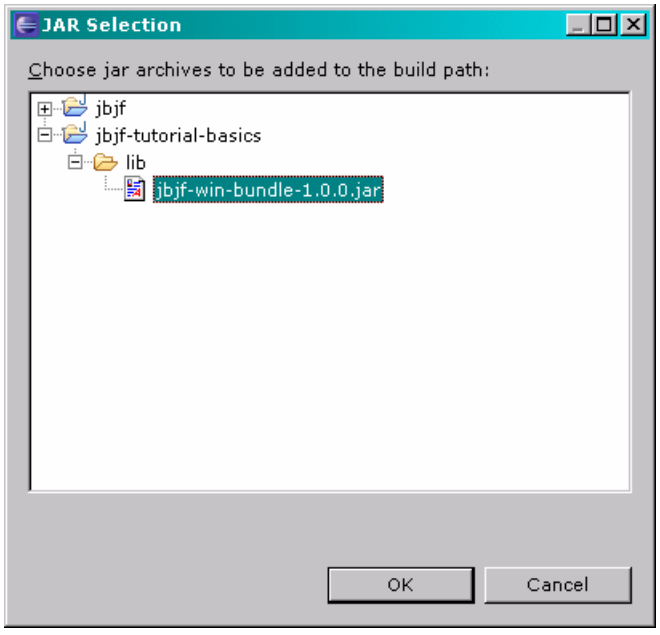
As mentioned before, the binary JBJF package can be added to the project as a library. We've already created a sub-directory in the tutorial project called ./lib. So take your JBJF jar file download package and place it in the ./lib sub-directory in the tutorial project.



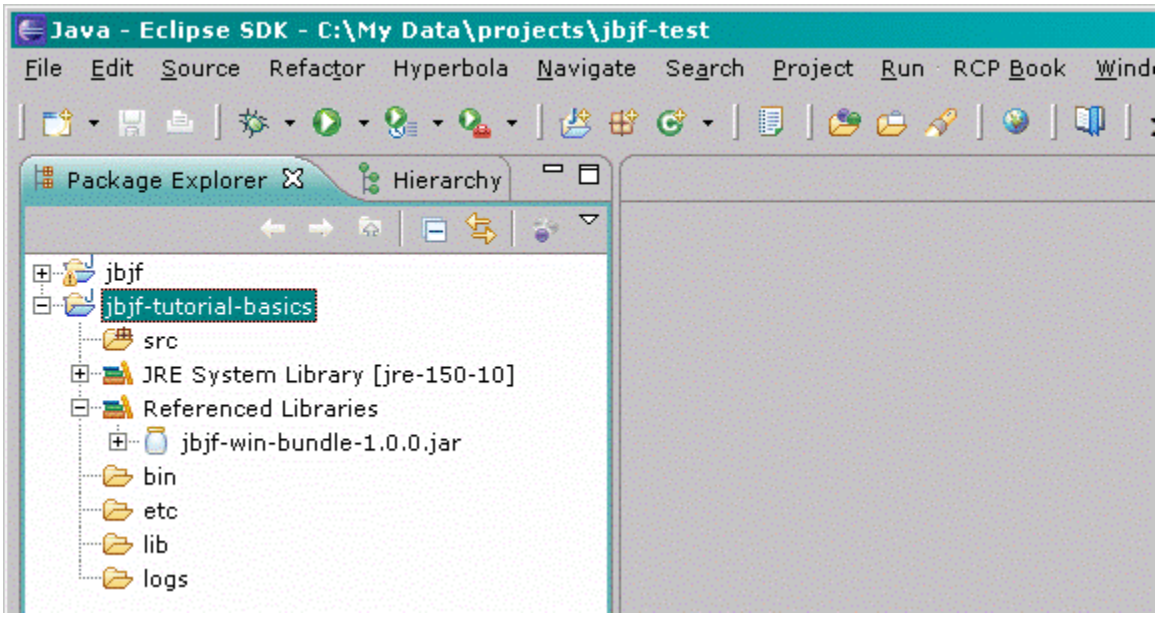
Now, right-click on the tutorial projects folder and select Properties at the bottom of the pop-up menu. This will bring up the Project Properties dialog. Click the Java Build Path option located on the side and then click the Libraries Tab.



Now click the Add JARS... button and locate the jbjf jar file located in the ./lib sub-folder of the tutorial project.



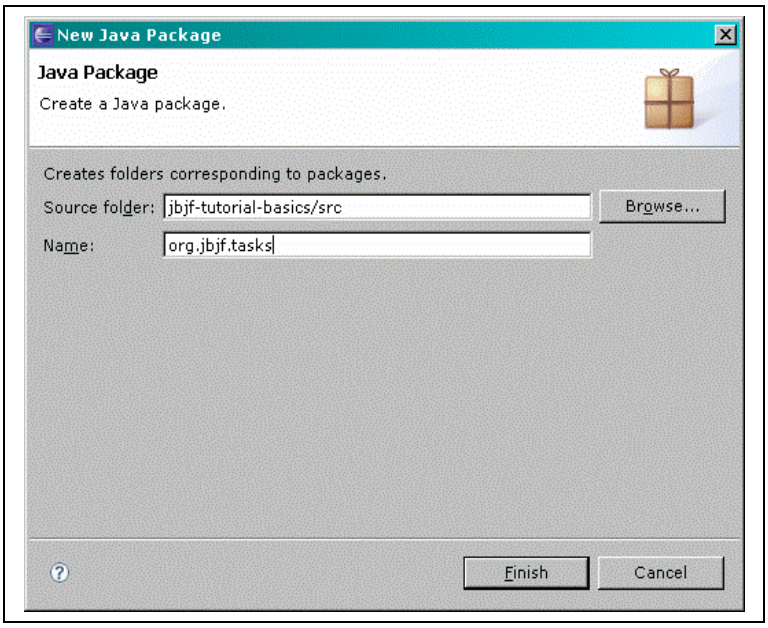
Click the OK button and the JBJF framework will be added to the tutorial project as a library.



Creating Classes

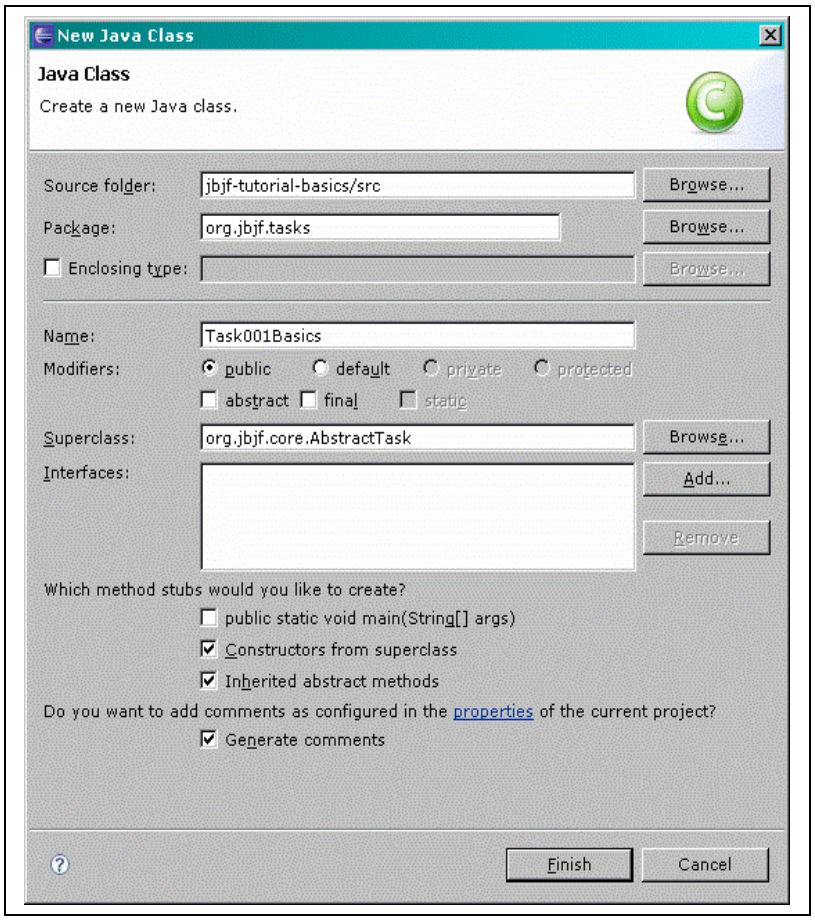
Before we begin creating classes, we first need to setup a couple of Java packages to hold our classes. Essentially, we'll be creating Tasks (sub-classes of the AbstractTask class) within the tutorial project. For this, we'll create a Java package called org.jbjf.tasks.

Within the jbjf-tutorial-basics project is a folder called src with a special source code icon. Right-click on the src folder and select New -> Package. Name the new package org.jbjf.tasks and click the Finish button to add the package.



Next, we need to add our first task class to the project. For this first tutorial, we'll be setting up a fairly useless batch job that iterates through a series of tasks. No real work gets accomplished and all output will be to the console and the logfile.

Right-click on the new package and select New -> Class from the pop-up menu. Name your class something associated with which tutorial and what task. For me, I named it Task001Basics. The Superclass: should be AbstractTask from org.jbjf.core. Make sure you click the checkboxes Constructors from superclass, Inherited abstract methods and Generate comments. One or more of these may already be selected.



Click the Finish button to create the new class. Depending on your Eclipse preferences, Eclipse should create the class, place it in the Editor and then flip to the Java Perspective. For this first tutorial, our tasks are going to write to the console...nothing fancy, but it will illustrate the basics of the JBJF. We also won't be creating an AbstractBatch; we'll use a built in DefaultBatch class to run our tasks. Again, we want the first tutorial to be quick, easy and fun.

Copy and paste the following code into your runTask() method of your new class:

```
getLog().debug( "Task [ Task001Basics ]...Starting..." );
getLog().debug( "Task [ Task001Basics ]...Complete..." );
```

The JBJF uses Apache's Log4j to handle the logging. As such, the Logger gets initialized in the AbstractBatch sub-class (parent class of this task), in our case the builtin DefaultBatch class within the JBJF. Thus, by the time the runTask() method is run, the Log4j logger is ready to use. We use the getLog() method to fetch that logger. The rest is standard log4j coding. The getLog().debug() means the log4j level needs to be set to DEBUG in order to see these logging entries. We'll set that level in a minute.

At this point your class should be similar to the following, focus on the constructor and the runTask() method:

```
public class Task001Basics extends AbstractTask {
    /**
     * Stores a fully qualified class name.  Used for debugging and
     * auditing.
     * @since 1.0.0
     */
    public static final String ID = Task001Basics.class.getName();

    /**
     * Stores the class name, primarily used for debugging and so
     * forth.  Used for debugging and auditing.
     * @since 1.0.0
     */
    private String SHORT_NAME = "Task001Basics()";

    /**
     * Stores a <code>SYSTEM IDENTITY HASHCODE</code>.  Used for
     * debugging and auditing.
     * @since 1.0.0
     */
    private String SYSTEM_IDENTITY = String.valueOf( System.identity-
HashCode( this ) );

    /**
     *
     */
    public Task001Basics() {
        // TODO Auto-generated constructor stub
    }

    /* (non-Javadoc)
     * @see org.jbjf.core.AbstractTask#runTask(java.util.HashMap)
     */
    @Override
    public void runTask( HashMap pjobParameters ) throws Exception {
        getLog().debug( "Task [ " + this.SHORT_NAME + " ]...Starting..." );
        getLog().debug( "Task [ " + this.SHORT_NAME + " ]...Complete..." );
    }
}
```

Don't be concerned about the reference to `this.SHORT_NAME`. This is a constant that gets created by my Eclipse Preferences for a new Class/Type. While the initial code

snippet used a hard-coded class name (`Task001Basics`). I've got a simple reference to a static constant for the class name.

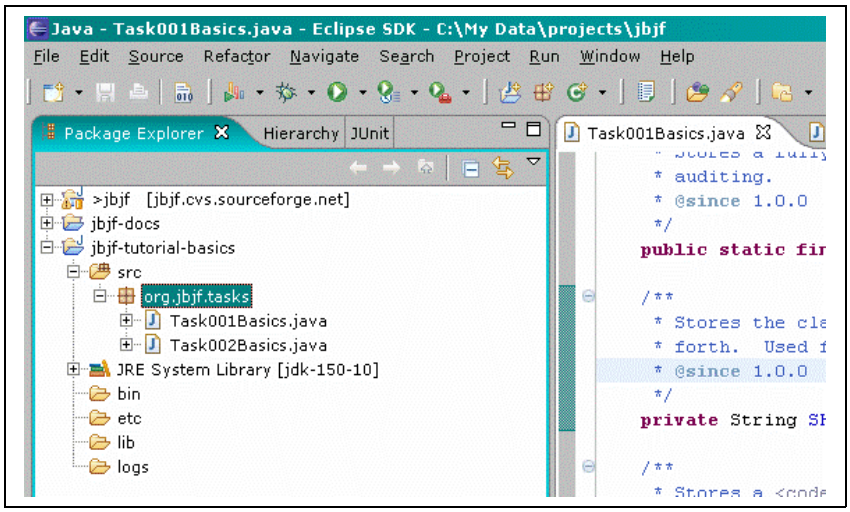
Eclipse Tip:
You can set various code preferences within Eclipse by going to Window -> Preferences -> Java -> Code Style -> Code Templates. Select and expand Code. In here are a number of different templates that can be used for code generation when certain Java elements get created. Thus, my static constant SHORT_NAME above is placed within the Class body template:

```
/**
 * Stores a fully qualified class name.  Used for debugging and
 * auditing.
 * @since 1.0.0
 */
public static final String ID = ${type_name}.class.getName();

/**
 * Stores the class name, primarily used for debugging and so
 * forth.  Used for debugging and auditing.
 * @since 1.0.0
 */
private String SHORT_NAME = "${type_name}()";

/**
 * Stores a <code>SYSTEM IDENTITY HASHCODE</code>.  Used for
 * debugging and auditing.
 * @since 1.0.0
 */
private String SYSTEM_IDENTITY = String.valueOf(System.identityHashCode(this));
```

Note the placement of variable such as ${type_name} in order to have this template work for any class.

Repeat the above steps for a second AbstractTask sub-class...again name the class something appropriate for a 2nd task in the first tutorial. Task002Basics or something.

When you're finished, you should have two classes in the org.jbjf.tasks package of your jbjf-tutorial-basics project. Each should be similar in structure, containing a default constructor and a runTask() method with a couple of log4j output statements.



By extending the AbstractTask we inherit the partial implementation of the initTask() method. The default functionality of the AbstractTask.initTask() will provide the following services for your sub-class:

- Iterates through any <resource> elements, resolving the objects as needed. The type attribute of the resource element serves two purposes, a key to the job stack or a key to a JBJF Named Resource within the Batch Definition file. In general, if type equals an existing JBJF element name, then it gets processed. For the AbstractTask, the following JBJF elements are recognized and processed by initTask():
  - log-definition
  - ftp-definition
  - plugin (prefix)
  - Any other type value is considered a custom <resource> elements are recognized by the initTask().
  - Other special resources such as sql-definition and export-definition objects need to be processed in your own task by over-riding the initTask() method. Your initTask() is free to call super.initTask() to pre-process some of the <resource> elements, but your initTask() should then process the various special objects needed by the task sub-class. See the Advanced tutorials on JBJF for further details on this.
- Check for archivist status and pull the Zipper and Archivist objects off the job-stack when the archivist is enabled.

It's critical to keep in mind the actions of initTask() as well as when and when-not it's being run. JBJF by default runs initTask() prior to runTask(), so be attentive to the subtle difference between extending AbstractTask and implementing ITask.

Creating the JBJF Batch Definition file

Now that we've got some tasks, let focus on the JBJF Batch Definition file. This is the XML file that provides all the data, parameters and task list to the batch job. As I've

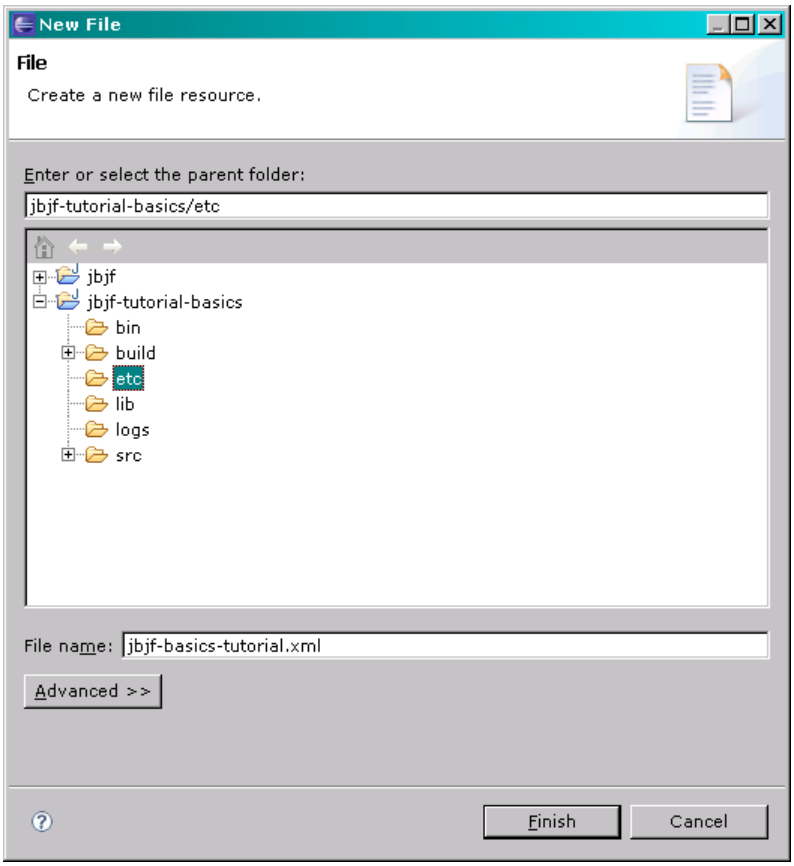
already mentioned, we won't be writing an AbstractBatch sub-class, we'll just use the DefaultBatch from the JBJF framework and supply it our JBJF Batch Definition file in the command line arguments.

For this initial JBJF Definition file we'll use a minimal number of XML elements, just enough to get the job to run. There is an example JBJF Batch Definition file included in the CVS repository (jbjf-base-definition.xml), I encourage you to download and copy it as a starting point. Look in the ./etc sub-directory. Or you can code the XML from scratch. For the purposes of this tutorial I'll build the file from scratch, placing XML segments at each stage that you may copy as we build the file up.

For this tutorial we need to implement:

- ✔ <jbjf-parameters>
- ✔ <jbjf-email>
- ✔ <jbjf-directories>
- ✔ <jbjf-tasks>
- ✔ <jbjf-logs>

Start by right-clicking on the ./etc sub-directory of the project and select New > File. In the File name: put jbjf-basics-tutorial.xml and click the Finish button.



root element

To start our XML file we need the root element. The following snippet will provide you a basic template to start from.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
FILE        : jbjf-tutorial-001.xml
DATE        : February 12, 2007
DEVELOPER   : Adym S. Lincoln
PURPOSE     :
Basic tutorial JBJF Batch Definition file
-->
<jbjf-batch-job>
</jbjf-batch-job>
```

jbjf-parameters

For this element we change the name to something like jbjf-tutorial-001, or something that indicates our first tutorial. We'll disable the archivist and enable the email.

```
    <jbjf-parameters>
        <name>jbjf-tutorial-basics</name>
        <enable-archivist>N</enable-archivist>
        <enable-email>Y</enable-email>
    </jbjf-parameters>
```

Type or copy and paste the above snippet within the root element. Your XML file should be similar to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
FILE        : jbjf-tutorial-001.xml
DATE        : February 12, 2007
DEVELOPER   : Adym S. Lincoln
PURPOSE     :
Basic tutorial JBJF Batch Definition file
-->
<jbjf-batch-job>
    <jbjf-parameters>
        <name>jbjf-tutorial-001</name>
        <enable-archivist>N</enable-archivist>
        <enable-email>Y</enable-email>
    </jbjf-parameters>

</jbjf-batch-job>
```

jbjf-email

The email parameters will differ, depending on the name of your SMTP host. Make sure to setup at least one email recipient and that the attachments attribute is missing or set to N. Also, the email recipients and the SMTP host below are not valid, you'll need to supply your own values in here.

```
    <jbjf-email>
        <notifications>
            <email attachments="N">lincolnb@hotmail.com</email>
            <email>lincolnc@hotmail.com</email>
        </notifications>
        <email-host>smtp.host.org</email-host>
        <email-sender>jbjf-tutorial-basics@hotmail.com</email-sender>
    </jbjf-email>
```

Again, type or copy and paste the XML snippet within the root element. Your XML file should be similar to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
FILE        : jbjf-tutorial-001.xml
DATE        : February 12, 2007
DEVELOPER   : Adym S. Lincoln
PURPOSE     :
Basic tutorial JBJF Batch Definition file
-->
<jbjf-batch-job>
    <jbjf-parameters>
        <name>jbjf-tutorial-001</name>
        <enable-archivist>N</enable-archivist>
        <enable-email>Y</enable-email>
    </jbjf-parameters>

    <jbjf-email>
        <notifications>
            <email attachments="N">lincolnb@hotmail.com</email>
            <email>lincolnc@hotmail.com</email>
        </notifications>
        <email-host>smtp.host.org</email-host>
        <email-sender>jbjf-tutorial-basics@hotmail.com</email-sender>
    </jbjf-email>

</jbjf-batch-job>
```

NOTE : From this point forward, each XML snippet should be placed within the root element.

jbjf-directories

For this first tutorial we use a minimal set of directories.

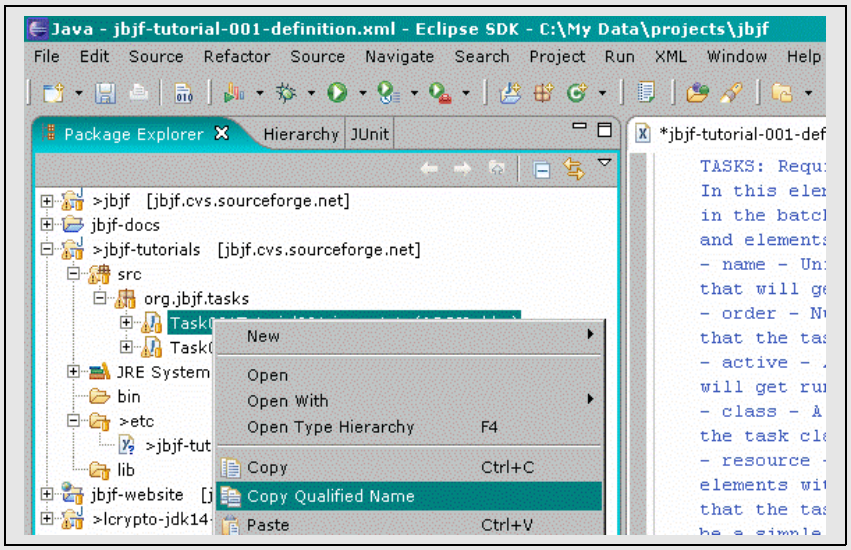
```
    <jbjf-directories>
        <directory name="base" addressing="relative">.</directory>
        <directory name="log4j" addressing="relative">etc</directory>
    </jbjf-directories>
```

jbjf-tasks

Using the fully qualified names from the classes we developed, let's setup the task-list for this batch job.

Eclipse Tip:

The fully qualified class name can be copied from the Java Perspective by simply right-clicking on the Java class and selecting "Copy Qualified Name". When you select Paste you'll need to remove the .java file extension, but it saves time.




```
    <jbjf-tasks>
        <task name="t001" order="1" active="true">
            <class>org.jbjf.tasks.Task001Basics</class>
        </task>
        <task name="two" order="2" active="true">
            <class>org.jbjf.tasks.Task002Basics</class>
        </task>
    </jbjf-tasks>
```

jbjf-logs

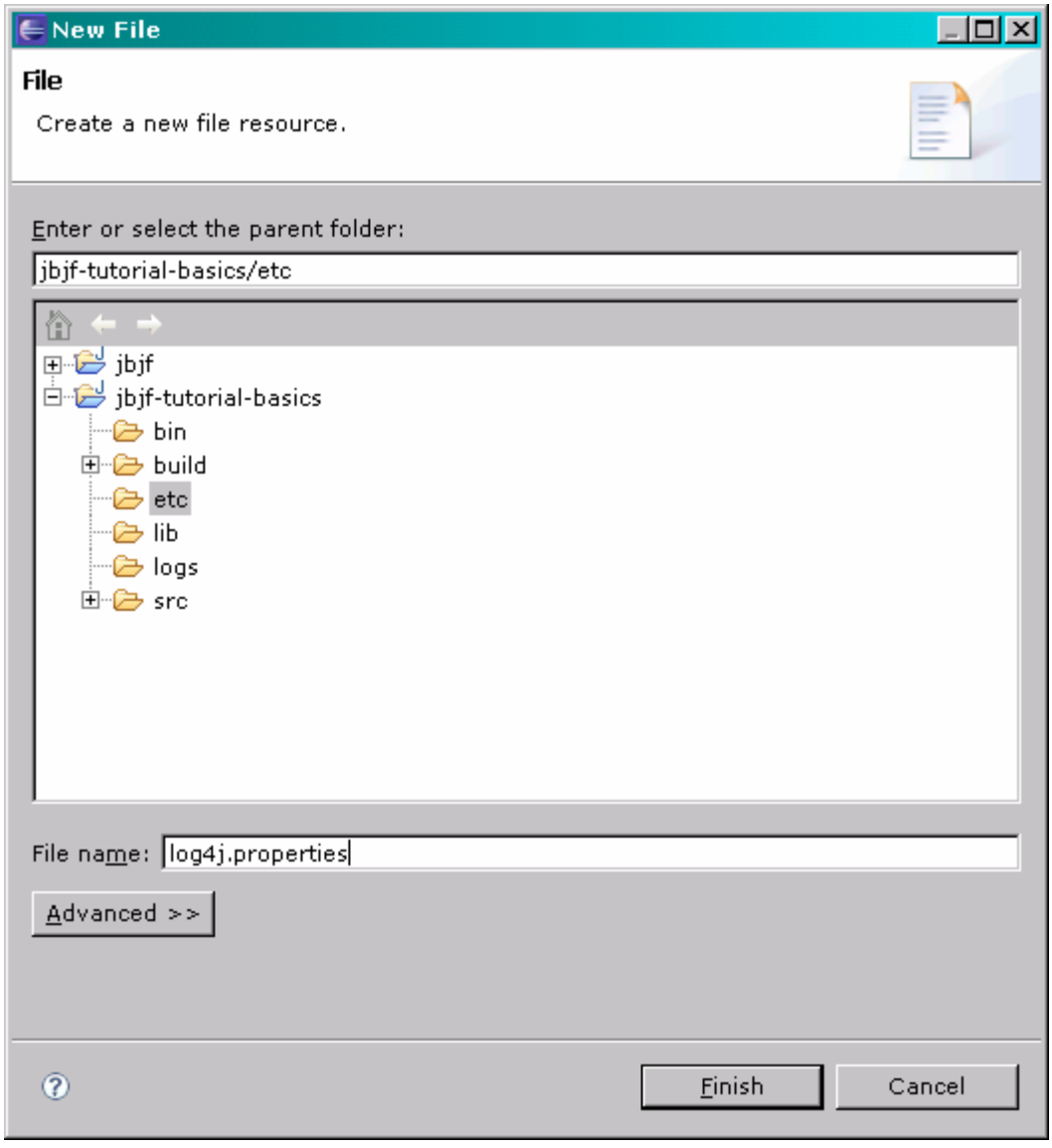
We only need a single log file for this first tutorial. We'll create the log4j.properties file in a little bit.

```
    <jbjf-logs>
        <log-definition name="default">
            <log4j category="org.jbjf.tutorial">./etc/log4j.proper-
ties</log4j>
        </log-definition>
    </jbjf-logs>
```

Create a new sub-directory in the ${project-dir} called etc...${project-dir}/etc. Then place your JBJF Batch Definition file in here. This completes the JBJF Batch Definition setup. In the next section we'll create a simple log4j.properties file using the same name as indicated in the `<jbjf-logs>` entry.

Creating the Log4j Properties file

As a final step, we need to create a log4j properties file. Without diving into all the details of log4j, we really just need to match up our log4j category in the properties file with the category attribute we put in the `<log4j>` entry in the `<jbjf-logs>` XML element. The following is a workable properties file. Again, right-click on the ./etc folder in the project and select New > File. In the File dialog, name the file log4j.properties:



Then copy the following code snippet and paste it into the log4j.properties file:

```
# See http://logging.apache.org/log4j/docs/manual.html
# for log4j configuration.

# See http://logging.apache.org/log4j/docs/api/org/apache/log4j/PatternLay-
out.html
# for conversion pattern formatting.
```

```
log4j.category.org.jbjf.tutorial=DEBUG, logfile, commandline

log4j.appender.logfile=org.apache.log4j.DailyRollingFileAppender
log4j.appender.logfile.file=./logs/jbjf-tutorial-basics.log
log4j.appender.logfile.DatePattern='.'yyyy-MM-dd
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
log4j.appender.logfile.layout.ConversionPattern=%d [%t] %-5p %c - %m%n

log4j.appender.commandline=org.apache.log4j.ConsoleAppender
log4j.appender.commandline.layout=org.apache.log4j.PatternLayout
log4j.appender.commandline.layout.ConversionPattern=%d [%t] %-5p %c - %m%n
```

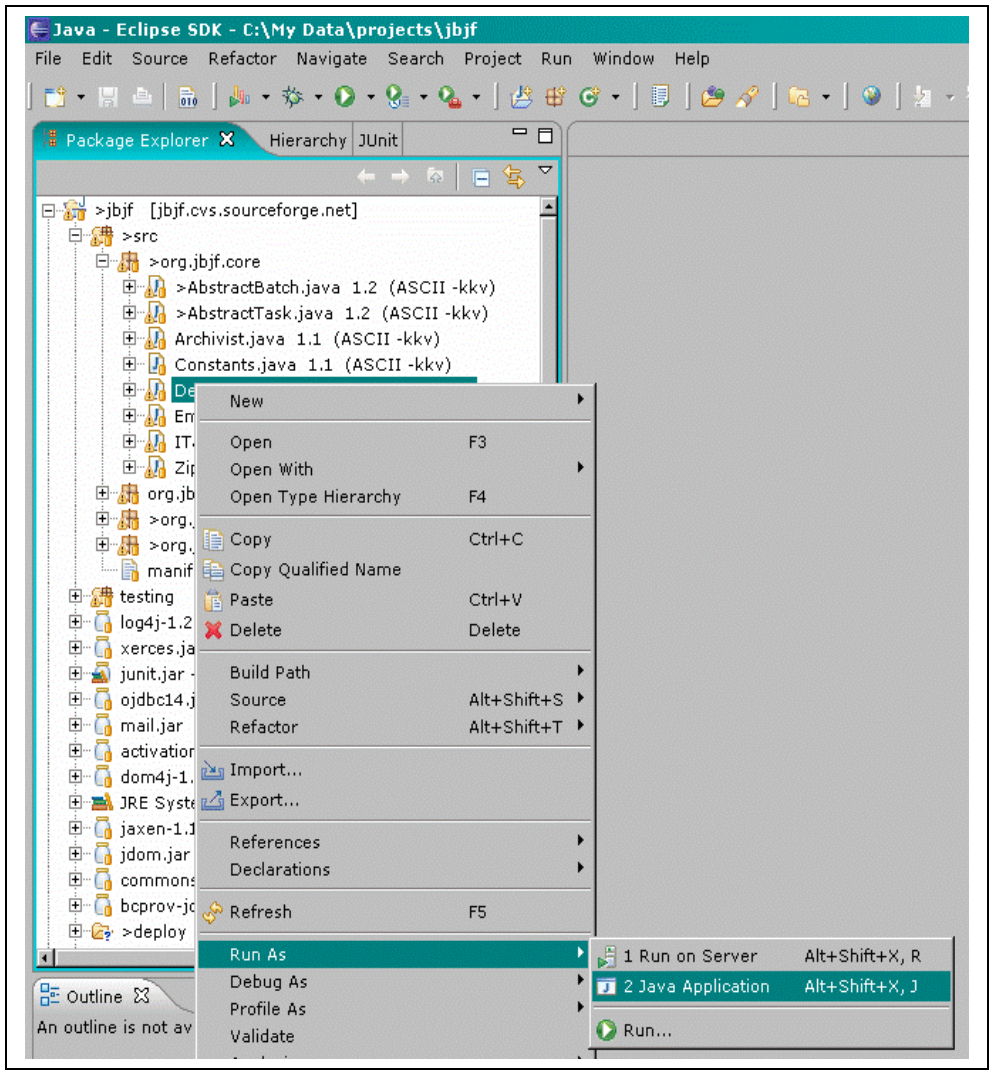
I've **bolded** and *italicized* the key entries for the properties file. You'll not the need to create a sub-directory in the ${project-dir} called logs...${project-dir}/logs.

Basics Tutorials - Running the Batch Job

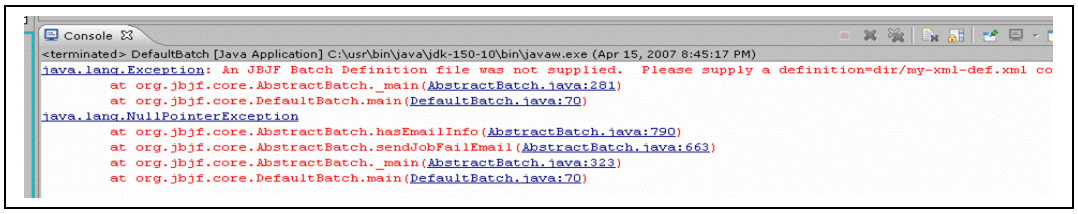
There are a couple of ways to execute the batch job. The easiest way involves using the Eclipse Launch facility. Another way is to use the traditional jarfile and script file process. More involved, but not impossible. The basics tutorial includes code, jarfiles and script files for both approaches.

Launch Facility

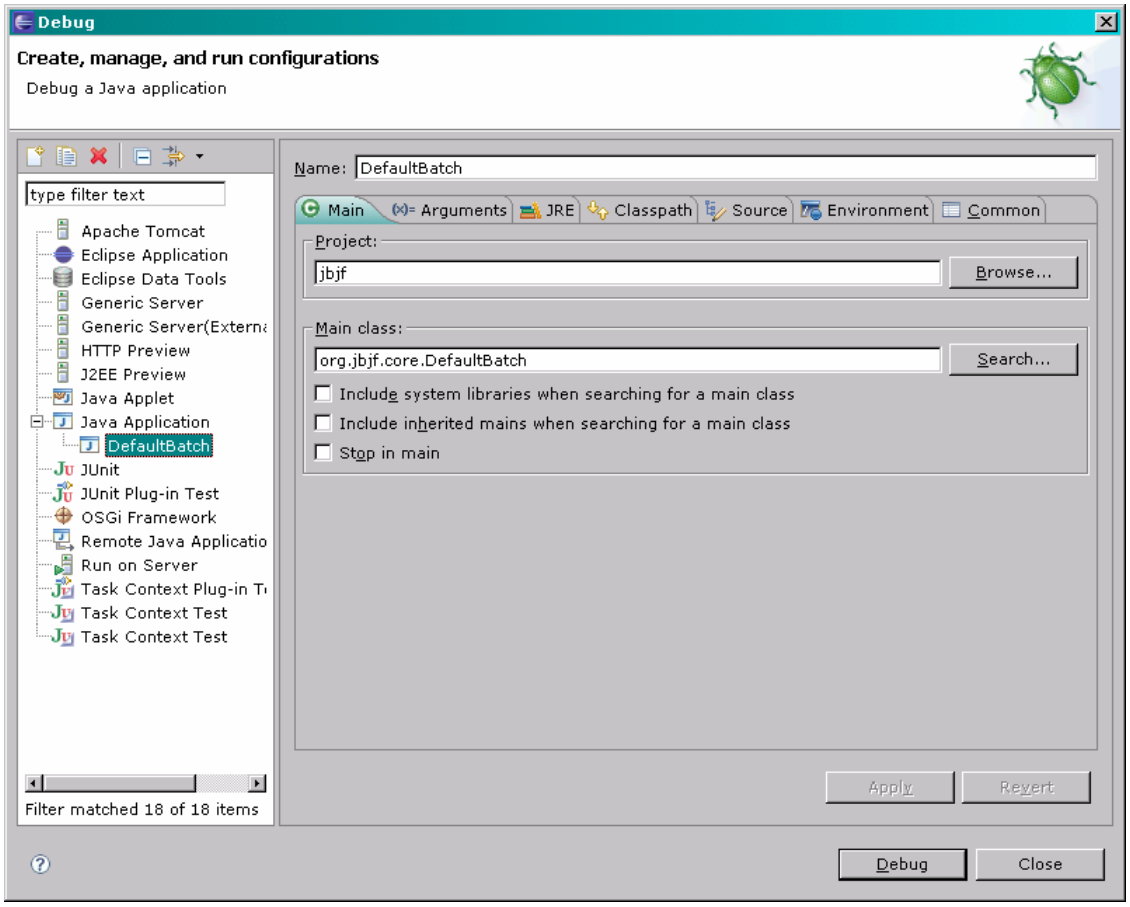
The easy approach is to use the Launch Facility built into Eclipse. From the jbjf project locate the DefaultBatch class...contained within the org.jbjf.core package. Right click on the DefaultBatch class and select Run As -> Java Application.



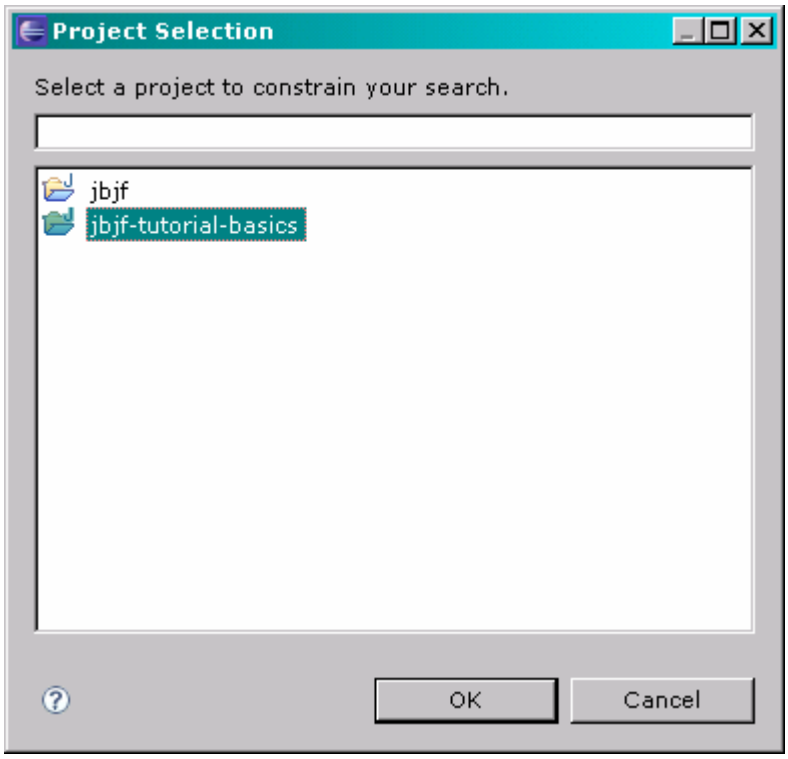
This attempts to run the DefaultBatch, but it will throw an exception immediately and this will show up in the console view of Eclipse:



The Exceptions occur because the JBJF Batch Definition file wasn't supplied via the command line, but this does create a Launch Configuration for us. To correct the exceptions bring up the Launch Configurations dialog...Run menu -> Open Run Dialog... OR Run menu -> Open Debug Dialog...

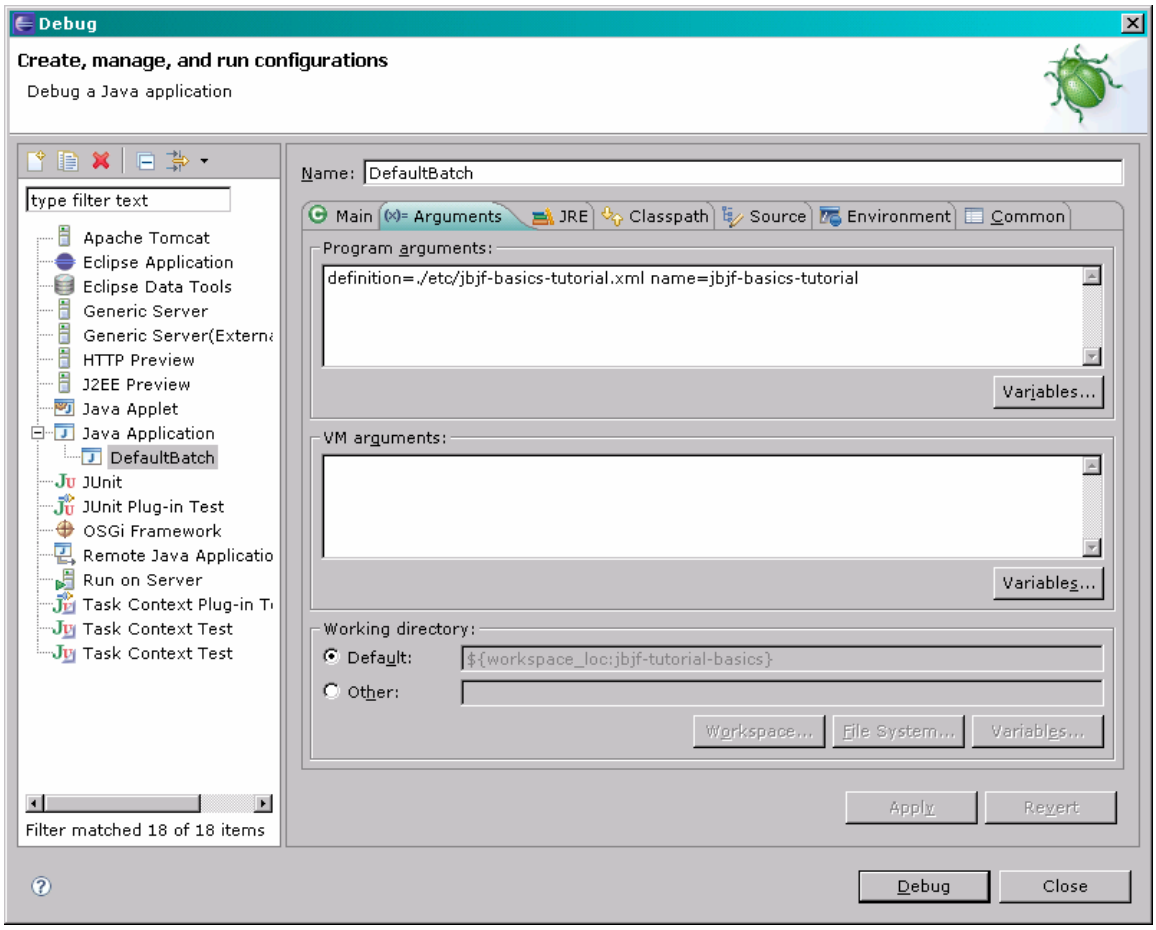


We need to make the following changes and additions. First, the Project: assigned to the Launch Configuration is jbjf. We need to change it to our jbjf-tutorials-basics project. Click on the Browse button and select the jbjf-tutorial-basics Project.



Next, we need to supply two command line parameters to the DefaultBatch class:
- definition=./etc/jbjf-basics-tutorial.xml
- name=jbjf-basics-tutorial

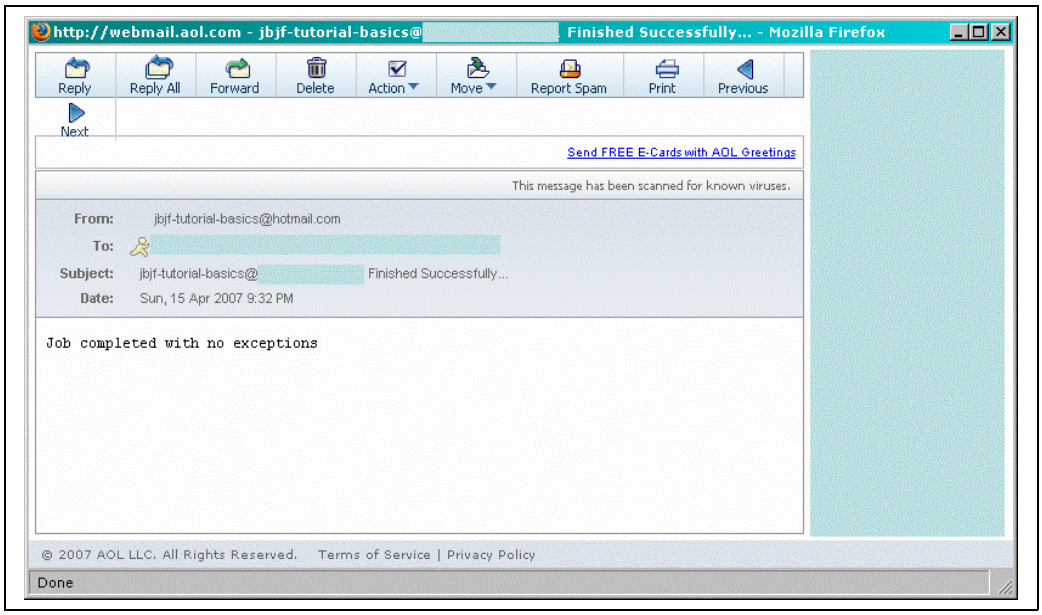
Ordinarily we would only need to supply a single command line argument, the JBJF Batch Definition file. But because we didn't write our own AbstractBatch sub-class, we are using the DefaultBatch sub-class in the jbjf Project and it requires a definition file as well as a batch job name. That said, click on the Arguments Tab and put in the two command line arguments.



Click the Apply button and then the Debug/Run button. This will rerun the application now, passing the command line argument as requested. Logging results and output will appear on the Console View of Eclipse...the amount of output will depend on the level of logging you set in the Log4j properties file. In my case, I have the DEBUG log level and my output is as follows:

```
2007-04-16 09:27:50,394 [main] INFO  org.jbjf.tutorial - Starting jbjf-tutorial-basics...
2007-04-16 09:27:50,394 [main] INFO  org.jbjf.tutorial - Work Unit ... Starting ...class
org.jbjf.tasks.Task001Basics
2007-04-16 09:27:50,394 [main] DEBUG org.jbjf.tutorial - Initialize Work Unit...Start...
2007-04-16 09:27:50,394 [main] DEBUG org.jbjf.tutorial - Initialize Work Unit...Com-
plete...
2007-04-16 09:27:50,394 [main] DEBUG org.jbjf.tutorial - Task
[ Task001Basics() ]...Starting...
2007-04-16 09:27:50,394 [main] DEBUG org.jbjf.tutorial - Task [ Task001Basics() ]...Com-
plete...
2007-04-16 09:27:50,394 [main] INFO  org.jbjf.tutorial - Work Unit ... Complete ...
2007-04-16 09:27:50,394 [main] INFO  org.jbjf.tutorial - Finishing jbjf-tutorial-basics...
```

Finally, if your SMTP email host, recipients and sender properties are correct, then you'll receive an email at those email recipients that indicates a successful batch job run. In the example below, I sent my email to a freebie recipient:



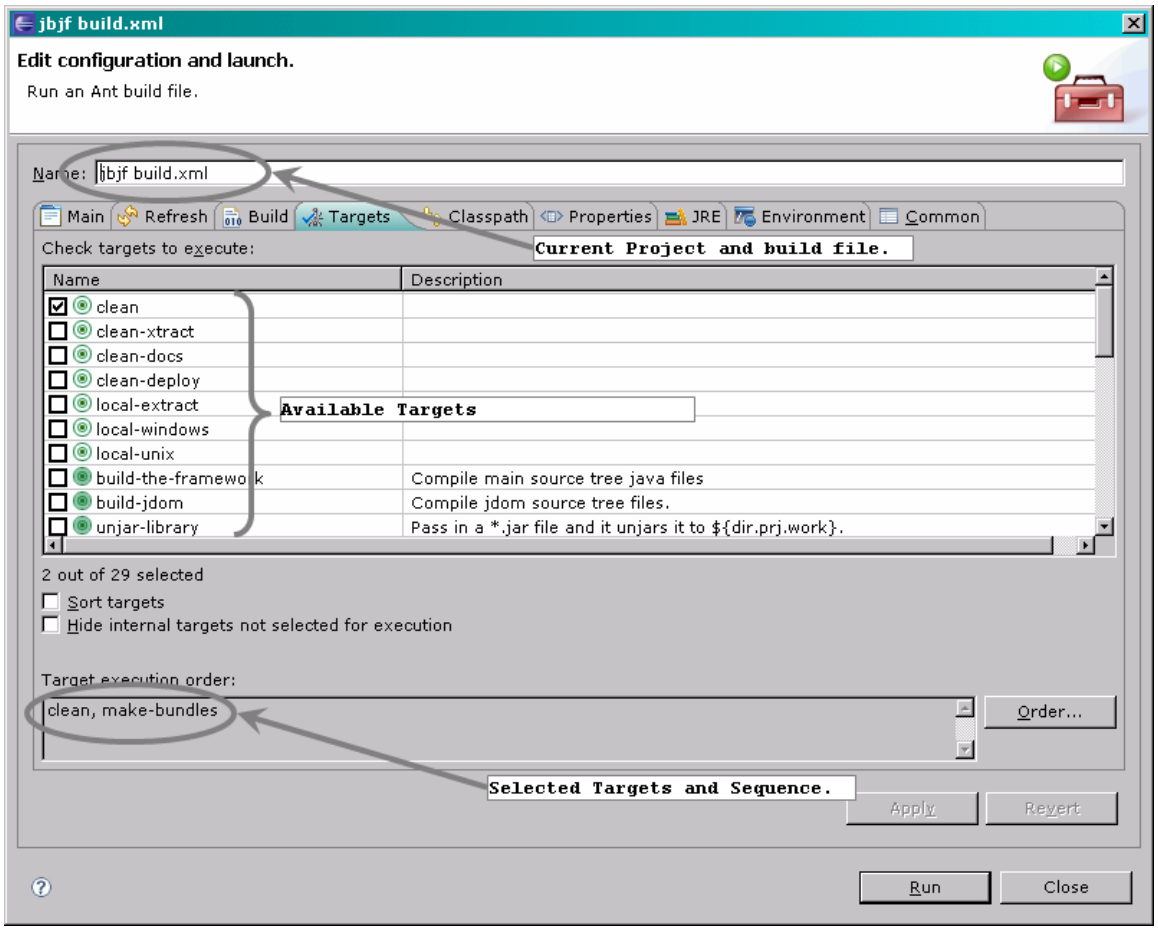
Console Configuration

The traditional approach is to compile our code into jarfiles then write script files that setup a CLASSPATH and call into the main() method of the Java class. To work through this section you should be familiar with Ant build files and scripting languages such as Windows BAT/CMD, Linux SH/BASH.

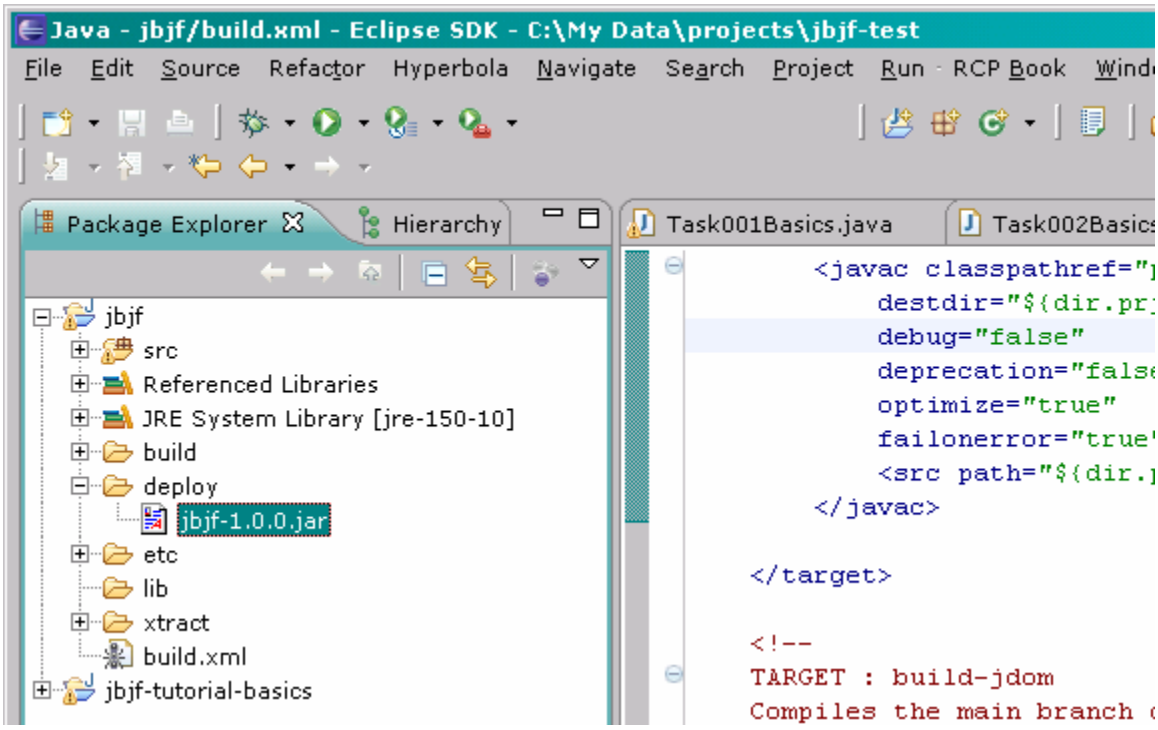
If you are currently using a source package with your tutorials, you'll need to work through this section. If you're already using a JBJF binary package, you can skip to the next section.

Start by building the jbjf into a jarfile. Move to the jbjf Project and right-click on the build.xml file contained at root of the Project directory. Select the Run As -> Ant Build... This brings up the Ant build dialog. Depending on your Eclipse preferences/setting, you should see the Targets Tab. You need to select two targets in the following order, by default the make-local target will already be selected. As such, you need to uncheck it, then check the clean and make-bundles to get the proper sequence:

- ✔ clean
- ✔ make-bundles

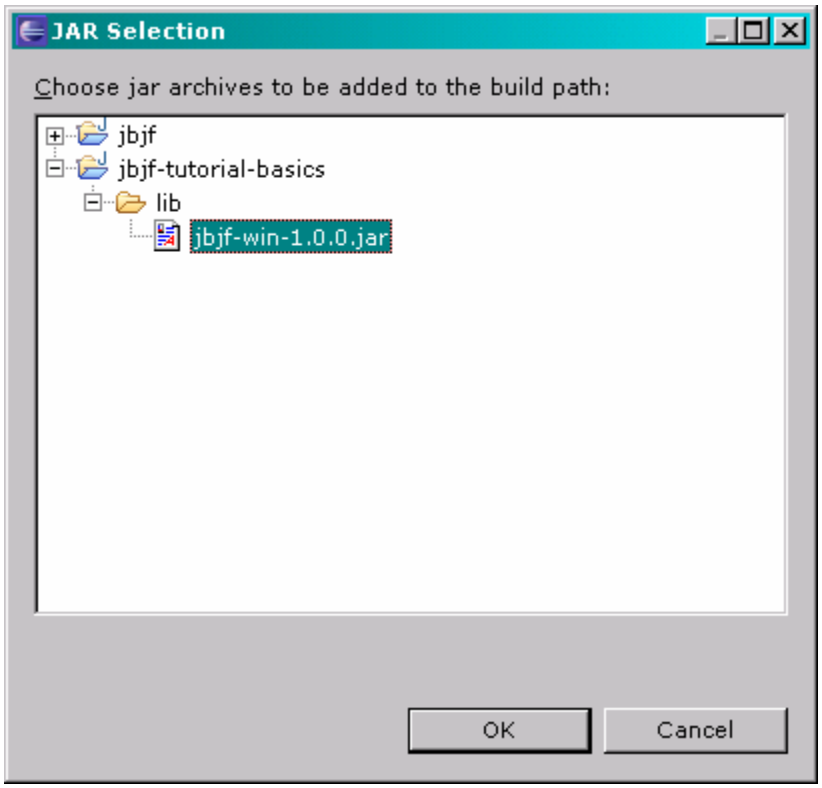


Click the Run button to have Ant compile and build the jarfiles. The make-bundles Ant target builds bundled jarfiles, the jbjf code along with all the supporting libraries for jbjf. There are multiple jarfiles for the Windows and *nix platforms. When working with the jarfile, make sure you copy the correct platform file. When the build completes the jbjf-<platform>-<ver>.jar file will be in the ./deploy directory of the jbjf project. You may have to select the jbjf project and press the F5 key to refresh the project. This will reveal the ./deploy directory so you can view the jbjf-<platform>-<ver>.jar file.

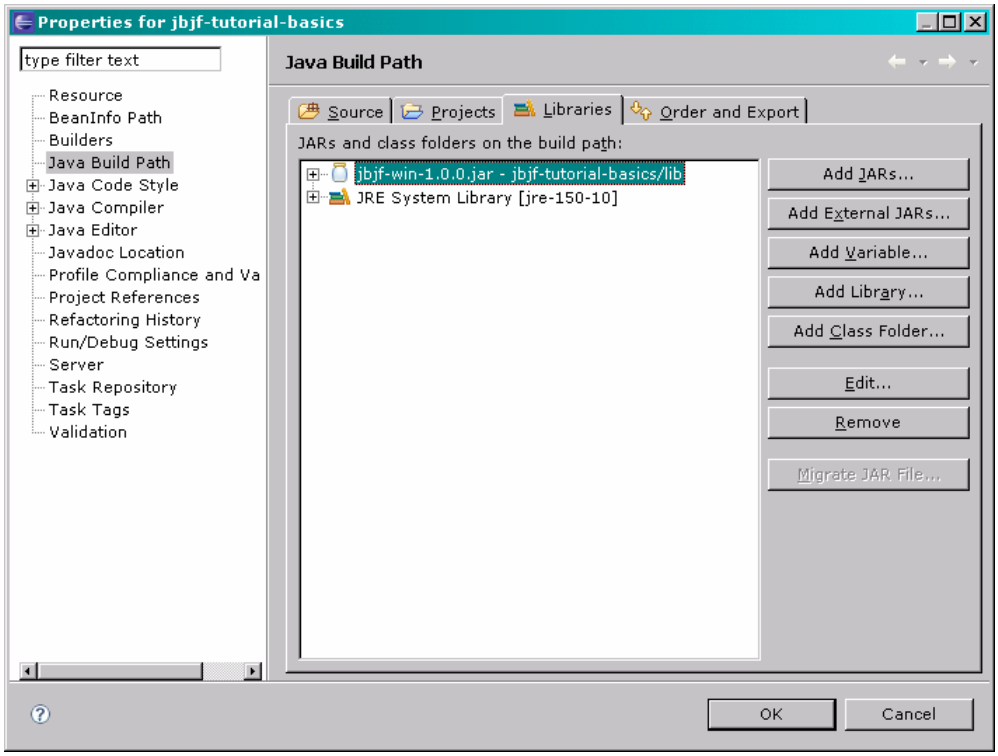


Copy the jbjf-<platform>-<ver>.jar file over to the ${tutorial-project-dir}/lib. From here, we'll be building a jarfile with our tutorial code in it, then use both jarfiles in the script files. When using the Ant build.xml file contained within the ${tutorial-project-dir}, make sure the jbjf-<platform>-<ver>.jar file is placed within the ${tutorial-project-dir}/lib directory. The tutorial Ant build file expects the jbjf-<platform>-<ver>.jar file to be in the ./lib directory of the tutorial project directory. Otherwise, adjust the Ant build.xml file to coincide with the correct location.

Finally, we need to swap out the source JBJF Project with the new jbjf-<platform>-<ver>.jar file. Right-click on the ${tutorial-project-dir} and select Properties from the pop-up menu. In the Properties dialog click the Java Build Path item. Click the Projects tab, select/highlight the jbjf project and hit the Delete key. Now click the Libraries tab, click the Add JARs button and locate the jbjf-<platform>-<ver>.jar file in the tutorials ./lib directory.



Click the OK button to add the jbjf-<platform>-<ver>.jar file to the tutorials project reference libraries.



Click the OK button to close out of the Properties dialog.

Before we can build the tutorials project with Ant, we need an Ant build.xml file. The following is a simple build.xml file that will do the job:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--

USAGE:
------------------------------------------------------------
Builds and packages the XML Batch Framework for a Java batch job.

-->

<project name="jbjf" default="make-local" basedir=".">
    <description>Author: Adym S. Lincoln</description>
    <description>Created: Nov 28, 2006</description>
    <description>Copyright: @2006 Adym S. Lincoln ALL RIGHTS RESERVED</description>
    <description>File used to build and package the Java Batch Job Framework</description>

    <property name="dir.prj.base"          value="." />
    <property name="dir.prj.src"           value="${dir.prj.base}/src" />
    <property name="dir.prj.libs"          value="${dir.prj.base}/lib" />
    <property name="dir.prj.xtract"        value="${dir.prj.base}/xtract" />
    <property name="dir.prj.etc"           value="${dir.prj.base}/etc" />
    <property name="dir.prj.deploy"        value="${dir.prj.base}/deploy" />
    <property name="dir.prj.jdom"          value="${dir.prj.base}/jdom" />
    <property name="dir.prj.build"         value="${dir.prj.base}/build" />
    <property name="dir.prj.docs"          value="${dir.prj.base}/docs" />
    <property name="dir.prj.api"           value="${dir.prj.docs}/api" />

    <property name="cvs.pkg.main"          value="jbjf-basics-tutorial" />
    <property name="cvs.pkg.src"           value="${cvs.pkg.main}/src" />
    <property name="cvs.pkg.jdom"          value="${cvs.pkg.main}/jdom" />
    <property name="cvs.pkg.etc"           value="${cvs.pkg.main}/etc" />
    <property name="cvs.pkg.bin"           value="${cvs.pkg.main}/bin" />
    <property name="cvs.pkg.lib"           value="${cvs.pkg.main}/lib" />
    <property name="cvs.pkg.test"          value="${cvs.pkg.main}/testing" />

    <property name="dir.prj.work"          value="${dir.prj.xtract}/${cvs.pkg.main}" />
    <property name="dir.prj.compile"       value="${dir.prj.work}/classes" />
    <property name="dir.prj.compile.lib"   value="${dir.prj.work}/lib" />
    <property name="dir.prj.compile.src"   value="${dir.prj.work}/src" />

    <!--
    JAR Deployment:
    Name of the final jar file for JBJF.
    -->
    <property name="jar.file.main"         value="jbjf-basics-tutorial" />
    <property name="jar.file.version"      value="1.0.0" />

    <path id="project.class.path">
        <pathelement location="${dir.prj.libs}/jbjf-win-1.0.0.jar" />
        <pathelement location="${dir.prj.work}/classes/**" />
    </path>

    <target name="clean">
        <echo message="Cleanup work environment..." />
        <antcall inheritAll="false" target="clean-xtract" />
        <antcall inheritAll="false" target="clean-docs" />
        <antcall inheritAll="false" target="clean-deploy" />

        <echo message="Delete ${dir.prj.build}/" />
        <delete>
          <fileset
              dir="${dir.prj.build}/"
              includes="**/*"/>
        </delete>
    </target>

    <target name="clean-xtract">
        <echo message="Delete ${dir.prj.xtract}" />
        <delete dir="${dir.prj.xtract}" />
    </target>
```

```
    <target name="clean-docs">
        <echo message="Delete ${dir.prj.api}" />
        <delete dir="${dir.prj.api}" />
    </target>

    <target name="clean-deploy">
        <echo message="Delete ${dir.prj.deploy}" />
        <delete dir="${dir.prj.deploy}" />
    </target>

    <!--
    Copy the source files from the local location into a working
    directory tree.
    -->
    <target name="local-extract">
        <echo message="Local Extract : ${dir.prj.xtract}" />
        <mkdir dir="${dir.prj.xtract}" />
        <mkdir dir="${dir.prj.work}" />

        <copy toDir="${dir.prj.work}" overwrite="yes" verbose="yes">
            <fileset dir="${dir.prj.base}">
                <exclude name="build/**" />
                <exclude name="bin/**" />
                <exclude name="doc/**" />
                <exclude name="docs/**" />
                <exclude name="etc/**" />
                <exclude name="lib/**" />
                <exclude name="xtract/**" />
                <exclude name="logs/**" />
                <exclude name=".project" />
                <exclude name=".classpath" />
                <exclude name="build.xml" />
                <exclude name="javadoc.xml" />
            </fileset>
        </copy>

        <!--
        CR, CRLF, LF : Convert CR, CRLF, LF to the proper setting
        based on the OS.  Also, replace <TAB> with spaces.
        -->
        <echo message="fixcrlf      : ${dir.prj.work}"/>
        <fixcrlf
            srcdir="${dir.prj.work}"
            includes="**/*"
        />

    </target>

    <!--
    TARGET : build-engine
    Compiles the main branch of source code for all java classes
    that make up the feed engine.
    <change>
    1.0.1; ASL; 04/04/2007; Removed the jdom dependency...
    </change>
        depends="build-jdom"
    -->
    <target name="build-the-framework" description="Compile main source tree java files">
        <echo message="destdir        : ${dir.prj.compile}" />
        <echo message="src path       : ${dir.prj.xtract}/${cvs.pkg.src}" />
        <mkdir dir="${dir.prj.compile}" />
        <javac classpathref="project.class.path"
            destdir="${dir.prj.compile}"
            debug="false"
            deprecation="false"
            optimize="true"
            failonerror="true">
            <src path="${dir.prj.xtract}/${cvs.pkg.src}" />
        </javac>

    </target>

    <target
        name="make-local"
        depends="clean,local-extract,build-the-framework">

        <antcall inheritAll="false" target="jar-the-project">
            <param name="jar.file.name" value="${jar.file.main}.jar" />
```

```
        </antcall>

    </target>

    <!--
    TARGET : jar-the-engine
    Packages the classes/ into a single JAR file for deployment.
    -->
    <target name="jar-the-project">
        <echo message="deploy dir: ${dir.prj.deploy}" />
        <echo message="destfile  : ${dir.prj.deploy}/${jar.file.engine}" />
        <echo message="basedir   : ${dir.prj.compile}" />
        <echo message="manifest  : ${dir.prj.compile.src}/manifest" />
        <mkdir dir="${dir.prj.deploy}" />

        <jar
            destfile="${dir.prj.deploy}/${jar.file.name}"
            basedir="${dir.prj.compile}"
            includes="**"
            update="true">
        </jar>

    </target>

</project>
```

Create a new file in the tutorials project at the root of the project and paste the above build.xml file into the file. Using the same technique as the JBJF build, compile and build the tutorial project...right-click on the build.xml file and select Run As -> Ant Build... Then select the clean and make-local in that order. Once the build completes, you should have a jarfile called jbjf-basics-tutorial.jar file in a ./deploy directory.

Now let's focus on the script files that will run our batch job. For this first tutorial we'll develop a single script file and since I'm on Windows I'll be using BAT/CMD scripting. Conversion of BAT/CMD into SH/BASH is simple enough. To run a java class from a command line we really only require a few items:

- ✔ Java Class name
- ✔ CLASSPATH
- ✔ Command line arguments

For Windows, you'll need to use the "\" in place for directory path separators. One very big gripe I have with Windows is this particular issue. Not only does the "\" cause you to rewrite the code when you port it to BASH/SH, but it's not consistent between versions of Windows. I've found some version of Windows only require a single "\" into Java, while other versions require two, "\\". Anyway, be attentive to this issue as you may encounter an exception related to this.

The following is a sample BAT/CMD script file that will allow you to run the tutorial code:

```
rem *
rem ***************************************************************
rem Simple command console script file to run a tutorial batch job.
rem ***************************************************************
rem *
set COMMAND_CLASS=org.jbjf.core.DefaultBatch
@echo off

rem Capture the current CLASSPATH variable...
rem
```

```
set tmpCLASSPATH=%CLASSPATH%

rem Set our CLASSPATH to the needed jarfiles...
rem
set CLASSPATH=.;..\lib\jbjf-<platform>-<ver>.jar;..\deploy\jbjf-basics-tutorial.jar;
%CLASSPATH%

java -cp "%CLASSPATH%" %COMMAND_CLASS% "definition=..\\etc\\jbjf-basics-tutorial.xml"
"name=jbjf-tutorial-basics"

if errorlevel 1 goto failed

:successful
    @echo on
    @echo Feed has completed
    @goto done

:failed
       @echo on
       @echo Feed has failed
       @goto done

:end-if-usage

:done
@echo off
set CLASSPATH=%tmpCLASSPATH%
@echo on
```

Create a new file called run-tutorial.bat in the ./bin tutorial directory. Paste the above code into the file and save the file. You need to edit the code and replace the <plat>-<ver>.jar with the correct platform and version for the jarfile. Next, you'll need to adjust the log4j setting in the jbjf-basics-tutorial.xml file, change the ./etc/log4j.properties to ../etc/log4j.properties. Next, adjust the ./etc/log4j.properties file, set the ./logs/jbjf-basics-tutorial.log to ../logs/jbjf-basics-tutorial.log.

To run this script file open up a command prompt console and change the directory to the ${jbjf-tutorial-dir}/bin directory. Do a dir command to get a file list...your script file should be on the list. Now just execute the script file. If all goes well, you should see output similar to the following:

```
C:\My Data\projects\jbjf\jbjf-tutorial-basics\bin>run-tutorial.bat

C:\My Data\projects\jbjf\jbjf-tutorial-basics\bin>rem *

C:\My Data\projects\jbjf\jbjf-tutorial-basics\bin>rem ****************************************

C:\My Data\projects\jbjf\jbjf-tutorial-basics\bin>rem Simple command console script file to run a
tutorial batch job.

C:\My Data\projects\jbjf\jbjf-tutorial-basics\bin>rem ****************************************

C:\My Data\projects\jbjf\jbjf-tutorial-basics\bin>rem *

C:\My Data\projects\jbjf\jbjf-tutorial-basics\bin>set COMMAND_CLASS=org.jbjf.core.DefaultBatch
2007-04-16 22:22:34,423 [main] INFO  org.jbjf.tutorial - Starting jbjf-tutorial-basics...
2007-04-16 22:22:34,423 [main] INFO  org.jbjf.tutorial - Work Unit ... Starting ...class
org.jbjf.tasks.Task001Basics
2007-04-16 22:22:34,423 [main] DEBUG org.jbjf.tutorial - Initialize Work Unit...Start...
2007-04-16 22:22:34,433 [main] DEBUG org.jbjf.tutorial - Initialize Work Unit...Complete...
2007-04-16 22:22:34,433 [main] DEBUG org.jbjf.tutorial - Task [ Task001Basics() ]...Starting...
2007-04-16 22:22:34,433 [main] DEBUG org.jbjf.tutorial - Task [ Task001Basics() ]...Complete...
2007-04-16 22:22:34,433 [main] INFO  org.jbjf.tutorial - Work Unit ... Complete ...
2007-04-16 22:22:34,433 [main] INFO  org.jbjf.tutorial - Finishing jbjf-tutorial-basics...
Feed has completed

C:\My Data\projects\jbjf\jbjf-tutorial-basics\bin>
```

When you run this script you may encounter the following exception:

```
<snip>
C:\My Data\projects\jbjf\jbjf-tutorial-basics\bin>set
COMMAND_CLASS=org.jbjf.core.DefaultBatch
log4j:ERROR Could not read configuration file [./etc/log4j.properties].
java.io.FileNotFoundException: .\etc\log4j.properties (The system cannot find the path
specified)
        at java.io.FileInputStream.open(Native Method)
        at java.io.FileInputStream.<init>(Unknown Source)
        at java.io.FileInputStream.<init>(Unknown Source)
        at
org.apache.log4j.PropertyConfigurator.doConfigure(PropertyConfigurator.java:297)
        at org.apache.log4j.PropertyConfigurator.configure(PropertyConfigurator.java:315)
        at org.jbjf.util.APILog4j.<init>(Unknown Source)
        at org.jbjf.core.AbstractBatch.initBatch(Unknown Source)
        at org.jbjf.core.AbstractBatch._runBatch(Unknown Source)
        at org.jbjf.core.AbstractBatch._main(Unknown Source)
        at org.jbjf.core.DefaultBatch.main(Unknown Source)
log4j:ERROR Ignoring configuration file [./etc/log4j.properties].
log4j:WARN No appenders could be found for logger (org.jbjf.tutorial).
log4j:WARN Please initialize the log4j system properly.
Feed has completed

C:\My Data\projects\jbjf\jbjf-tutorial-basics\bin>
</snip>
```

To correct this, edit the JBJF Batch Definition file and adjust the log4j properties file directory path. When running the tutorial from Eclipse's Launch Facility, the working directory is ${project-dir}. But the console command script file runs from ${project-dir}/bin. Thus, the log4j properties file is actually at ../etc/log4j.properties.

Other Resources

JBJF Website – http://jbjf.sourceforge.net/
JBJF Tutorials/Documentation - http://jbjf.sourceforge.net/documentation.html
Tutorial Source Code - http://jbjf.sourceforge.net/downloads/jbjf-tutorial-basics.zip