

QuickStart Tool

Eclipse

By Bill Dudney

Getting a good start in any new technology or programming language often depends on finding the best available information. The Builder.com QuickStart Tools give you the information you need to quickly grasp the fundamentals of developing in a new IDE, using a new programming language, or working with a new development tool.

Besides explaining the basics, the Builder.com QuickStart Tools show common tasks, expose strengths and weaknesses, demonstrate some of the best uses of the technology, and list a variety of other online and offline resources that can help you build a solid foundation of practical knowledge.

Table of Contents

Fundamentals.....3

Creating and running a Java project5

Strengths7

Weaknesses7

Online resources8

Other resources8

Additional articles8

About Builder.com.....9

Fundamentals

Eclipse.org was founded in November 2001, principally by IBM in the form of a donation of software valued at \$40 million. The organization is led by a “board of stewards,” which is responsible for setting the overall direction of the Eclipse platform. The main focus of Eclipse, in contrast to other open source consortiums, is to build a robust platform for creating commercial tools. Many of the tool vendors that are part of the Eclipse board of stewards have commercial products based on Eclipse (IBM’s WSAD and Metanology’s MDE for J2EE, for example). Although other open source communities have no problem with commercial entities using their software, they were not formed with this as a main goal, as Eclipse.org was.

The question then becomes, What does this mean to us as consumers of Eclipse? The biggest gain we get is a managed release cycle that more or less meets the advertised dates. Many of the contributors to Eclipse get to work on the code as a full-time job instead of having to make time outside of normal working hours.

In addition, because many of the members have products that depend on a robust infrastructure, they are financially motivated to have their employees work on Eclipse. All in all, this is a very beneficial setup for consumers of Eclipse—not only is there a great group of volunteers, but there is also a lot of financial clout behind the ongoing development of the platform.

Getting Eclipse

Let’s get started using Eclipse to develop Java code. The first step, of course, is to download and install the default IDE. You will find all of the currently available versions of Eclipse on the Eclipse download page. Once you have selected a build from the main download page, you will be taken to that build’s specific download page. This page provides many different options for related packages to download. The typical option is to download the Eclipse SDK. The SDK is the most common way to use Eclipse; this package has the Java Development Tools (JDT) and the Eclipse Plug-in Development Environment (PDE) plug-ins already loaded.

The download options are listed in a table. The first column indicates the status of the build (the definition of the status depends on which version you are downloading, but typically a green check is good and a red X is bad). The second column is the platform/OS for which the build is intended. The third column lets you select between using HTTP or FTP to download. The final column is the name of the file that you will download.

Installing and running Eclipse

Now that you have your desired version of Eclipse, install it by unzipping the download into the directory of your choice. In Windows, the typical install is into *C:\DevTools\Eclipse-version*, and on the Macintosh, it is into */Users/Shared/Application/Eclipse-version*. You can, of course, install it in any directory you’d like.

In Windows, you need to have a *java.exe* somewhere in your system path for Eclipse to start up properly. If you don’t, Eclipse will fail to start, and a message will appear stating that it can’t find a JRE to run within. If that happens, just click OK and add the JRE and JDK bin directories to your system path.

Now that the IDE is installed, let’s take a quick look at some platform-specific options for adjusting the runtime of Eclipse.

Platform	Process
Windows	Create a shortcut for <i>eclipse.exe</i> , select Properties from the context menu (usually a right-click of your mouse), and add the command line arguments to the end of the Target: text field. If you are going to change the location of the workspace directory, specify the Start In: directory as the parent directory of your workspace.
Macintosh OS X	Go to the <i>Eclipse.app</i> bundle in the finder, select Show Package Contents, and then open the file <i>Content/Info.plist</i> .
Linux	Create an alias in your shell’s startup script (i.e., for bash, <i>.bash_profile</i>) that contains the Eclipse executable and the options.

Once you have identified the OS-specific place to put the options, they are the same on every OS. Generally, you specify the options with the following format: `eclipse [platform options] [-vmargs VMOptions]`. The *platform options* are passed to Eclipse, and the *VMOptions* (everything past `-vmargs`) are passed to the Java Virtual Machine (JVM) that is running Eclipse. The options for the JVM obviously depend on the JVM that you are running.

The one you will most likely have to change is the amount of memory allocated to the JVM. You do that with the `-Xms<size>` and `-Xmx<size>` options available on most JVMs. Eclipse 3.0 is shipping with a default setting of 150 MB max memory used, which is fine for just getting started and for small projects, but for doing serious development you should increase the memory to 256 MB. To do so, specify `-Xmx256MB` on the command line.

Eclipse options

Option	Description
<code>-application [App ID]</code>	<i>App ID</i> specifies the application to run and defaults to the Eclipse workbench. This option will become particularly important for development with and use of the Rich Client Platform (which we will talk briefly about later).
<code>-boot [boot code path]</code>	Allows you to change the location of Eclipse's startup classes, which are contained in <i>boot.jar</i> and <i>startup.jar</i> . The typical developer using Eclipse will not use this option.
<code>-consolelog</code>	Puts the error log onto the console in addition to writing it to Eclipse's internal log. If you are having a problem starting Eclipse, this is a good option for debugging what has gone wrong.
<code>-data [workspace path]</code>	The location of the workspace. This is where Eclipse keeps its metadata and is the most likely option to be specified by developers using Eclipse. (We will talk more about this option later.)
<code>-debug [options path]</code>	Specifying this option with a path to an options file will allow you to specify debugging options for the Eclipse platform. This options file is very useful if you are developing plug-ins for Eclipse.
<code>-dev [classpath entries]</code>	This option allows you to add classpath entries to the classpath of every plug-in.
<code>-nosplash</code>	This option will keep the Eclipse splash page from appearing on startup.
<code>-os [os-id]</code>	You only need to specify this option if Eclipse can't figure it out on its own. This usually happens when a new OS is added to the list of supported OSs. Most installations will not have to specify this option.
<code>-vm [vm-path]</code>	This option allows you to specify an alternate JVM to use for running Eclipse. In Windows, if you do not specify this, Eclipse will use the first <code>java.exe</code> it finds in the System PATH variable.
<code>-ws [ws-id]</code>	You only need to specify this option if Eclipse can't figure it out on its own. This usually happens when a new windowing system is added to the list of supported windowing systems. Most installations will not have to specify this option.

Users of Eclipse often use the `-data` option to specify an alternate place for Eclipse to store their projects. Changing this location allows you to put your projects into a directory that will be backed up on a regular basis. Whether or not you use the default location for your projects, make sure to back up the entire workspace directory on a regular basis. Eclipse keeps important metadata in the `.metadata` directory in your workspace. If something goes wrong with this data, Eclipse sometimes won't start, and you will have to set up all of your projects again to continue to use Eclipse.

Quick Java development tools UI

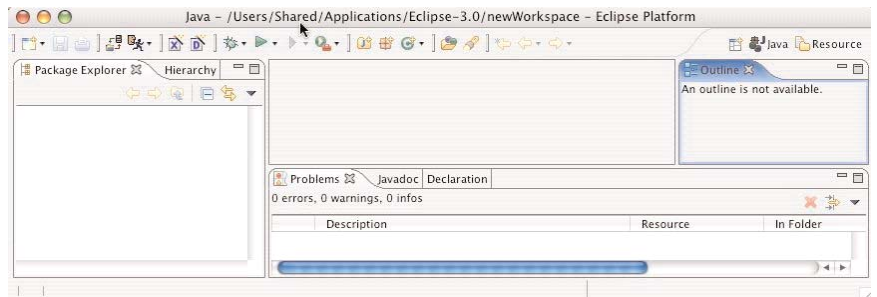
When you first open Eclipse, it will be in what is known as the Welcome View. This view has a group of buttons to help you to get to know Eclipse. Since we won't be discussing the tutorials or samples, etc., we'll skip straight to the Workbench.

Once in the Workbench, you'll be in what is known as the Resource Perspective. Perspectives are groups of views and tools that are arranged to help you do particular tasks. Since the Resource Perspective is not designed for doing

Java development, we won't spend any more time on it. Instead, once you have Eclipse in the Resource Perspective, switch to the Java perspective. Do this by clicking on Window | Open Perspective | Java. You should see a window that looks like **Figure A**.

This window is made up of several regions. The leftmost region, where the Package Explorer views can be seen, is referred to as the navigation region. The next region to the right is the editor region; this is where the Java editor will appear when we edit Java files. The next is the auxiliary view region; this region contains useful views, such as the Outline view shown in Figure A. The region along the bottom is the utility region and contains views that allow us to see interesting information about our projects.

Figure A



Creating and running a Java project

To create a new project in Eclipse, you need to invoke the New Project wizard from the Package Explorer or use the New Project button. We'll start with the context menu invocation in the Package Explorer. Within the Package Explorer, invoke the context menu (right-click in Windows; Control-click on the Mac), and then select New | Project. The first page of the new project wizard appears on the screen, as shown in **Figure B**.

If it's not already selected, choose Java Project, as shown in Figure B, then choose the Next > button. The second page of the wizard will appear. Enter the name of the project (*HelloWorld* in this simple case), as shown in **Figure C**.

In this page, you specify the name of the project and its location. The Create Project In Workspace checkbox is on by default because Eclipse expects that your typical development will reside in the workspace directory. For existing development where you will be creating a project with an existing code base, you will want to uncheck this box and select the directory that the existing code base is in. For now, leave the checkbox checked. Notice that the location is the path to the Eclipse workspace plus the project name. If you ever need to view your project using a program other than Eclipse, you will find it in this directory.

You can also specify the layout of the project on this page. The layout refers to the location of the source code and the generated class files (the compiled Java code). The default location is the root directory of the project. You can, however, edit the defaults via the Configure Defaults button. If you do edit the defaults, the next time you create a new Java project, the Create Separate Source And Output Folders radio button will be selected. It is typical for big projects to have at least one source code directory, and often there are several. For simple experimenting with Eclipse, you can leave the default of keeping the source in the root directory of the package. When you get serious about using Eclipse for real development, though, it is recommended that you create a directory to put your source code into.

Figure B

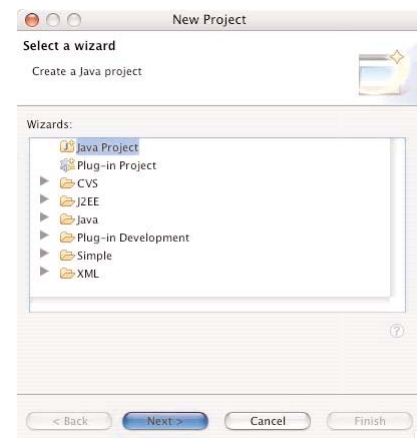
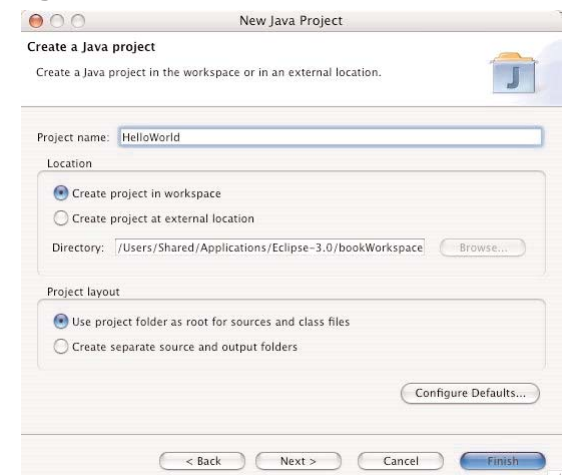


Figure C



For many simple projects, it will be fine to finish here and go directly to the code. In many cases, however, you will want to move on to the next page to set up the *classpath* for the project, as well as other options. Click the Finish button on this wizard, and your first project will show up in the Package Explorer.

Hello World

Because we never want to leave code in the default package in a real Java project, the first step is to make a new package into which to put the code. The example package name is *com.sb.eclipselive*, but you may choose whatever name you like for your package (subject to the naming conventions of Java). The New Package dialog box allows you to specify the name. In the project explorer, use the context menu item New | Package to bring up the dialog box and create the package. Once the new package is created, Eclipse should look like **Figure D**.

Now that we have a project and a package in which to place our source code, it's time to start creating some Java code. Our example HelloWorld project is to print out "Hello World" using a total of three classes: the text based UI, the Hello provider, and the World provider. We will go through the creation of the code one class at a time.

First we create the *HelloWorld* class, which is the main class in our application. It is responsible for gathering the *Hello* and *World* from their providers and putting them together into a string to print out. Eclipse has a couple of ways to build new classes. First, you can choose New | File from the context menu in the package explorer and then specify a file with the *.java* extension. Although this way will work, it does not take full advantage of the JDT. The second way is to use the new class wizard invoked from the context menu New | Class or by clicking the New Class button (the green circle with a C in it—see **Figure E**).

We want to create a new class called *HelloWorld* with a main method that will kick off our program. Check the options and type *HelloWorld* as shown in Figure E. The next step is to modify the code to do what we want it to do. Before it will really work, however, we need to have the *HelloProvider* and the *WorldProvider*. Invoke the wizard two more times to create these two classes, but uncheck the main method check box. You should now have three files/classes: *HelloWorld.java*, *HelloProvider.java*, and *WorldProvider.java*.

Figure D

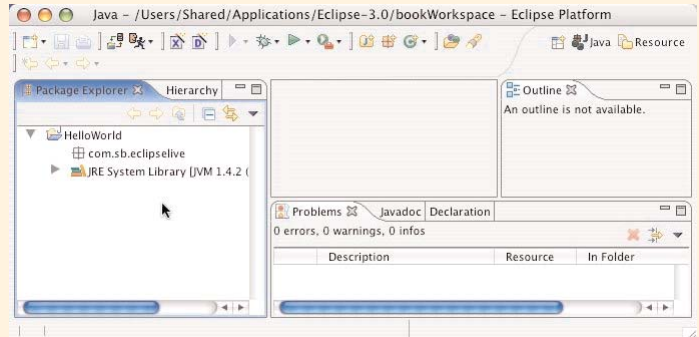


Figure E



Class	Add this code to the class
HelloProvider	<pre>public String getHello() { return "Hello"; }</pre>
WorldProvider	<pre>public String getWorld() { return "World"; }</pre>
HelloWorld	<pre>public static void main(String[] args) { HelloProvider hp = new HelloProvider(); WorldProvider wp = new WorldProvider(); }</pre>

```
String hello = hp.getHello();
String world = wp.getWorld();
System.out.println(hello + " " + world);
}
```

Code completion

To try out the code completion feature of the JDT, delete the last half of the *getHello()* call. With your cursor just past the *get*, hit the auto complete key sequence. (On the Mac, it's [Command][Space]; in Windows it's [Ctrl][Space]). Notice a pop-up selection box that looks like the one in **Figure F**.

You may select an option by using either your mouse or the arrow keys. This feature works with almost all Java elements. For example, if you wanted to add a List to the main method, you could declare it with *java.util.L* and then hit the Content Assist feature. It would fill in the *ist* as well as add an import.

You are finally ready to run your first Java program in Eclipse. To run applications in Eclipse, you must have a Run Configuration that tells Eclipse how to invoke your application. Most of the time, however, it's easy to set up one of these with Eclipse. **Figure G** shows the Run Configuration menu activated with the Run As | Java Application item selected.

Once you click this menu item, you should see the console appear, and your message "Hello World" should be displayed. Think of the Run Configuration menu as a shortcut to tell Eclipse how to run your application. Notice that we did not have to go through a compile step. Eclipse is sophisticated enough to know that your project should be compiled before it's run, so it builds all the code before the invocation actually happens.

You now know how to create a new project, create some classes, and run the program. Although we have not exercised any great depth of functionality, this is the basic life cycle of any project in Eclipse.

Figure F

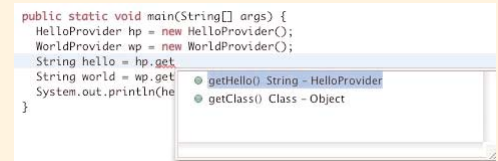
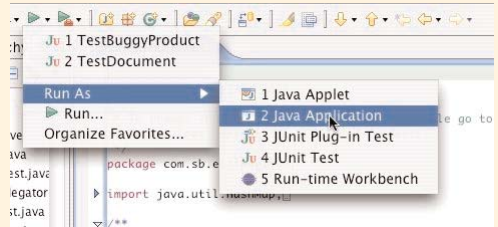


Figure G



Strengths

Great tool set	The Java Development Tools (JDT) are some of the best in the industry.
Huge developer community	There are literally hundreds of plug-ins available for use in Eclipse.
Commercial model	From the foundation of Eclipse there has been a focus on commercial-grade tools, so the participants have a vested financial interest in making dates and allowing employees time at work to spend on the tool.

Weaknesses

No one to call for support	Users of Eclipse are dependent on news groups to get help.
Integration can sometimes be a hassle	Integrating various plug-ins can some times be more complex that it should be.
Lack of documentation	The help that comes with Eclipse is quite good but doesn't go into great depth about many of the features, leaving you to experiment to figure out how the feature works.
Plug-ins are of vastly different quality	It's left to you to download and experiment with a plug-in to see if it is ready for production-level use.

Online resources

[Eclipse.org Main Page](#)

[Eclipse Plugin Central](#)

[EclipsePlugins](#)

Other resources

[Eclipse 3 Live](#)

Bill Dudney's Eclipse blog.

[Java Developers Guide to Eclipse](#)

By Sherry Shavor, Jim D'Anjou, Scott Fairbrother, Dan Kehn, John Kellerman, and Pat McCarthy. Addison-Wesley, 2003, Book and CD-ROM. ISBN: 0321159640.

This book was written by some of the people working for IBM on the Eclipse project. While it has a lot of great information, the book is based on Eclipse 2.0, so some of it is out of date.

[Eclipse: Step by Step](#)

By Joe Pluta. MC Press, 2003, Book and CD-ROM. ISBN: 1583470441.

Written for beginners with no knowledge of Java.

Additional articles

[IBM to bring Eclipse tools to desktop applications](#)

[Is Java cooling off?](#)

[Winners of Builder's 2nd Annual Readers' Choice Awards](#)

[WebSphere Studio Site Developer for Windows](#)

[Put Eclipse Features to Work for You](#)

[Debugging with the Eclipse Platform](#)

About Builder.com

Thank you for downloading this Builder.com tool; we hope it helps you make better technology decisions. If you need help with the site or just want to add your comments, let [us know](#).

Builder.com provides actionable information, tools, trialware, and services to help Web and software developers get their jobs done. Builder.com serves the needs of developers on all major development platforms, including .NET and J2EE, and supplies the knowledge and tools every developer needs to build better and more robust applications.

Free how-to articles: Covering everything from open source software to the latest .NET technologies, Builder articles give developers the breadth and depth of coverage they need to build better applications.

Free e-newsletters: Keep up to date on any aspect of the developer industry—from Web services to Visual Basic—with Builder's e-newsletters, delivered right to your e-mail inbox.

Free trialware: We've collected all the latest trialware and free downloads—covering everything from application servers to HTML editors—to make your job easier.

Discussion Center: Open a discussion thread on any article or column, or jump into preselected topics, such as Java, HTML, and career management. The fully searchable Discussion Center brings you the hottest discussions and threads.

Your free Builder membership opens the door to a wealth of information. Our online developer community provides real-world solutions, the latest how-to articles, and discussions affecting developers everywhere. Get access to full-text books, exclusive downloads, and posting privileges to our discussion boards, all free!