Get started with the Eclipse Platform

Use Eclipse plug-ins to edit, compile, debug, and act as a foundation for your applications

Chris Aniszczyk
David Gallardo

July 17, 2007

Find out about the Eclipse Platform, including its origin and architecture. Starting with a brief discussion about the open source nature of Eclipse and its support for multiple programming languages, we demonstrate the Java™ development environment with a simple programming example. We also survey some of the software development tools available as plug-in extensions.

This follow-up to David Gallardo's "Getting started with the Eclipse Platform" offers new information relevant for Eclipse V3.3. Read "Eclipse Ganymede at a glance" for information relevant for V3.4 and "An Eclipse Galileo flyby" for information relevant for V3.5.

What is Eclipse?

Develop skills on this topic

This content is part of a progressive knowledge path for advancing your skills. See Open source development with Eclipse: Master the basics

Eclipse is a Java-based, extensible open source development platform. By itself, it is simply a framework and a set of services for building applications from plug-in components. Fortunately, Eclipse comes with a standard set of plug-ins, including the well-known Java Development Tools (JDT).

While most users are quite happy to use Eclipse as a Java integrated development environment (IDE), its ambitions do not stop there. Eclipse also includes the Plug-in Development Environment (PDE), which is mainly of interest to those who want to extend Eclipse, since it allows you to build tools that integrate seamlessly with the Eclipse environment. Because everything in Eclipse is a plug-in, all tool developers have a level playing field for offering extensions to Eclipse and providing a consistent, unified IDE for users.

This parity and consistency isn't limited to Java development tools. Although Eclipse is written in the Java programming language, its use isn't limited to the Java language. For example, plug-ins are available or planned that include support for programming languages like C/C++ and COBOL.

The Eclipse framework can also be used as the basis for other types of applications unrelated to software development, such as content management systems.

A great example of an Eclipse-based application is the IBM® Rational® Software Architect (see Related topics), which forms the basis of IBM's family of Java development tools.

Eclipse is open source

Open source software is software released with a license intended to ensure that certain rights are granted to users. The most obvious right, of course, is that the source code must be made available so users are free to modify and redistribute the software. This protection of users' rights is accomplished with a device called a copyleft: The software license claims copyright protection and prohibits distribution unless the user is granted these rights. The copyleft also requires that any redistributed software be covered by the same license. Since this, in effect, stands the purpose of copyright on its head — using the copyright to grant rights to the user, rather than reserve them for the developer of the software — copyleft is often described as "all rights reversed."

Much of the fear, uncertainty, and doubt that has been spread about regarding open source software involves the so-called viral nature of some copyleft licenses — the idea that if you use open source software as part of a program you develop, you will lose your intellectual property because the license will "infect" the proprietary parts you develop. In other words, the license may require that all software bundled with the open source software, including any newly developed software, must be released under the same license. While this may be true of the most well-known copyleft license, the GNU General Public License (under which Linux®, for example, is released), there are other licenses that provide a different balance between commercial and community concerns.

The Open Software Initiative is a nonprofit organization that defines what open source means explicitly and certifies licenses that meet its criteria. Eclipse is licensed under the OSI-approved Eclipse Public License (EPL) V1.0, which is intended to facilitate the commercial adoption of Eclipse while being fair to the open source authors.

Those who create plug-ins for Eclipse or who use Eclipse as the basis for a software development application are required to release any Eclipse code they use or modify under the EPL, but are free to license their own additions in any way they like. Proprietary code bundled with software from Eclipse does not need to be licensed as open source, and the source code does not need to be made available.

Although most will not use Eclipse to develop plug-ins or to create new products based on Eclipse, the open source nature of Eclipse is important beyond the mere fact that it makes Eclipse available for no cost (and despite the fact that a commercial-friendly license means that plug-ins can cost money). Open source encourages innovation and provides incentives for developers, even commercial developers, to contribute code back to the common open source-code base. There are a number of reasons for this, but perhaps the most essential is that the more developers contribute to the project, the more valuable the project becomes for everyone. As the project becomes more

useful, more developers will use it and create a community around it, like those that have formed around Apache and Linux. (See Related topics for more information about licenses.)

What is Eclipse?

Eclipse is an open source community whose projects are focused on building an open development platform comprised of extensible frameworks, tools, and runtimes for building, deploying, and managing software across the life cycle. The Eclipse Foundation is a not-for-profit member-supported corporation that hosts the Eclipse projects and helps cultivate an open source community and an ecosystem of complementary products and services.

The Eclipse Project was originally created by IBM in November 2001 and supported by a consortium of software vendors. The Eclipse Foundation was created in January 2004 as an independent not-for-profit organization to act as the steward of the Eclipse community. It was created to allow a vendor-neutral, open, and transparent community to flourish around Eclipse. Today, the Eclipse community consists of individuals and organizations from a cross-section of the software industry.

The Eclipse Foundation manages and directs Eclipse's ongoing development. The foundation provides services to the community, but does not employ the open source developers (called *committers*), who actually work on the Eclipse projects. Eclipse committers are typically employed by organizations or are independent developers that volunteer their time to work on an open source project.

Now that we've looked at some of the theory, history, and politics behind Eclipse, let's take a look at the product itself.

The Eclipse Workbench

The first time you open Eclipse, you see the welcome page, which exists inside the workbench (see Figure 1). As an Eclipse user, you'll be given a few options of going to an overview page, which I recommend (see Figure 2). See what's new, explore some samples, or go through some tutorials.

Figure 1. Eclipse Welcome Page

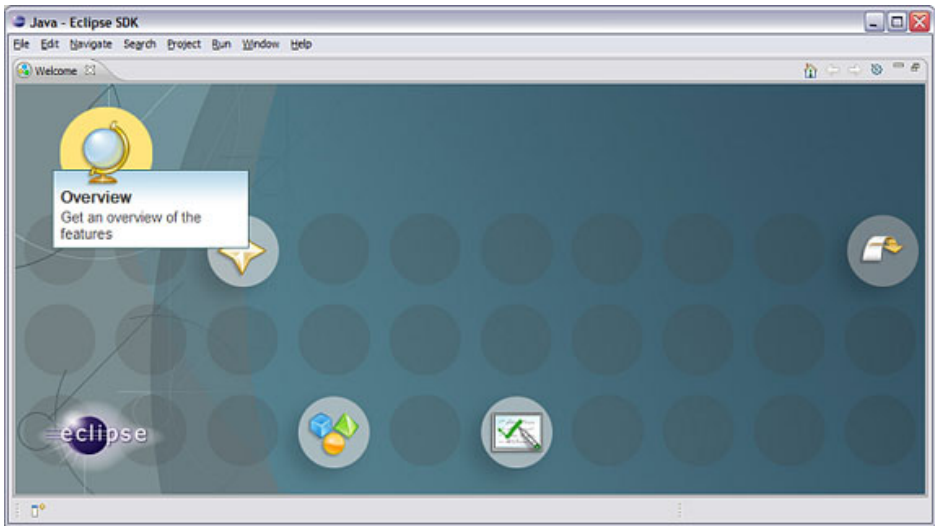
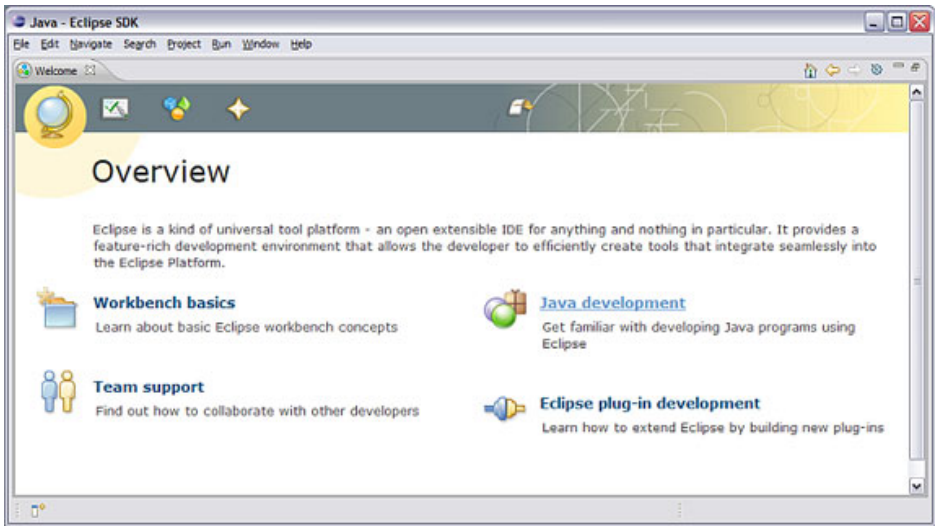


Figure 2. Eclipse Overview Page



The Eclipse workbench consists of several panels known as *views*, such as the navigator or outline views. A collection of these views is called a *perspective*. One of the most common perspectives is the Resource perspective, which is a basic set of views for managing projects and viewing and editing files in a project.

I recommend most novice users start with the Overview page featured in Figure 2 and learn about Eclipse. The workbench basics section contains a lot of good starter information about the various pieces of Eclipse and how they interact. Spend a few minutes reading the material, then let's dive directly into the Java Development Tools (JDT) in Eclipse. There's no better way to learn than doing it hands-on.

To continue this short tour of Eclipse, we'll create a a new Java project. Select **File > New > Java Project** and enter `Hello` when prompted for the project name, then click **Finish**.

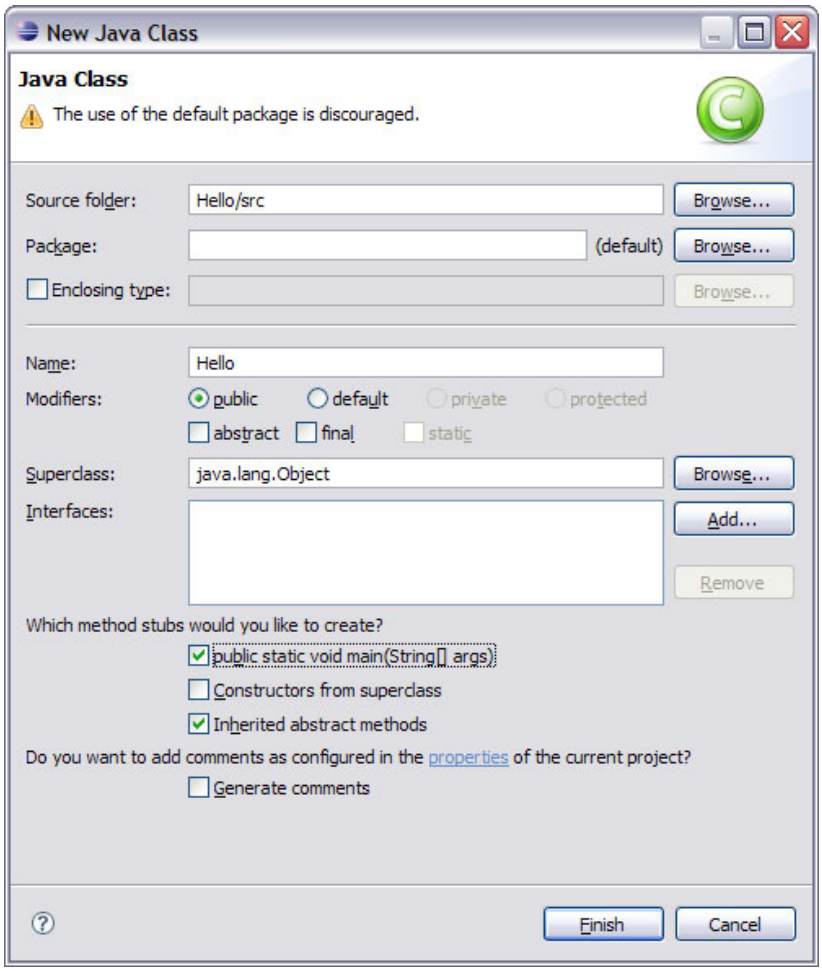
Next, we'll take a look at the Java perspective (if you aren't already there). Depending on how you like to manage your screen, you can change the perspective in the current window by selecting **Window > Open Perspective > Java** or you can open a new window by selecting **Window > New Window** and selecting the new perspective.

The Java perspective, as you might expect, has a set of views that are better suited for Java development. One of these includes, as the top-left view, a hierarchy containing various Java packages, classes, JARs, and miscellaneous files. This view is the called the Package Explorer. Also notice that the main menu has expanded to include two new menu items: Source and Refactor.

The Java Development Tools (JDT)

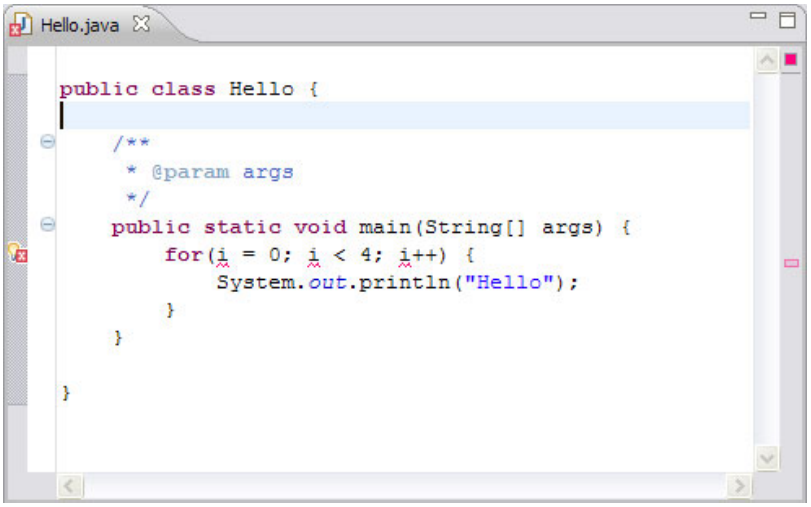
To try out the Java development environment, we'll create and run a Hello World application. Using the Java perspective, right-click on the Hello project's source folder (src) and select **New > Class**, as shown in Figure 3. In the dialog box that appears, type `Hello` as the class name. Under **Which method stubs would you like to create?** check **public static void main(String[] args)**, then **Finish**.

Figure 3. Creating a new class in the Java perspective



This will create a .java file with a Hello class and an empty `main()` method in the editor area, as shown in Figure 4. Add the following code to the method (note that the declaration for *i* has been omitted deliberately).

Figure 4. The Hello class in the Java editor



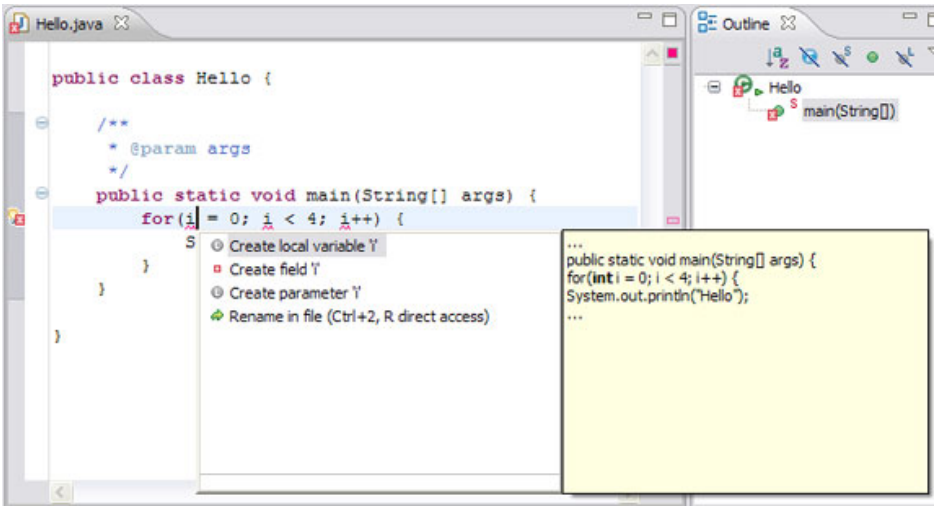
You'll notice some of the Eclipse editor's features as you type, including syntax checking and code completion. Also, when you type an open parenthesis or double quote, Eclipse will provide its partner automatically and place the cursor inside the pair.

In other cases, you can invoke code completion by using **Ctrl+1**. Code completion provides a context-sensitive list of suggestions selectable by keyboard or mouse. The suggestions can be a list of methods specific to a particular object or a code snippet to expand, based on various keywords such as `for` or `while`.

Syntax checking depends on incremental compilation. As you save your code, it is compiled in the background and checked for syntax errors. By default, syntax errors are underlined in red, and a red dot with a white X appears in the left margin. Other errors are indicated with a lightbulb in the editor's left margin; these are problems that the editor might be able to fix for you (a feature called Quick Fix).

The code above has a lightbulb next to the `for` statement because the declaration for `i` has been omitted. Double-clicking on the lightbulb will bring up a list of suggested fixes. In this case, it will offer to create a class field `i`, a local variable `i`, or a method parameter `i`; clicking each of these suggestions will display the code that would be generated. Figure 5 shows the list of suggestions and the code it suggests for a local variable.

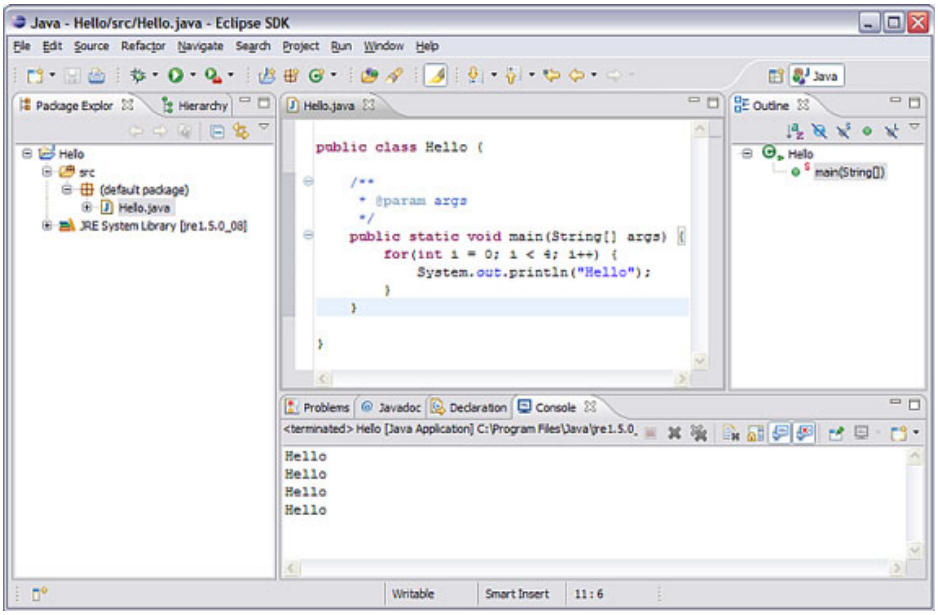
Figure 5. Quick Fix suggestions



Double-clicking on the suggestion inserts the code in the proper location in the code.

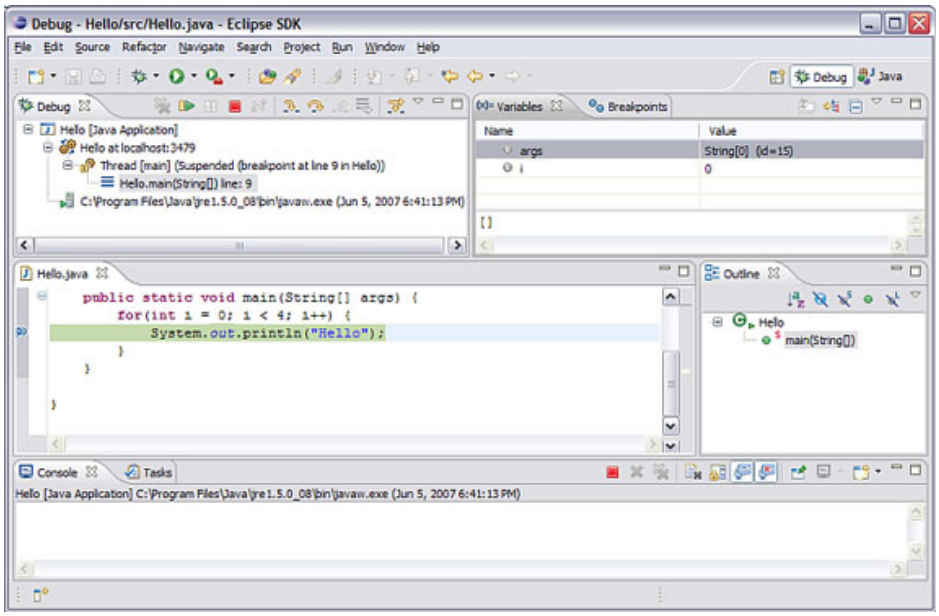
Once the code compiles without error, you can execute the program by selecting **Run** from the Eclipse menu. (Note that there is no separate compilation step because compilation takes place as you save the code. If your code has no syntax errors, it's ready to run.) A Launch Configurations dialog box appears, with appropriate defaults; click **Run** at the bottom right. A new tabbed panel appears in the lower panel (the Console), displaying the program's output, as shown below.

Figure 6. Output from the program



You can also run the program in the Java debugger. First, set a breakpoint in `main()` `System.out.println()` by double-clicking in the gray margin on the left side of the editor view, next to the call to `System.out.println()`. A blue dot will appear. From the **Run** menu, select **Debug**. As described, a Launch Configurations dialog will appear. Select **Run**. The perspective will change to the Debug perspective automatically, with a number of interesting new views, as shown below.

Figure 7. The Debug perspective



Notice the Debug view at the top left of the perspective. This view shows the call stack and has a toolbar in the title bar that allows you to control the execution of the program, including buttons to resume, suspend, or terminate the program, step into the next statement, step over the next statement or return from a method. The panel at the top right contains a number of tabbed views, including Variables, Breakpoints, Expressions, and Display. I've clicked Variables so we can see the current value of `i`.

You can obtain more information about any of the views with the context-sensitive help; click on the title of the view and press **F1**.

Additional plug-ins

In addition to plug-ins like the JDT for editing, compiling, and debugging applications, plug-ins are available that support the complete development process from modeling, build automation, unit testing, performance testing, version control, and configuration management.

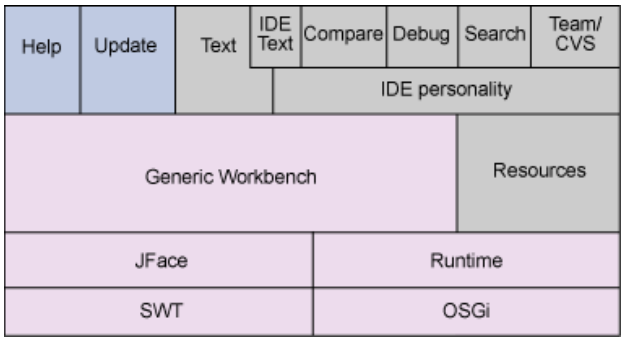
Eclipse comes with a plug-in for working with the open source Concurrent Versions System (CVS) for source control. The Team plug-in connects to a CVS server, allowing the members of a development team to work on a set of source-code files without stepping on each other's changes. Source control from within Eclipse won't be explored here further because it requires setting up a CVS server, but the capability for supporting a development team, not just stand-alone development, is an important and integral feature of Eclipse.

Plug-ins supported and hosted by the Eclipse Foundation are found on the Eclipse Web site. For a more complete list of available plug-ins, go to Eclipse Plug-in Central, which acts as an index for plug-ins.

Eclipse Platform architecture

The Eclipse Platform comes with a powerful set of plug-ins (see Figure 8) that support projects, such as JDT and the PDE.

Figure 8. Simplified Eclipse Platform architecture



The dark blue pieces signify components that are the core part of Eclipse's Rich Client Platform (RCP). The concept of RCP itself is outside the scope of the article, but think of RCP as just a set of plug-ins from Eclipse people can use to develop applications, such as Lotus Notes® 8. The light blue pieces are optional (but recommended) pieces to be included in RCP-based applications. And the gray pieces are completely optional. The platform consists of several components, of which I'll cover several:

Runtime
: The runtime is the code that defines Eclipse's plug-in model, based on the OSGi specification, and notion of extensions and extension points. The runtime also provides additional services like logging and concurrency.

JFace/SWT
: The Standard Widget Toolkit (SWT) is the widget set that gives Eclipse its look and feel. JFace is simply a layer on top of SWT that provides some Model-View-Controller (MVC) classes to make it easier to develop graphical applications.

Workbench
: The workbench gives Eclipse its personality. The concept of views, perspectives, and things like editors are defined at this level.

Help (User Assistance)
: The Eclipse component allows you to provide assistance to your users. This can be done via the help system, which allows users to search for help documentation or through cheatsheets, which can be thought of interactive task lists for end users.

Update
: The update component of Eclipse provides the facilities to allow you to update your applications from version to version.

Team
: The team component consists of a framework to allow vendors to plug in their own version-control systems. An exemplary implementation of a provider is the CVS plug-in provided out of the box in Eclipse.

Conclusion

From an initial contribution of code from IBM, Eclipse has grown into a full-fledged open source ecosystem in which more than 100 companies participate. A portable, extensible open source framework isn't a new idea, but because of its mature, robust, and elegant design, Eclipse brings a whole new dynamic into play.

Welcome to the Eclipse ecosystem. Our goal was to get you started with the Eclipse platform quickly. We accomplished this with some introductory text accompanied with a simple hands-on exercise. Take the knowledge gained from reading this article and become an active member of the Eclipse ecosystem.

Related topics

- Interested in what's happening inside the Eclipse community? Check out PlanetEclipse.
- Information about open source software, including certified open source licenses, such as the Eclipse Public License, is available at OpenSource.org.
- Check out EclipseLive for webinars featuring various Eclipse technologies.
- Learn more about Rational Software Architect.
- Eclipse is available under the Eclipse Public License (EPL) V1.0.
- Eclipse Plug-in Central is an index for plug-ins. You can get an RSS feed of the latest plug-ins.
- The Rich Client Platform (RCP) provides a foundation for building cross-platform applications.
- Check out the "Recommended Eclipse reading list."
- Download Eclipse Platform and other projects from the Eclipse Foundation.
- Browse all the Eclipse content on developerWorks.
- Follow developerWorks on Twitter.
- New to Eclipse? Read the developerWorks article "Get started with the Eclipse Platform" to learn its origin and architecture, and how to extend Eclipse with plug-ins.
- As someone interested in development with Eclipse, you might want to check out a trial of IBM's Rational Application Developer Standard Edition, a commercial development tool built on Eclipse technology.