

```
run() {  
    I  
    resource resource= g  
    if (resource == null)  
        return;  
    Map attributes= getInitialAttributes  
    TaskPropertiesDialog dialog = new Ta  
    dialog.setResource(resource);  
    dialog.setInitialAttributes(attribu  
    dialog.open();  
    static Thing cre  
    return new Thing();  
} void run() {  
abstract class A {  
    public abstract void send(String message)  
    message);  
    attributes);  
    extends A {  
        Map attributes= getIn  
        void send(String message) (  
            TaskPropert
```

BREAK ON

coverstory *By Robert Cravotta, Technical Editor* **THE OPEN-SOURCE ECLIPSE PROJECT IS POISED TO IMPACT THE WORLD OF EMBEDDED-DEVELOPMENT TOOLS.**

THROUGH

IN RECENT YEARS, the open-source software-development process has demonstrated some impressive successes (see **sidebar** “Defining open source”). According to the December 2003 Netcraft Web Server Survey (news.netcraft.com), users employ the Apache HTTP Server project, an open-source project of the Apache Software Foundation (www.apache.org), to run more than 67% of the Websites on the Internet. The GNU/Linux operating system (www.linux.org) is another widely used and supported open-source software project. The Eclipse Platform is an open-source software project that is garnering support for embedded-development tools.

<i>At a glance</i>	50
<i>Defining open source</i>	50
<i>Native versus emulation</i>	52
<i>Extending Eclipse</i>	54
<i>For more information</i>	56

Eclipse is an open-source software-development project focused on building a platform for hosting and developing tightly integrated software-development tools. IBM, with a \$40 million donation, launched Eclipse as an open-source project in November 2001 as a way to more cost-effectively create a platform that supports best-of-breed tool plug-ins. IBM has continued to be the major content contributor to the project and has headed the project organization. The project is currently changing to an independent entity that replaces the IBM leadership with an independent management organization. This transition emphasizes the project’s focus on creating an industrywide platform for tool integration. The name of the project may also change as part of the transition.

The Eclipse and the Sun-driven NetBeans projects are similar open-source projects that compete with each other as Java-development environments. Sun recently formally declined to join the soon-to-be independent entity's board of stewards, citing a lack of common ground between the Eclipse and the NetBeans projects that would preclude an equitable share in mutual development. Both Eclipse and NetBeans are Java-based platform frameworks and language-independent IDEs (integrated-development environments). Being Java-based, they can operate on a range of hardware and operating systems, including Windows, Linux, Solaris, and Mac

AT A GLANCE

- ▶ Eclipse and NetBeans are similar but competing open-source projects.
- ▶ Companies are successfully using the Eclipse platform for delivering embedded development tools.
- ▶ The CPL (Common Public License) balances the needs of open source with the need to enable proprietary components.
- ▶ Consider evaluating the Eclipse platform if the IDE is not the core benefit you want to add to your tools.

OSX. Both IDEs are free downloads, require a JDK (Java development kit) 1.3 or higher compatible virtual machine, and immediately support Java development. Additional language or development-tool support, such as for C/C++ development, requires you to download additional plug-ins or modules.

Both Eclipse and NetBeans allow you to create your own tools that integrate with and extend the IDE to support any software-development function. Tool extensions are called modules for NetBeans and plug-ins for Eclipse. The plug-in/module interface differs between the two platforms, so the plug-ins and modules for each are incompatible, and the

DEFINING OPEN SOURCE

There are many open-source licenses—including the CPL (Common Public License), which Eclipse uses, and the SPL (Sun Public License), which NetBeans uses—that comply with the OSI (Open Source Initiative) open-source definition. The OSI definition addresses the distribution criteria for gaining access to a program's source code and making it available to others. The licensing criteria are summarized below. The official open-source definition is available from the OSI at www.opensource.org.

To ensure that there is a mechanism for the free redistribution of software, a license can neither restrict you from selling or giving away open software as a component of an aggregate software distribution that includes software from different sources, nor require you to collect a royalty or other fee for any sale.

To promote and facilitate community-based evolution of the software, any program must include source code. If you do not distribute the source code with the program, you must offer a well-publicized means, such as an Internet download, from which others can obtain the source for no more than a reasonable reproduction cost. The

license must allow distribution of source code as well as compiled forms. You must provide the source code so that a programmer can modify the program. You may not deliberately obfuscate the source code or distribute only intermediate forms of the source, such as the output of a preprocessor or a translator.

To encourage rapid evolution and experimentation of the software by the community, the license must allow others to modify the software and derive works from it. It must also allow anyone to distribute the results under the same terms as the license of the original software.

To allow others to more clearly identify who is responsible for the software, as well as to enable authors and maintainers to protect the integrity of their work, the license may restrict the distribution of the source code in modified form. However, it may do so only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at building time. This provision allows the distribution of unofficial changes and enables you to readily distinguish them from the base source. The license must explicitly permit distribution of software

you build from modified source code, and it may require derived works to carry a different name or version number from the original software.

To facilitate diverse contribution sources with the expectation that this diversity will lead to the best benefits during the open-source-development process, the license may not discriminate against any person or group of persons. A license may warn licensees of applicable legal restrictions and remind them that they are obliged to obey the law, such as the export restrictions for some types of software, but it may not incorporate the restrictions itself.

To prohibit license traps that prevent the commercial use of open-source software and to encourage commercial users to use and participate in the open-source-development process, the license may not restrict anyone from using the program in a specific field of endeavor. To prevent a license from locking up software by indirect means, such as through requiring a nondisclosure agreement, the rights attached to a program must apply to anyone to whom the program is redistributed without the need for that party to execute an additional license.

To address another class of license trap, the rights attached to the program must not depend on the program's being part of a particular software distribution. If you extract the program from that distribution and use or distribute it within the terms of the program's license, all parties to whom you redistribute the program should have the same rights that the original software distribution grants.

To allow distributors of open software to make their own licensing choices about their own software, the license may not restrict other software that is distributed along with the licensed software. The license may not require that other programs distributed on the same medium be open-source software.

Finally, to address any license that requires an explicit gesture of assent to establish a contract between licensor and licensee, no provision of the license may be predicated on any individual technology or style of interface. This stipulation allows for the redistribution of the software over channels that do not support "click-wrapping," as well as covered code that may run in an environment that cannot support pop-up dialogues.

platforms cannot share them. The cross-target graphical tool kits that each project uses are another source of significant divergence (see **sidebar** “Native versus emulation”).

NetBeans uses the Swing/AWT (Abstract Windowing Toolkit) graphical tool kits to provide a “write-once, run-anywhere” capability. The Eclipse project uses the JFace/SWT (Standard Widget Toolkit) graphical tool kits, which can provide better native performance as well as the look and feel at the expense of some target portability. Another fundamental difference is that Swing uses the Java garbage collector, and SWT relies on the programs to free operating-system resources. This difference allows SWT to expose operating-system functions and avoid timing issues—especially between versions of the JVM (Java Virtual Machine)—from the time an object becomes garbage to when the system performs finalization to free the resources.

EMBEDDED DEVELOPMENT

Eclipse and NetBeans share a strong enterprise-software-development bias, but integrated tools for embedded development that use the Eclipse framework are starting to appear. The member-company representation in the Eclipse organization encompasses the enterprise, Linux, and embedded-development segments for tool development. Each of these segments has a different audience with different needs, and the platform workbench includes commodity functions that apply to each development

segment. Tensilica and WindRiver are recent adopters of the Eclipse platform from the embedded-system industry.

Tensilica’s Xtensa Xplorer IDE uses the Eclipse platform for processor-configuration and software-development tools. Even though Wind River has the Tornado IDE, company officials feel that using the Eclipse framework allows it to focus its development resources more on tool functions and less on commodity-infrastructure functions. Eclipse’s tighter integration with the native platform, which resulted in near-native performance, and its more native look and feel were considerations in the companies’ decisions to adopt Eclipse.

The Eclipse framework is distributed under the CPL (Common Public License), which encourages collaborative open-source development of the code base but is flexible enough to let you use and integrate the code with software licensed under other licenses, including commercial licenses. The OSI (Open Source Initiative) approved the CPI, and it is available at www.opensource.org. If you modify a program licensed under the CPL, such as the Eclipse code base, and you distribute the object code, you are obligated to make the modified source code available to others. You need not make your modifications available to others if you use the modified program only internally and do not publicly distribute the changes. You also need not make the source code publicly available if you write and distribute plug-in modules that interface with but do not change the CPL code.

The provision that plug-in modules need not be open-source if they interface with but do not change the base code is important if you want to distribute value-added functions under a commercial license. This licensing model allows you to freely use the Eclipse framework and develop value-added functions as plug-ins instead of using your tool-development resources to build an environment around your functions without giving up proprietary IP (intellectual-property) rights.

Developing and maintaining software tools from scratch can be costly. Open-source projects, such as Eclipse, may help you reduce your building and maintenance costs for the commodity portions of your tools. The plug-in architecture of the Eclipse platform couples with the terms of the CPL to allow you to balance between using and distributing the commodity and proprietary portions of your code and benefit from and contribute to the commodity portion of the software at lower cost than if you did not use the platform (see **sidebar** “Extending Eclipse”). However, not all software is created for distributed commercial sale; in-house tools and embedded software are two examples. As open-source commodity software continues to mature, it may make sense for you to periodically evaluate whether and how platforms such as Eclipse might help you develop and maintain your in-house software tools at lower cost.

The embedded-software-development community is fragmented and diverse.

NATIVE VERSUS EMULATION

Debate is ongoing and fervent about whether graphical tool kits should stress flexibility, generality, and portability versus integration with the native windowing system. Swing/AWT (Abstract Windowing Toolkit), which NetBeans uses, and JFace/SWT (Standard Widget Toolkit), which Eclipse uses, differ in that Swing stresses abstraction for portability and consistency, and SWT stresses exposing the native operating-system support for more natively like performance and look and feel.

AWT supports native widgets with encapsulated C code for each windowing operating system. To maintain universality, the AWT tool kit includes only widgets that all targets natively support. The AWT tool kit does not include widgets that are not present in each target system; for example, neither Motif nor AWT includes a tree widget. The Java Swing tool kit provides a higher level widget abstraction that directly implements widgets in Java. Because it does not depend on the operating system

to provide widgets, it enables target portability and a consistent look and feel across targets. The Swing tool kit provides look-and-feel emulation layers to more closely resemble the underlying native window system, but it is subject to possible sluggish response and a different look and feel from native applications.

SWT encapsulates the native operating system using a one-to-one mapping from Java to C that allows the JFace layer more flexibility to exploit native widgets. If

there is native-widget support, the tool kit uses it; otherwise, the tool kit emulates the widget. In this way, SWT can expose and tightly integrate with a native system-specific API (application-programming interface) that provides unique features unavailable on other systems. But it can cost you in portability and the need to build platform-specific versions of the application code. Portability may not be an issue for you if you need to host your application only on one target platform.

The embedded-development environment is teeming with processor architectures and development tools to address dissimilar application-specific requirements for constrained resources, such as memory usage, processor performance, and power consumption. Embedded software often tightly couples

with the underlying hardware to meet the performance and cost constraints of real-time embedded-system applications. The fragmented nature of the embedded-software-development environment complicates the wide applicability that benefits many open-source software efforts, because most of the embedded-sys-

tem architectures differ significantly and are exclusive to each other.

However, the embedded-development market's fragmentation strengthens Eclipse's core value-proposition, because the open-source base code covers the lower commodity-value portion of IDE construction and supports proprietary-

EXTENDING ECLIPSE

Every component and function in Eclipse, with the exception of a small runtime kernel, is a plug-in (**Figure A**). Your custom plug-in integrates into the Eclipse environment in the same way as core plug-ins. The PDE (Plug-in Development Environment) and JDT (Java Development Tooling) are core extensions of Eclipse. The PDE includes wizards to

tion allows Eclipse, using an operating-system-independent registry, to invoke a registered tool based on the resource type. Invocation integration allows you to integrate legacy tools. The invoked tool does not run in the workbench, it does not share the same JVM (Java Virtual Machine), and it cannot integrate with other Eclipse views or

its format, understanding it, and maintaining its integrity.

API (application-programming-interface) integration provides data encapsulation to access data through plug-in-specific APIs. A non-Java plug-in provides the API implementations of the Java interface through JNI (Java Native Interface). To make the API available to other plug-ins, you need to describe the API in the plug-in manifest file.

User-interface integration allows plug-ins to operate with other plug-ins as if they were single applications. With user-interface integration, you can break a tool into multiple plug-ins, which allows you release plug-in components rather than a single monolithic unit. Plug-ins can register events that other plug-ins generate to accordingly adjust their user interface or even to extend another plug-in, such as a tool menu. By using a common view with your plug-ins, you can minimize the number of opened windows and provide a more consistent look and feel.

The plug-in interconnection model supports any number of named extension points and any number of extensions to one or more extension points in other plug-ins. Plug-ins can extend other plug-ins. For example, any plug-in can contribute its own user preferences to the workbench because, as a plug-in, the workbench plug-in includes an extension point for user preferences. Any plug-in may define new extension points and pro-

vide a new API for other plug-ins to use.

To support testing and debugging of your plug-in, the PDE provides a self-hosted development environment that lets you run a plug-in without installing it in a separate instance of the workbench. To install your plug-in after you have tested it, you need to copy the program and manifest files in a subdirectory in the Eclipse "plugins" directory. You must stop and restart Eclipse so it can recognize and load the new plug-in.

Two other types of Eclipse plug-ins bear mention. A plug-in fragment forms a part of a target plug-in. A fragment differs from a full plug-in in that it does not have a plug-in class, and the target plug-in manages its life cycle. You can use a fragment to localize a target plug-in for different languages, provide platform-specific functions, or add features to a plug-in without a full new release.

A plug-in feature is the other type of plug-in. It requires no coding. A feature in the Eclipse architecture is the packaging of a group of related plug-ins into an integral product. For example, the JDT is a feature comprising plug-ins such as a Java editor, debugger, and console. The primary feature sets the look and feel of the platform, including characteristics such as a splash screen, that gives the workbench its identity. By packaging and making a set of plug-ins into the primary feature, you can rebrand the workbench to create a new product.

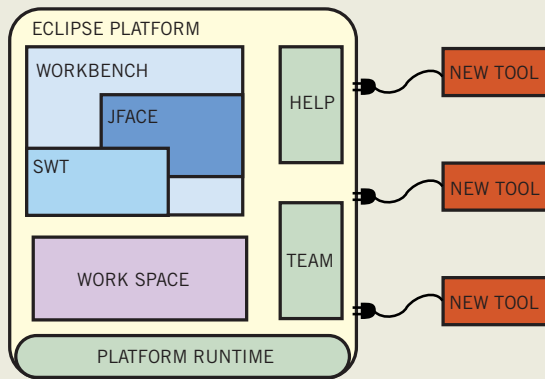


Figure A

Except for the runtime kernel, the Eclipse platform implements every component and function, including the major components and APIs using the plug-in architecture.

help you create Java plug-ins, including the manifest file, which contains descriptive information that Eclipse uses to integrate the plug-in into the framework.

Eclipse supports five levels of plug-in integration that determine how a set of plug-ins must behave and what they can accomplish. The simplest integration level is when you need no integration because the plug-in is completely self-contained or has limited relationships with other resources.

Invocation, or launch, integra-

editors. However, it can use the Eclipse project-model and resource-management facilities.

Integrated data sharing allows plug-ins to share manipulated data. For this type of integration, a plug-in must implement an access method to define how it accesses another plug-in's data, an interchange protocol to specify the form in which it accesses the data, and data-transformation facilities to make the data more useful to requesting plug-ins. Each plug-in that uses the data is responsible for decoding

tool extensions. Usually, software-development tools from processor vendors are strategic components to selling their silicon and are generally not subject to direct competition from similar tools from other processor providers. A possible indirect benefit of using Eclipse-based development-tool suites is that your in-house tools could exist in the same environment and support better integration between in-house tools and other tools, proprietary or otherwise, that have also adopted the Eclipse platform. On the other hand, pure-software-tool providers could see Eclipse-based tools as sources of complication because these providers base a significant portion of their tools' value on creating a consistent and fully integrated environment that covers a range of processor devices and architectures.

The GCC (GNU Compiler Collection), available at www.gnu.org, provides an open-source compiler front end for several programming languages, including C and C++. To support a specific microcontroller, someone must port the GCC to support it, and ports for embedded processors, such as ARM and MIPS architectures, exist. The GCC can probably support code compilation for any microcontroller or DSP. However, application-specific resources and optimizations, such as special-purpose registers, multiple memory spaces, and modal instructions that some microcontrollers and DSPs use, do not play to the strengths of the GCC engine and may make the porting effort impractical. The compiler output-code quality and performance are tool differentiators. Proprietary embedded-compiler tools are quick to highlight material performance margins when you compare them with

GCC output, and those margins can critically affect the hardware costs for high-volume, cost-sensitive applications.

The Eclipse IDE completely avoids the target-code-efficiency issue unless you are developing for a Java target. Eclipse's current proposition is that, as a tool developer, you do not have to build the commodity components of an IDE, such as the workbench, file navigator, wizards, text editor, component-version management, and publishing services. The CDT (C/C++ Development Tools) project provides a set of plug-ins that implement a C/C++ IDE. Each component of the workbench is extensible and customizable via the plug-in interconnection model, allowing you to focus more of your development resources on the value-added tool function. If your suite of tool functions is loosely coupled, you can with little effort probably focus most of your resources on plug-in development and maintain compatibility with future versions of the base code. If your suite of tool functions is tightly coupled, you may decide it is worth changing the base code and contribute the changes back to the community.

COMMODITIZING

Under the terms of the CPL, if you distribute software or tools that make changes to the Eclipse base code, you must make the source code for those changes available to the community. Why would you be willing to invest resources to modify code that you must give away? The answer to this question is central to the short- and long-term viability of the open-source community. The roots of the formal open-source community arguably derive from the idea that a community of individuals can provide alter-

natives to and compete with proprietary software from big businesses.

Many theories try to explain why programmers choose to develop open-source software despite the fact that this type of development generally precludes direct monetary rewards. For some, open-source development is a way to build a reputation; for others, it is a fun hobby; for yet others, it is a way to learn how to be a better programmer. Some do it because it aligns with their political or moral positions. Others see it as a way to give back to the community, to protest proprietary software, to make software affordable, or to take and keep control over the software they use.

These reasons may apply to individuals, but why would a business invest resources to build and test code that is freely available to its competitors? Joel Spolsky, owner of Fog Creek Software, applies microeconomic concepts to theorize why companies would spend their resources on open-source software (**Reference 1**). Every product in the marketplace has substitutes and complements. A substitute is an acceptable alternative product you might use if another product costs you too much. You could view open-source software as a substitute to proprietary software.

A complement is a product that you usually buy together with another product—for example, computer hardware and an operating system. Embedded processors and software-development tools are also complements. An important relationship between complements is that demand for a product increases as the prices of its complements decrease. Processor vendors try to strategically commoditize the software-development tools available for their processors to enhance the demand for their processor offerings. In contrast, pure-software-development vendors try to commoditize the processor support, by supporting a range of processor devices and architectures to enhance the applicability and demand for their software-development environment and tools across a wider audience.

Commoditizing the complements to your company's core competencies makes good business sense. Communitywide use of Eclipse can commoditize the base functions of development-software tools and IDEs. Hardware is not the only complement to development tools; support

FOR MORE INFORMATION...

For more information on products such as those discussed in this article, contact any of the following manufacturers directly, and please let them know you read about their products in *EDN*.

Eclipse
www.eclipse.org

MontaVista
1-408-328-9200
www.mvista.com

Scapa Technologies
1-212-792-4032
www.scapatech.com

Wasabi Systems
1-757-248-9601
www.wasabisystems.com

Genuitec
1-972-491-3704
www.genuitec.com

NetBeans
www.netbeans.org

Sun Microsystems
1-800-555-9786
www.sun.com

Wind River
1-510-748-4100
www.windriver.com

IBM
1-800-426-4968
www.ibm.com

QNX
1-800-676-0566
www.qnx.com

Tensilica
1-408-986-8000
www.tensilica.com

and consulting services can also benefit from commoditizing the tool environment. The Eclipse platform and the CPL do not preclude value-added plug-in tools that commoditize consulting services.

Keys to deciding whether to use an open-source platform are the overall time and cost differences between building and maintaining your own platform and using the open-source platform. Considerations include modifications you might need to better accommodate your needs, such as easing the complexity to integrate and more naturally present tightly coupled tools to your user. If a tighter integration mechanism will add strategic value to your product, an open-source platform may be a bad business choice. If the Eclipse platform provides you with sufficient IDE function, using it may save you significant development resources that you can focus on more valuable tools and functions.

INTEROPERABILITY AND SUPPORT

Development tools provide value by lowering development risks. They can reduce development time, overall cost, and complexity by enabling you to work at higher abstraction levels. Although the Eclipse platform is free, the tools on top of it are not necessarily so. And, although tool interoperability is not the focus of the platform, it is another potential benefit of the Eclipse platform whereby plug-in tools built by independent developers can interact with each other in an integrated way without prior effort. Examples of this type interoperability are code-development tools from a provider and library/version-management tools. Their coincidental interoperability is due to the fact that little to no coupling exists between the tools; they perform functions that are exclusive to each other. Any interoperability between tightly coupled tools is the result of deliberate planning and cooperation using and extending the integration levels built into the Eclipse platform.

Most Eclipse-based tools are available as complete installation packages that include the Eclipse binary. In this way, the tool provider provides you with a single

source for getting the tools running. It also enables the tool provider to ensure version compatibility between all of the dependent components and to bundle any complementary plug-ins from a single installation. Eclipse-based tools can also take advantage of the update-manager plug-ins to manage updates. Tool vendors may also make their Eclipse-based tools available as just plug-in files; this approach requires users to install Eclipse on their systems and places responsibility for version synchronization in their hands.

Each company handles Eclipsed-based technical support differently. Just because the platform code is open source does not mean it is in the tool providers' best interest for you to troubleshoot problems without their support. In general, technical support for the tools is a key value, and each company may act as a single point of contact for you with regard to its tools within the environment. If you are experiencing trouble within the base platform or between two tools, some companies may still act as a single point of contact to determine the cause of and a resolution to the problem.

As an embedded-system designer and user of development tools, you may not care whether your tool provider uses the Eclipse platform. If you develop in-house tools, you may value the opportunity for tool integration that the Eclipse platform provides. You may also value the independent tool-interoperability opportunities; be sure to recognize that you may

already be using a proprietary tool environment that supports similar integrated tool interoperability. The Eclipse platform has a strong enterprise bias in its architecture, but it may become more appropriate for supporting embedded-development tools, especially as companies from the embedded-tool environment on the board of stewards

influence the code base to better accommodate the needs of embedded development. □

You can reach Technical Editor Robert Cravotta at 1-661-296-5096, fax 1-661-296-1087, e-mail rcravotta@edn.com.



REFERENCE

1. Spolsky, Joel "Strategy Letter V" www.joelonsoftware.com/articles/StrategyLetterV.html, June 12, 2002.