

The Eclipse Rich Client Platform

Technical Information Paper – v1.0

20 January 2005

This documentation is proprietary to IBM and may be distributed UNMODIFIED to third parties without the prior written permission of IBM. Any unauthorized use or modification of this document is strictly prohibited. IBM owns all right, title, and interest in and to this documentation. Copyright © 2004, IBM All rights reserved.



Authors:

Name	Contact eMail
Mark Rogalski	Rogalski@us.ibm.com

Attention: This document was printed from an online system and may be used only for reference purposes. The online document must be considered as the current valid version. Please ensure that this printed document is current, and to preserve its integrity, do not remove any pages.

Introduction

The Eclipse 3.0 SDK is an amazing tooling platform. Basically, it provides an open framework that anyone can use to enhance functionality of the Integrated Development Environment (IDE). Better yet, these features (or plug-ins) are easily versioned and dynamically installed or updated without restarting the IDE. Software developers using Eclipse have often wished for a similar model for their desktop applications. With previous versions of Eclipse, this was possible but difficult, especially when you wanted to heavily customize the menus, layouts, and other user interface elements. That was because the IDE nature of Eclipse was hard-wired into it. Eclipse Version 3 introduces the Rich Client Platform (RCP), which is basically a refactoring of the fundamental parts of the UI, enabling RCP to be used as a general-purpose application platform. RCP internals are the same OSGi run time and GUI toolkit provided by the Eclipse IDE, but now these are easily used by applications to provide robust, customizable, and portable Java™ applications.

Because of the Eclipse open source license, you can use the technologies that went into Eclipse to create your own commercial-quality programs. The GUI toolkits used by Eclipse RCP are the same used by the Eclipse IDE and enable applications with optimal performance that have a native look and feel on any platform that they run on.

Reasons for RCP

Developers often componentize large applications for a variety of reasons, including encapsulating code and distributing development, which give the user the ability to install smaller sets of function. The Eclipse development platform provides all these capabilities. The Eclipse IDE lets multiple, distributed tool developers write rich GUI features that easily plug into the IDE workbench. Eclipse developers saw that extending this model to applications could be quite beneficial and so started the Eclipse Equinox project to investigate the possibilities. The result was Eclipse RCP. Now application developers can spend their time more effectively on their business logic, and not have to spend as much time on core infrastructure.

OSGi framework

One of the primary benefits of the Eclipse 3.0 platform is the OSGi run time. This run time enables Java code from multiple sources to all run together in a single Java Virtual Machine (JVM). In OSGi, *bundles* are encapsulations of various files that have the following characteristics:

- Provide code, resources, and metadata
- Provide applications, services, or packages that can be used by other bundles

- Encapsulate Eclipse plug-ins
- Are easily versioned
- Can be served from remote bundle servers so that when bundles are deployed, the deployment program can automatically find and obtain their pre-requisite bundles

Bundles contain lists of their package and service prerequisites. The OSGi framework automatically loads and runs bundles. This provides the mechanism by which plug-ins can be automatically detected and loaded into the Eclipse IDE.

The RCP includes an Install/Update Manager, which takes advantage of the OSGi deployment capabilities. It provides a GUI to show the current configuration, that is, a list of installed plug-ins. It assists the end user in finding and installing new plug-ins. It is also capable of scanning through the list of already installed plug-ins to look for updates to these plug-ins. The component records the device's configuration history and enables you to undo changes or revert to previous configurations. You can also disable and later re-enable plug-ins without uninstalling or re-installing them.

UI toolkits

The following UI toolkits are used by the Eclipse IDE and plug-ins. These plug-ins work equally well for RCP applications.

The Standard Widget Toolkit (SWT) provides a completely platform-independent API that is tightly integrated with the operating system's native windowing environment. Java widgets actually map to the platform's native widgets. This gives Java applications a look and feel that makes them virtually indistinguishable from native applications. In cases where native function is not provided, SWT emulates it in a manner in keeping with the platform's normal look and feel. This toolkit overcomes many of the design and implementation trade-offs that developers face when using the Java Abstract Window Toolkit (AWT) or Java Foundation Classes (JFC). AWT gives the least common denominator approach and is therefore functionally limited. JFC is more flexible, but because all widgets are painted by the toolkit, JFC always seems to have trouble precisely emulating a native look and feel.

The JFace toolkit is a platform-independent user interface API that extends and interoperates with the SWT. This library provides a set of components and helper utilities that simplify many of the common tasks in developing SWT user interfaces. For example, it provides the dialogs, wizards, and rich text editors used by the Eclipse IDE. JFace also has tables and trees that utilize a model view controller (MVC) architecture to separate data access logic from data display logic. JFace also provides the mechanisms by which plug-ins programmatically contribute to the workbench, which is further discussed in the next topic.

RCP application model

In the Eclipse IDE model, plug-ins contribute to the standard IDE workbench. In the RCP model, the application *becomes* the workbench and can customize it in numerous ways. The workbench provides *extension points* that the plug-ins extend. The plug-ins provide functionality that is integrated into the GUI just as if it were always part of the application. Extensions can provide *views* and *perspectives*. Views are child windows of the workbench, which are drawn by plug-ins. Plug-ins can provide any number of views. Perspectives are conglomerations of views that can be from one or more plug-ins. The workbench's extension points also enable plug-ins to add functionality to the menu and tool bars of the workbench. Figure 1 shows the various components of the RCP application model and how they fit together.

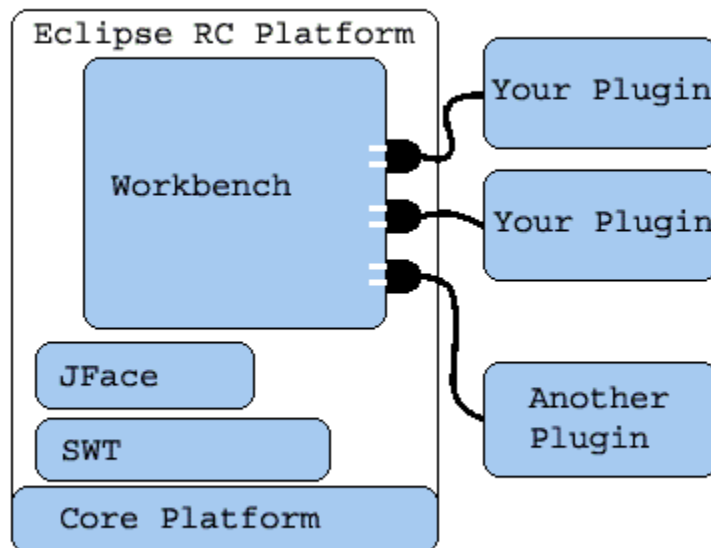


Figure 1. Eclipse RCP application model

How is RCP most useful?

RCP is most appropriate for larger applications where it makes sense to divide the application into components, or where you want to enable other developers to contribute function to your application. The benefit of using plug-ins for function that is normally all within the same application is that the function can easily be versioned and distributed separately.

Consider a productivity suite, with e-mail, calendar, and ToDo list components. Typically, even if an application is well structured, the components are all tied together within the single application. Updating any one of the components requires updating the

entire application. If the components are plug-ins, then updating one can be done easily and dynamically through the Eclipse RCP framework. Now imagine that the suite developer wants to add a scheduling component. A traditional application must be updated to make this new function accessible, but an RCP application simply detects that the new plug-in is available and can automatically incorporate it into the workbench GUI.

Building a Rich Client Platform application

The Rich Client Platform is really the minimal set of Eclipse SDK plug-ins needed to build an application with a UI. However, rich client applications are free to use any API necessary for their feature set, and can require any plug-ins beyond the bare minimum. The main thing that differentiates a rich client application from the standard SDK workbench is that the application is responsible for defining which class should be run as the main application. Your RCP application itself can be an Eclipse plug-in. As such, you can use a plugin.xml file to specify your application's start-up class, which creates the workbench window. More information can be found at the Rich Client Platform home page at <http://dev.eclipse.org/viewcvs/index.cgi/%7Echeckout%7E/platform-ui-home/rcp/index.html>.

RCP application deployment

Because RCP is built on the OSGi framework and applications are plug-ins built as OSGi bundles, a host of benefits is available to the RCP application delivery model. Applications can import the packages and services they need from other OSGi bundles. Thus, large applications can be deployed a piece at a time as connectivity allows, or on-demand as the user requires extra functionality. Bundles can be stopped, updated, and restarted under control of the framework. The bundles an application relies on can also be dynamically updated. This is all done without creating new JVMs or stopping and restarting the JVM that OSGi is running in. This capability saves the often considerable time a JVM takes to load and initialize.

Extending the RCP application model

The Rich Client Platform application model is applicable across a wide spectrum of device types, from desktop to embedded devices. The RCP platform architecture is already well adapted to a low-end environment because it enables running multiple applications in a single JVM. However, the size of the platform and desktop-oriented nature of the UI tool kit in the standard RCP platform might limit it to fairly capable devices such as desktop and notebook machines. A new Eclipse open source project is underway to address these issues and deliver a more targeted RCP application model for embedded devices. This embedded version of RCP is known as eRCP.

Because resources are generally limited on embedded devices, memory usage needs to be conserved where ever possible. The eRCP is a smaller platform and has a slightly

different application model, which conserves memory usage. In eRCP, the various components that make up the RCP are sub-set to produce a smaller footprint. For instance, rather than SWT, JFace, Workbench, and Update components, eRCP has eSWT, eJFace, eWorkbench, and eUpdate components. There are also Mobile Extensions for SWT that handle the unique requirements imposed by embedded devices that have small displays and limited user input features.

eRCP application model

Most OSGi framework capabilities provided in RCP are available in eRCP— for instance, enabling multiple applications to run within a single JVM. However, rather than applications each providing their own workbench, eRCP provides a single system workbench. The generic workbench provided by the eRCP platform is typically branded and optimized for a particular device by the device manufacturer or device customizer. All applications in the eRCP model are plug-ins that extend the system workbench. Because plug-in applications do not need to provide the workbench functionality or GUI, they too can generally be smaller than a similar application written for the full RCP platform.

eRCP perspectives and views

As mentioned in the previous topic, the eRCP platform provides a generic workbench that can be customized by manufacturers. In addition, the workbench can use different perspectives to handle different screen modes or orientation changes. For example, there could be a portrait perspective for when a smart phone is held in the normal orientation and a landscape perspective for when it is rotated. The landscape perspective might use a different layout of the parts of the workbench to take advantage of the different screen aspect ratio when in landscape mode. Because a plug-in application provides, or draws, a view wherever the perspective specifies, there is no need for the application to be cognizant of the orientation change. Applications simply receive a resize event with new window coordinates in which to draw.

As in the RCP model, applications can also provide perspectives. However, this capability is less likely to be of practical use on embedded devices, given that device screens are typical quite small. They do not often have the real estate to make simultaneous display of multiple views worthwhile.

eRCP compatibility with RCP

Because eRCP is basically a subset of RCP, applications developed to run on embedded devices automatically run on an RCP platform, as long as a Mobile Extensions bundle is available for that particular RCP platform. As eRCP applications will likely be optimized for small screens, display on a desktop might not be optimal, but the application should

be functional. Besides the direct benefit of application portability to other device classes, the underlying benefit of the model commonality is that if there is a demand for an eRCP application on the desktop, it should be very easy to upgrade the application to the full RCP model.

Sources of further information

The Eclipse IDE Help contains detailed information and a tutorial for creating an RCP application.

The Rich Client Platform home page has many links to more detailed information such as FAQs, articles, tutorials, and specific Help topics:

<http://dev.eclipse.org/viewcvs/index.cgi/%7Echeckout%7E/platform-ui-home/rcp/index.html>

The IBM developerWorks® site has an RCP tutorial at:

<http://www-128.ibm.com/developerworks/edu/os-dw-os-rcp1-i.html>

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

developerWorks

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.



Other company, product and service names may be trademarks or service marks of others.