Web-enabled RCP Applications with the Rich Ajax Platform

RCP goes Web 2.0

Based on an article by **Benjamin Muskalla** and **Ralf Sternberg**
in Eclipse-Magazin Vol. 12; translation by Innoopract Inc; last update Oct 19th 2007

The AJAX hype keeps growing. Buzzwords like Comet or RIA are on everybody's lips. In the area of enterprise software RCP is already firmly established as a client application platform. Replacing or extending existing Rich-Client-Applications with web front-ends has traditionally required a significant investment. The Rich Ajax Platform (RAP) provides a fast-path bridge between the application development with RCP and the increasingly important Web 2.0 environment.

How do I get my RCP-Application on the Web? This question is asked more and more these days. Besides pure provisioning solutions like Java Web Start there are different approaches that use Eclipse Equinox on the server and recreate the user interface using JSPs or similar technologies. This approach has the disadvantage of a reduced code reuse in the UI, since it is almost impossible to recreate the workbench concept and the familiar extension points with different technologies. This is remedied by RAP, a web-platform that can execute regular RCP applications with few restrictions. This gives developers the opportunity to take an existing RCP application and deliver it as an "ajaxified" web application with a small effort, without coming into contact with Servlets, HTML or JavaScript. With it, RCP concepts like Views, Perspectives or Wizards can be put into web applications.

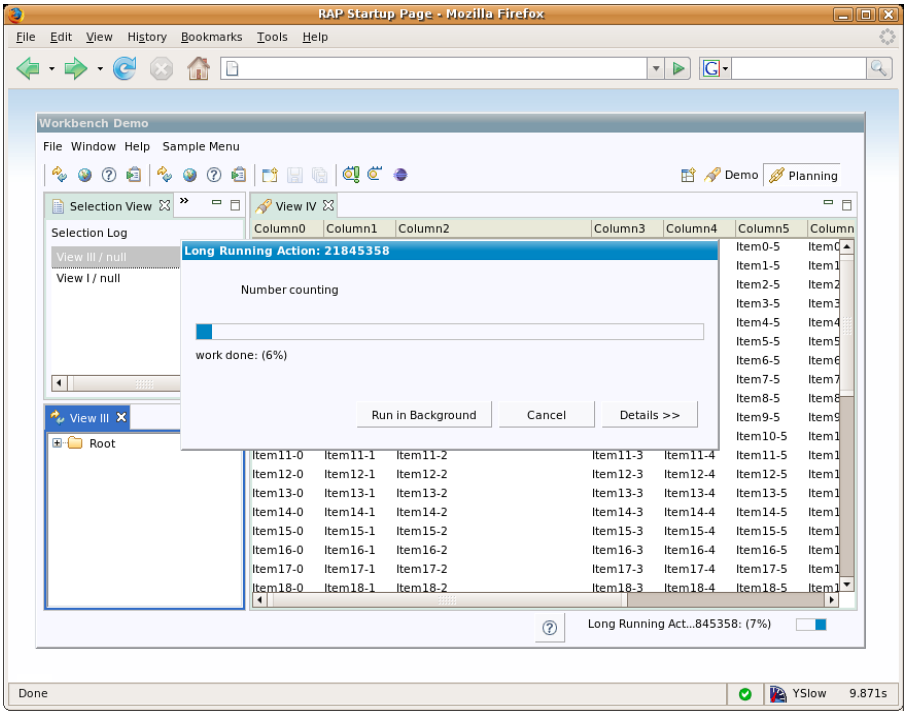


Figure 1: RAP in Action

The First Steps

In order to become familiar with RAP we will start by installing and starting the available demo application. To do this, we download a new RAP version from the project's homepage [1] and unpack it in any directory. RAP is being delivered as a target platform because it will be executed on a server and cannot use the standard implementations of SWT, JFace and the Workbench. The Eclipse IDE is best started with an empty workspace so that the target does not affect other projects. We now open the Target Platform preference page (Window | Preferences | Plug-in Development | Target Platform) and change the Location to the *eclipse* subdirectory of the unzipped target. Thereafter we import the supplied Demo-Project in the workspace. Under File | Import… we choose *Plug-ins* and *Fragments,* ensure that the project

will be imported with sources (*"Import as projects with source folders"*) and on the next page select the project *org.eclipse.rap.demo*. After importing has finished, the Run-Dialog (RUN | OPEN RUN DIALOG…) will contain a new OSGi Launch Configuration that we can use to start the demo application. The Console View should show something like this:

```
osgi>Jul 15, 2007 11:24:57 AM org, morthbay.
http.HttpServer doStart

INFO: Version Jetty/5.1x
Jul 15, 2007 11:24:57 AM org.mortbay.util.Contain-
er start
INFO: Started org mortbay.jet-
ty.servlet.ServletHandler@F0eed6

Jul 15, 2007 11:24:57 AM org.mortbay.util.Contain-
er start
INFO: Started HttpContext[/,/]
Jul 15, 2007 11:24:57 AM org.mortbay.http.Sock-
etListener start

INFO: StartedSocketListener on 0.0.0.0:9090
Jul 15, 2007 11:24:57 AM org.mortbay.util.Contain-
er start
INFO: Started org.mort-
bay.http.HttpServer@1d99a4d
```

This shows that the server has started correctly. We now open the address http://localhost:9090/rap in a browser and are inside of a RAP application (Figure 1). The immediately recognizable similarity to RCP

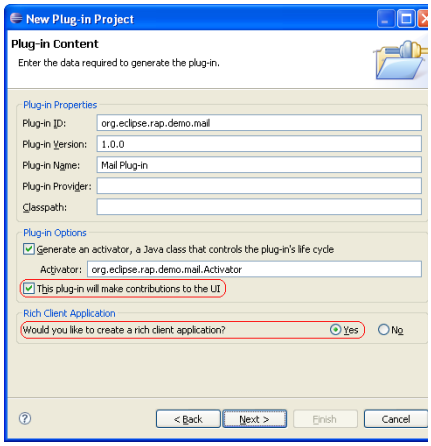


Figure 2: Preset of the Email Template

applications is not only of a visual nature. Views and Perspectives are present together with familiar RCP concepts like: Selection Service Commands and Handler, as well as the complete Editor infrastructure and data-binding introduced with Eclipse 3.3. The demo offers multiple Perspectives so that you can view more features live. While looking through the source code of the demo-project one notices the absence of JavaScript and HTML files. It is an actual RCP application.

From RCP to RAP

After running the demo application we should return to our original question and bring an existing application to the web. The well-known e-mail template from Eclipse will serve as a foundation, which we will transform into a RAP application. For that we will create a new plug-in in our workspace, call it *de.eclipsemag.rap.mail* and change some of the wizard's settings in order to generate the e-mail application (Figure 2). To be able to coplete the following steps it is necessary that the RAP Target Platform is installed and configured.

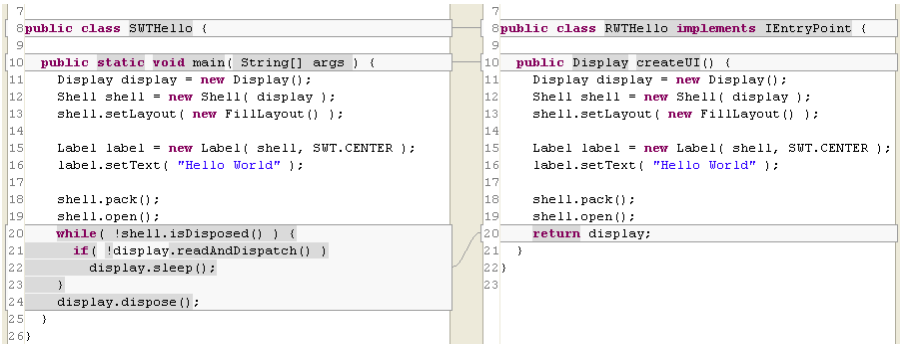
On the third page of the *New Project Wizard* we choose *RCP Mail Template* and keep the default settings. After the project creation is completed we switch to the *Dependencies* tab in the Plug-in Manifest Editor, delete the dependency on *org.eclipse.ui* and add *org.eclipse.rap.ui* instead. After that we go to the *Extensions-T*ab.

Here we add a new contribution for the Extension Point, *org.eclipse.rap.ui.entrypoint.* The mission of our Entrypoint is comparable to the *IApplication* implementation in the original. We set the attribute *parameter* to "*default*", which makes this extension the standard entrypoint. The attribute *class* expects a class which implements the interface *IEntrypoint*. Our entry point should look like the one shown in Listing 1.

Some compile errors emerge through these changes, as RAP is only a subset of RCP. Next we will address those errors, starting with the class *ApplicationActionBarAdvisor*. This class contains an action to open an additional workbench window. Until now this is not possible to do in RAP, so we just ignore this action and instead comment out the corresponding lines (32, 54, 55 and 74). We also go back to the plugin.xml and remove the contribution to the *org.eclipse.ui.bindings* extension point. Keyboard bindings are already preset by the browser that is hosting the application, so RAP does not support this extension point.

Session and Singletons

When porting a RCP application to RAP special consideration must begiven to all classes that implementthe Singleton pattern [3]. A Singletonis distinguished by the fact that onlyone instance of it can exist per application. In contrast to RCP, there are more than one active user sessions in a RAP application. That means that in RAP all users would access the same singleton object. In many cases this is exactly the original purpose of Singletons. In other cases a Session Singleton should be used instead, which is instantiated once per session (i.e. once per user). The building of such a Session Singleton is supported by RAP through the abstract class *SessionSingletonBase.* The JavaDocs of this class give more information about its use.



```
public class SWTHello {

  public static void main( String[] args ) {
    Display display = new Display();
    Shell shell = new Shell( display );
    shell.setLayout( new FillLayout() );

    Label label = new Label( shell, SWT.CENTER );
    label.setText( "Hello World" );

    shell.pack();
    shell.open();
    while( !shell.isDisposed() ) {
      if( !display.readAndDispatch() )
        display.sleep();
    }
    display.dispose();
  }
}
```

```
public class RWTHello implements IEntryPoint {

  public Display createUI() {
    Display display = new Display();
    Shell shell = new Shell( display );
    shell.setLayout( new FillLayout() );

    Label label = new Label( shell, SWT.CENTER );
    label.setText( "Hello World" );

    shell.pack();
    shell.open();
    return display;
  }
}
```

Figure 3: SWT and RWT in comparison

After these simple adjustments we can start our application for the first time. For that we use the launch configuration of the previously imported demo plug-in. We remove the bundles *org.eclipse.rap.demo* and *org.eclipse.rap.demo.databinding* from the launch configuration and add our recently created plug-in *de.eclipsemag.rap.mail*. A click on RUN starts the supplied Jetty-Server and makes the application available on the familiar URL http://localhost:9090/rap. In the browser we now find the e-mail example in a new attire − as Web 2.0 application. With a glance at the time we notice that this adjustment only took a few minutes. With an alternative technology, depending on the scope of its functions and services, this venture would probably have taken several hours – if not days to complete. Although this is only an example project it clearly shows what a powerful tool RAP is. Even when applications cannot be carried over one-to-one, the code reuse factor is very high.

How does RAP work?

RAP utilizes the basic components of Eclipse in order to execute RCP-Code in a Servlet Container. Equinox, the Eclipse OSGi implementation, supports the execution in a Servlet Container. Building on that, RAP uses many standard bundles from RCP, like *org.eclipse.core.runtime*. Thus the most important Eclipse concepts like the plug-in architecture and extension points are available. The plug-ins for SWT, JFace and the Workbench are replaced with their RAP equivalents. These implementations establish a connection between the RCP code that is running on the server and the users' browsers. The user interface on the client-side is rendered using the JavaScript library "qooxdoo" [2] which is developed by 1&1. In contrast to other technologies, RAP does not transform Java code into JavaScript, but exchanges information between client and server through AJAX.

An important difference between RCP and RAP is the inherent multi-user-requirement of web applications. The developer must take this into consideration when working with user data (see "Sessions and Singletons" box). It is important to notice that RAP already provides all available workbench functions in a session-oriented manner, so that the developer does not have to struggle with it.

The basis of RAP is formed by RWT – the RAP Widget Toolkit – which recreates a large part of the functionality of SWT. While SWT invokes native methods of the operating systems to create widgets and read events, RWT receives events as a request from the browser and sends JavaScript code back, which renders the widgets on the client. On top of RWT, RAP offers customized versions of JFace and the Eclipse Workbench. The plug-ins for Eclipse forms and data binding are also provided. Large parts of these plug-ins could be ported, which makes a high reuse of existing RCP-applications possible.

Given this, it is clear that an RCP- or RAP developer does not have to shift to a new programming or markup language, but can write his code in the usual RCP style and rely on previously used concepts. The really troublesome conversion is handled by the Rich Ajax Platform in the background. The API compatibility, to which RAP has committed itself, is a huge benefit to that cause.

SWT vs. RWT

What are the differences between programming with RWT and SWT? As you know, a code snippet says more than a thousand words. Figure 3 shows the source code of a normal SWT application in comparison to the corresponding RWT variant. First of all it is apparent that the starting point of the application is not the *main* method, but rather the *createUI* method of the Interface *IEntryPoint*. This difference is due to the multi-user ability of RAP. Furthermore, the handling of the event queue in RAP is taken over by

the Platform. However the actual application code stays the same.

RWT offers only a subset of SWT, as some parts could not be emulated offhand. An example of this is the *GC* (Graphics Context), which is used in SWT to draw directly onto surfaces. While some approaches for performing similar functions in JavaScript do exist, those are either browser specific or very crude. Furthermore frequently occurring events (like events for mouse movement) are posing a challenge. However the situation is not always hopeless. For example, it was possible implement the *ModifyListener* for text fields in RAP with only minimal semantic deviations, by collecting multiple keystrokes and transmitting those at short intervals as one packet. For the user the difference is nearly invisible.

Moreover, with RWT is it possible to write your own widgets. However this requires, just as in SWT, a good knowledge of the inner workings of the platform in use. In the case of RAP some knowledge about the basics of JavaScript, fundamentals of the Qooxdoo library and also the bottom layers of RWT will be beneficial. An interesting example of a custom widget is the Google Maps widget (Figure 4), which can be found in the RAP-Wiki [4].

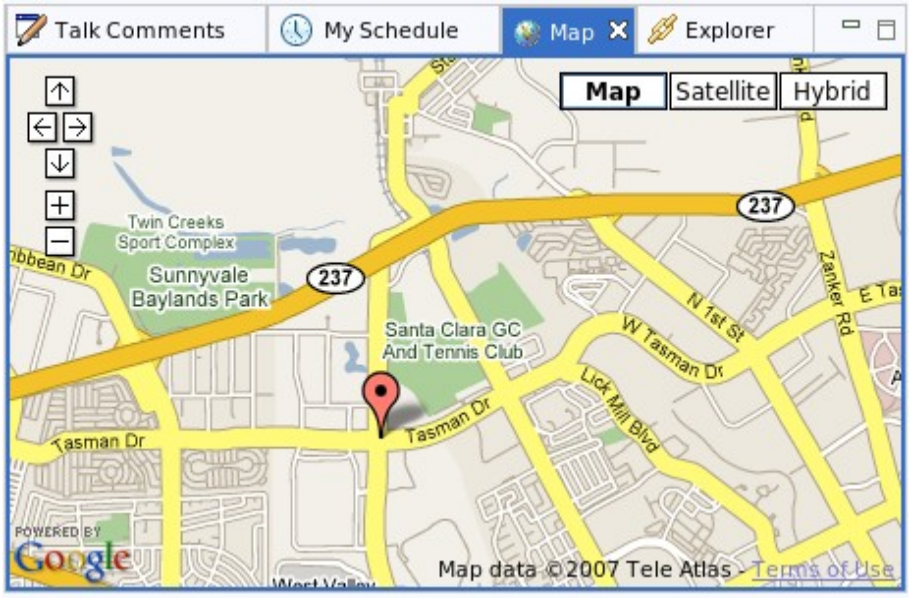


Figure 4: Example of a user-defined RAP widget: Google Maps

Digression: Theming

When you decide what technology to use, the look and the feel of the graphical interface plays a big role. That is also the case with web applications; a trendy outfit is a plus. To meet these expectations RAP is equipped with a theming infrastructure. This allows the developer to adapt the appearance to the intended usage. The appearance of an SWT application is linked to the current settings of the desktop environment. In a similar fashion it is possible to customize the appearance of a RAP-based application through theming settings. Besides fonts, background and text colors, it is possible to adjust the width and style of borders plus the inner and outer distances of different widgets. This enables the developer and/or designer to decide how they want to present the application. Theming also helps with the integration into an existing environment (keyword: corporate design). A good example of a RAP application without the typical workbench look is Eclipse Discovery [5].

In order make building a RAP theme as simple as possible, we have used a simple *.properties* file. The syntax of individual values resembles those used in CSS2. Therefore the theme developer does not have to learn a new syntax. Since theme definitions are purely declarative it is not even required to have programming knowledge.

The theme does not only affect the appearance in the browser. The server must also evaluate the adjustments that have been made to sizes, colors and fonts to be able to correctly layout the application. SWT system colors can also be changed at will. For example the method *Display#getSystemColor(SWT.-COLOR_LIST_BACKGROUND)* will return the color associated with "list.background" key in the theme. This is the same color that is used for the background of lists and text fields.

Listing 2 shows a snippet from an exemplary theme definition. The key *shell.title.background* configures the background color of the title bar of an active Shell (the "window" in SWT). This theme file must be registered as an extension in the *plugin.xml* (Listing 3). The attribute *default* determines if this is the default theme of the application.

A small tip: user defined themes can also be activated via an URL parameter. To do this the parameter *"?theme=<id>"*, specifying the id of the theme to be used, must be appended to the usual URL. By the way the demo plug-in also contains an alternate example theme *(org.eclipse.rap.theme.alttheme)*. Not supported yet is switching the theme in a running application. Up to date information and a good overview about theming of RAP applications can be found in [6].

Scalability & Performance

Every type of AJAX application requires a smooth interaction between the (thin-) client and the server. In RAP the biggest part of the event processing happens on the server so that the latency of the connection is an important criterion for the interactivity of the application. Practical experience has shown that RAP applications do also work smoothly over the Internet, since only a Delta – e.g. the current changes - are exchanged during a Request/Response cycle. The amount of data send over the wire is normally a few hundred bytes and the answer time is a few milliseconds.

Since the major components of the Eclipse functionality in RAP run on the server, scalability deserves particular attention. During load tests with 500 parallel user-sessions using a simple workbench the heap memory consumption of the JVM was approx. 100 MB, demonstrating the scalability of RAP applications.

Summary

The Rich Ajax Platform offers a possibility to develop web applications using the RCP paradigm and also to port existing applications to the web with a small effort. With this, RAP opens many new possibilities regarding the standardization of application platforms for desktops and web clients.

Even though this article is not able to cover all the details it should have given a good introduction into the development with RAP. More information can be found on the RAP homepage at Eclipse.org [1] Innopract's RAP web pages [8] and the RAP-Wiki [4]. Questions and suggestions can also be discussed with the developers in the very active newsgroup [7].

Listing 2

```
# Border for shells with BORDER
# or TITLE style
shell.BORDER.border: 3px solid #1695d4

# Height of the title bar of Shells
shell.title.height: 25px

# Background color for the title bar
# of active Shells
shell.title.background: #9dd0ea

# Text color for the title bar of active Shells
shell.title.foreground: white

# Font for title bar of Shells
shell.title.font: bold 14px Arial, Helvetica,
sans-serif
```

Listing 3

```
<extension
  id="my.name.space.themes"
  point="org.eclipse.rap.ui.themes">

  <theme
    id="my.name.space.them.skyblue1"
    name="Sky Blue Theme 1"
    file="theme1/theme.properties"
    default="false"/>

</extension>
```

Links & Literature

1. www.eclipse.org/rap
2. www.qooxdoo.org
3. en.wikipedia.org/wiki/Singleton_pattern
4. wiki.eclipse.org/RAP
5. www.eclipsediscovery.org
6. wiki.eclipse.org/RAP_Theming
7. news://eclipse.technology.rap
8. www.innoopract.com/en/products/rich-ajax-platform-rap.html

Benjamin Muskalla is a Developer and Consultant at Innoopract Informationssysteme in Karlsruhe and an independent author. He is an active contributor to the Eclipse Platform itself and a committer to the Rich Ajax Platform and the Linux Distributions Project.
eMail: bmuskalla@innoopract.com

Ralf Sternberg is a Developer at the Eclipse-specialist Innoopract in Karlsruhe. As a committer he is actively involved in the development of RAP since the beginning of this year. His focus lies on RWT and theming.
eMail: rsternberg@innoopract.com