Explore Eclipse's embedded Rich Client Platform

Your mobile device wants Eclipse

Chris Aniszczyk, Software Engineer, IBM, Software Group
Uriel Liu (liukl@tw.ibm.com), Software Engineer, IBM, Software Group

Summary: Get an introduction to the embedded Rich Client Platform (eRCP). Learn about the various components that make up eRCP and get some examples on how to use them in your applications.

Date: 21 Mar 2006
Level: Introductory
Activity: 2116 views
Comments: 0 (Add comments)

★★★★☆ Average rating

Background

The embedded Rich Client Platform (eRCP) came about as a way to bring the advancements of the Eclipse Rich Client Platform (RCP) and apply it to the embedded space.

The eRCP is made up of the following components:

- Standard Widget Toolkit (eSWT) -- Core, Expanded and Mobile extensions
- eJFace
- eWorkbench
- eUpdate

We will discuss each of these components and use code examples where appropriate.

---

eSWT

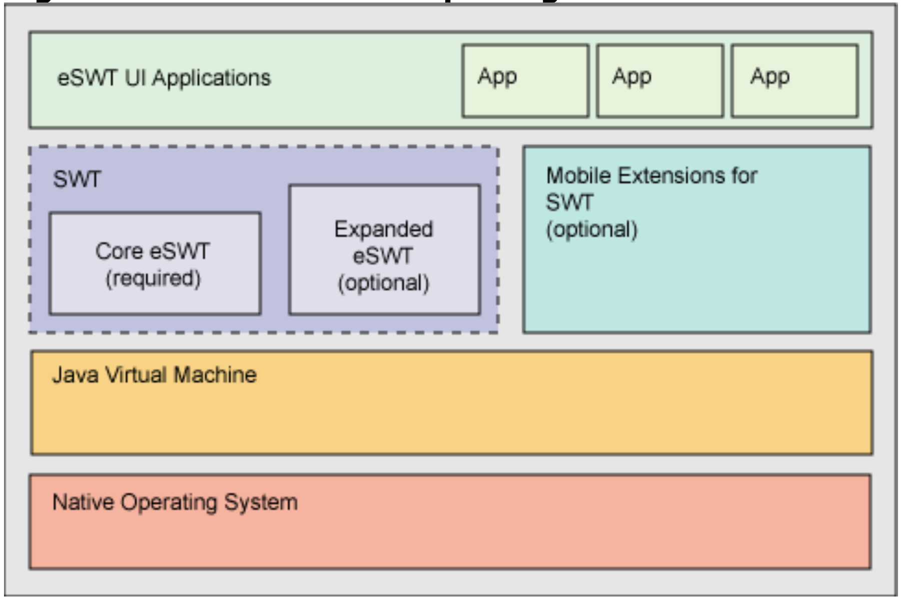
The embedded Standard Widget Toolkit (eSWT) is a subset of the well-known Java™ graphics tool kit, the Standard Widget Toolkit (SWT). It provides a set of controls, panels, and other widgets commonly used as building blocks of user interfaces (UIs). In addition to what was originally included in SWT, eSWT introduced a new component: mobile extensions (a jointly designed specification among IBM, Nokia, and Motorola), primarily targeted for mobile devices like PDAs and smart phones.

The design of eSWT is different from its cousin SWT in terms of platform-independence. SWT uses the platform-independent approach by trying to keep the native code as simple as possible to increase portability among operating systems. The problem is that portability and performance are competing issues, so eSWT decided on another approach: the Universal Graphics Layer (UGL), which still preserves the Java Native Interface (JNI) on the native tool kit implementation. But, instead of acting as a 1:1 JNI wrapper, UGL tries to keep the native implementation as close as possible, only requiring the information to callback through JNI. eSWT's approach sacrifices portability since the native tool kit completely depends on what graphics system is used, but this approach increases performance dramatically (a major concern in mobile devices).

There are three components included in eSWT (see Figure 1):

- Core
- Expanded
- Mobile Extensions

Figure 1. eSWT UI tool kit packages



The core and expanded components are subsets of eSWT. The newly invented mobile extensions component is targeted at mobile devices. This kind of componentization allows for flexibility to configure what components should be included on the device, based on device capability and purpose. The core component is mandatory and contains the minimum functions required to run a basic application. The expanded and mobile extensions components are optional.

In the following sections, we will walk through each component, along with sample applications.

eSWT Core

eSWT Core contains the fundamental UI elements, including low-level graphics, events, and basic widget infrastructure. Table 1 shows the classes in eSWT Core.

Table 1. eSWT Core org.eclipse classes

swt.widgets	swt.graphics	swt.events	swt.layout	swt.swt
Button	Color	ControlEvent	FormLayout	SWT
Canvas	Device	DisposeEvent	FormData	SWTException

Combo	Font	FocusEvent	FormAttachment	SWTError
Composite	FontData	KeyEvent	-	-
Control	FontMetrics	MenuEvent	-	-
Decorations	GC	ModifyEvent	-	-
Dialog	Image	MouseEvent	-	-
Display	ImageData	PaintEvent	-	-
Event	PaletteData	SelectionEvent	-	-
FileDialog	Point	ShellEvent	-	-
Item	Rectangle	TraverseEvent	-	-
Label	Resource	TypedEvent	-	-
Layout	RGB	VerifyEvent	-	-
List	-	-	-	-
Menu/MenuItem	-	-	-	-
MessageBox	-	-	-	-
ProgressBar	-	-	-	-
Scrollable	-	-	-	-
Scrollbar	-	-	-	-
Shell	-	-	-	-
Slider	-	-	-	-
Synchronizer	-	-	-	-
Text	-	-	-	-
TypedListener	-	-	-	-

eSWT Core example

Let's start with a complete eSWT program that creates a window that displays "HelloWorld from eSWT" on the application title bar. Then we'll add a few core controls (text, button, list). For layout, we'll use `FormLayout` and `FormData`. We'll also add `SelectionListener` on a button to bring up a message box. The complete code can be seen in Listing 1.

Listing 1: HelloWorld, eSWT style

```
public class HelloWorldeSWT {

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

    Text text = new Text(shell,SWT.SINGLE);
    text.setText("This is a text");
    Button buttonleft = new Button(shell, SWT.PUSH);
    buttonleft.setText("Left Push!");
    Button buttonright = new Button(shell, SWT.PUSH);
    buttonright.setText("Right Push!");
    buttonright.addSelectionListener(new SelectionListener(){
      public void widgetSelected(SelectionEvent e) {
        MessageBox messageBox = new MessageBox(shell,
          SWT.ICON_INFORMATION| SWT.YES | SWT.NO);
        messageBox.setText("MessageBox");
        messageBox.setMessage("Can you see me?");
        messageBox.open();
      }
      public void widgetDefaultSelected(SelectionEvent e) {
      }});
    List list = new List(shell,SWT.MULTI|SWT.BORDER);
    for (int i=0; i<5; i++) {
list.add("item "+i);
    }

    shell.setText("HelloWorld from eSWT");
    FormLayout layout = new FormLayout();
    layout.spacing = 5;
    layout.marginHeight = layout.marginWidth = 9;
    shell.setLayout(layout);

    FormData textData = new FormData();
    textData.top = new FormAttachment(0);
    textData.left = new FormAttachment(0);
    textData.right = new FormAttachment(90);
    text.setLayoutData(textData);

    FormData buttonleftData = new FormData();
    buttonleftData.top = new FormAttachment(text);
    buttonleftData.left = new FormAttachment(0);
    buttonleftData.right = new FormAttachment(40);
    buttonleft.setLayoutData(buttonleftData);

    FormData buttonrightData = new FormData();
    buttonrightData.top = new FormAttachment(text);
    buttonrightData.left = new FormAttachment(buttonleft);
    buttonright.setLayoutData(buttonrightData);
    FormData listData = new FormData();

    listData.top = new FormAttachment(buttonleft);
    list.setLayoutData(listData);

    shell.setSize(240,320);
    shell.open();

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

Figure 2 shows the application running on a Japanese edition PocketPC device. Once the user taps the **Right Push!** button, a message box will appear (as shown in Figure 3). eSWT utilizes the underlying native graphics library support to provide a consistent UI (a native application look and feel, rather than inventing a new look).

Figure 2. HelloWorld, eSWT style

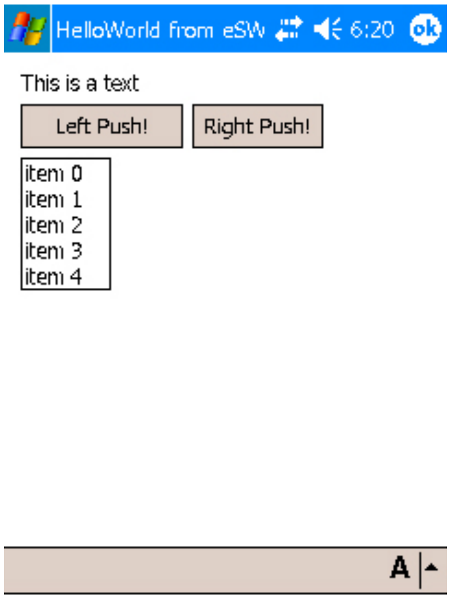
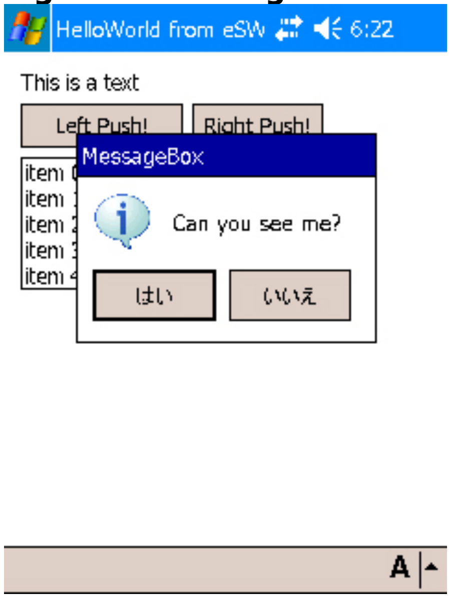


Figure 3. MessageBox with a native look and feel



Example evaluation

It's worthwhile to give a line-by-line explanation of what we accomplished in Listing 1.

Lines 04-05:
We create a display and a shell. The shell sets the display as its parent to make it become a top-level window. *Top-level windows* are automatically maximized on the PocketPC platform for consistency. We created the shell with a *final* modifier because it's used in the `SelectionListener` later.

Lines 07-08:
Creates text and calls `setText()` to set the text string.

Lines 09-22:
We create two buttons. In the right button, we add a `SelectionListener` to bring

up a simple message box.

Lines 23-26:
We create a list containing five items.

Line 28:
We set the text on the window's title bar.

Lines 29-53:
We use `FormLayout`, `FormData`, and `FormAttachment` to assist with the layout process.

Lines 55-56:
We set the shell size and open it. `setSize()` doesn't take effect for top-level window in this case (due to the device).

Lines 58-61:
Inside the `while()` loop, we establish an explicit loop to keep reading and dispatching user events from the operating system. If no more events are available, `display.sleep()` gets called and goes to sleep waiting for the next event.

Line 62:
The `display.dispose()` call at the last line of code explicitly disposes the display and releases any associated resources from the eSWT application.

eSWT Expanded

eSWT Expanded is an optional component that contains more sophisticated UI elements and layouts. This functionality is commonly found on high-end mobile devices and PDAs. Table 2 shows a detailed list of classes in eSWT Expanded.

Table 2. eSWT Expanded classes

org.eclipse.swt.widgets	**org.eclipse.swt.browser**	**org.eclipse.swt.dnd**	**org.eclipse.swt.graphics**
ColorDialog	Browser	ByteArrayTransfer	ImageLoader
DirectoryDialog	LocationEvent	Clipboard	-
FontDialog	ProgressEvent	TextTransfer	-
Table	StatusTextEvent	Transfer	-
Tree	TitleEvent	TransferData	-

eSWT Expanded example

The Browser widget is a fancy control in eSWT Expanded. We will create a fully functional Web browser using the Browser control (see Listing 2). We'll allow users to set the URL, move forward, backward, or reload the page in this example.

Listing 2. Simple HTML browser

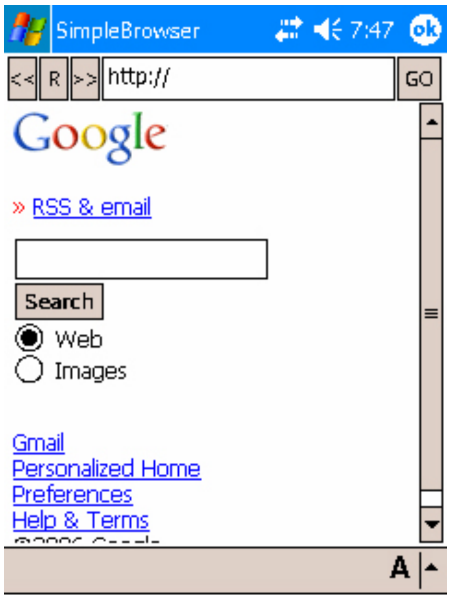
```
public class BrowserTest {

        static Button prev, reload, next, go;
        static Text url;
        static Browser browser;
```

```
    public static void main(String[] args) {
            final Display display = new Display();
            Shell shell = new Shell(display);

            //set window title and size
            shell.setText("SimpleBrowser");
            shell.setSize(240,320);

            //previous button
            prev = new Button(shell, SWT.PUSH);
            prev.setText("<<");
            prev.addSelectionListener(new SelectionListener(){
                    public void widgetSelected(SelectionEvent e) {
                            browser.back();
                    }
                    public void widgetDefaultSelected(SelectionEvent e) {
                    }});

            //reload button
            reload = new Button(shell, SWT.PUSH);
            reload.setText("R");
            reload.addSelectionListener(new SelectionListener(){
                    public void widgetSelected(SelectionEvent e) {
                            browser.refresh();
                    }
                    public void widgetDefaultSelected(SelectionEvent e) {
                    }});

            //next button
            next = new Button(shell, SWT.PUSH);
            next.setText(">>");;
            next.addSelectionListener(new SelectionListener(){
                    public void widgetSelected(SelectionEvent e) {
                            browser.forward();
                    }
                    public void widgetDefaultSelected(SelectionEvent e) {
                    }});

            // url text
            url = new Text(shell, SWT.SINGLE|SWT.BORDER);
            url.setText("http://");

            // go button
            go = new Button(shell, SWT.PUSH);
            go.setText("GO");
            go.addSelectionListener(new SelectionListener(){
                    public void widgetSelected(SelectionEvent e) {
                            // TODO Auto-generated method stub
                            browser.setUrl(url.getText());
                    }
                    public void widgetDefaultSelected(SelectionEvent e) {
                    }});

            // browser
            browser = new Browser(shell, SWT.NONE);
            browser.setUrl("http://www.google.com");

            FormLayout formLayout = new FormLayout();
            shell.setLayout(formLayout);
            formLayout.spacing = 1;
            formLayout.marginHeight = formLayout.marginWidth = 2;

            FormData prevData = new FormData();
            prevData.left = new FormAttachment(0);
```

```
                prevData.top = new FormAttachment(0);
                prevData.width = 16;
                prev.setLayoutData(prevData);

                FormData reloadData = new FormData();
                reloadData.left = new FormAttachment(prev);
                reloadData.top = new FormAttachment(0);
                reloadData.width = 16;
                reload.setLayoutData(reloadData);

                FormData nextData = new FormData();
                nextData.left = new FormAttachment(reload);
                nextData.top = new FormAttachment(0);
                nextData.width = 16;
                next.setLayoutData(nextData);

                FormData urlData = new FormData();
                urlData.left =  new FormAttachment(next);
                urlData.top = new FormAttachment(0);
                urlData.right = new FormAttachment(89);
                urlData.bottom = new FormAttachment(browser);
                url.setLayoutData(urlData);

                FormData goData = new FormData();
                goData.left = new FormAttachment(url);
                goData.top = new FormAttachment(0);
                goData.width = 24;
                go.setLayoutData(goData);

                FormData browserData = new FormData();
                browserData.top = new FormAttachment(prev);
                browserData.left = new FormAttachment(0);
                browserData.right = new FormAttachment(100);
                browserData.bottom = new FormAttachment(100);
                browser.setLayoutData(browserData);

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

Figure 4 shows the simple browser running on PocketPC.

Figure 4. A simple browser



eSWT Mobile Extensions

eSWT Mobile Extensions is an optional component that provides UI elements commonly found on myriad mobile devices. Table 3 shows classes in the eSWT Mobile Extensions component.

Table 3. eSWT Mobile Extension classes

org.eclipse.ercp.swt.mobile
CaptionedControl
Command
ConstrainedText
DateEditor
HyperLink
Input
ListBox/ListBoxItem
MobileDevice/MovileDeviceEvent/MobileDeviceListener
MobileShell
MultiPageDialog
QueryDialog
Screen/ScreenEvent/ScreenListener
SortedList
TaskTip
TextExtension
TimedMessageBox

eSWT Mobile Extensions example

In this example, we'll use `MobileShell` instead of `Shell` to provide full-screen functionality. `MobileShell` contains a `SortedList` on the top and a `ListView` on the bottom. Commands are associated with the `MobileShell` for the user to change full-screen mode and with `ListView` to change layout density. The source is in Listing 3.

Listing 3. MobileExtension example

```
public class MobileExtensionSample implements IPlatformRunnable {
        public static void main(String[] args) {
                Display display = new Display();
                final MobileShell shell = new MobileShell(display);
                final Button resetButton = new Button(shell, SWT.PUSH|SWT.BORDER);

                Command shellCommand = new Command(shell, Command.SELECT,0);
                shellCommand.setText("FullScreen");
                shellCommand.addSelectionListener(new SelectionListener(){
                        public void widgetSelected(SelectionEvent e) {
                                shell.setFullScreenMode(true);
                                resetButton.setVisible(true);
                        }
                        public void widgetDefaultSelected(SelectionEvent e) {
                        }});

                resetButton.setText("Normal Screen");
                resetButton.addSelectionListener(new SelectionListener(){
                        public void widgetSelected(SelectionEvent e) {
                                shell.setFullScreenMode(false);
                                resetButton.setVisible(false);
                        }
                        public void widgetDefaultSelected(SelectionEvent e) {
                        }});

                // Create SortedList and add items
                SortedList sortedList = new SortedList(
                                shell,
                                SWT.MULTI|SWT.V_SCROLL|SWT.BORDER,
                                SortedList.FILTER);
                sortedList.add("banana");
                sortedList.add("123");
                sortedList.add("12");
                sortedList.add("happyhour");
                sortedList.add("toobad");
                sortedList.add("youknowwhat");
                sortedList.add("yes");
                sortedList.add("886222333");

                // Create ListView and add items with image set
                Image[] image = new Image[4];
                image[0] = new Image(
                                Display.getDefault(),
                                MobileExtensionSample.class.getResource\
                                AsStream("/icons/sample.gif"));
                image[1] = new Image(
                                Display.getDefault(),
                                MobileExtensionSample.class.getResource\
                                AsStream("/icons/sample.gif"));
                image[2] = new Image(
                                Display.getDefault(),
                                MobileExtensionSample.class.getResource\
                                AsStream("/icons/sample.gif"));
                image[3] = new Image(
                                Display.getDefault(),
                                MobileExtensionSample.class.getResource\
                                AsStream("/icons/sample.gif"));

                final ListView lv = new ListView(shell, SWT.MULTI|SWT.BORDER);
                for (int i=0; i<20; i++) {
                        lv.add("item"+i, image[i % 4]);
                }

                //Create a Command for setting low density
                Command lowCommand = new Command(lv, Command.SELECT, 0);
                lowCommand.setText("LOW");
                lowCommand.addSelectionListener(new SelectionListener(){
                        public void widgetSelected(SelectionEvent e) {
                                lv.setLayoutDensity(ListView.LOW);
                        }
                        public void widgetDefaultSelected(SelectionEvent e) {
                        }});

                //Create a Command for setting high density
                Command midCommand = new Command(lv, Command.SELECT, 0);
                midCommand.setText("MEDIUM");
                midCommand.addSelectionListener(new SelectionListener(){
                        public void widgetSelected(SelectionEvent e) {
                                lv.setLayoutDensity(ListView.MEDIUM);
                        }
                        public void widgetDefaultSelected(SelectionEvent e) {
                        }});

                FormLayout layout = new FormLayout();
                layout.spacing = 2;
                layout.marginHeight = layout.marginHeight = 2;
                shell.setLayout(layout);
                FormData sortedListData = new FormData();
                sortedListData.top = new FormAttachment(0);
                sortedListData.left = new FormAttachment(0);
                sortedListData.right = new FormAttachment(100);
                sortedListData.height = 120;
                sortedList.setLayoutData(sortedListData);
                FormData lvData = new FormData();
                lvData.top = new FormAttachment(sortedList);
                lvData.right = new FormAttachment(100);
                lvData.left = new FormAttachment(0);
                lvData.height = 130;
                lv.setLayoutData(lvData);
                FormData resetData = new FormData();
                resetData.top = new FormAttachment(lv);
                resetData.left = new FormAttachment(0);
                resetButton.setLayoutData(resetData);
                resetButton.setVisible(false);

                shell.setSize(240,320);
                shell.setText("Mobile Example");
                shell.open();
                while (!shell.isDisposed()) {
                        if (!display.readAndDispatch())
                                display.sleep();
                }
                display.dispose();
        }
}
```

Figure 5 shows the result running on the PocketPC platform, and the full-screen mode is shown in Figure 6.

Figure 5. MobileShell with SortedList and ListView

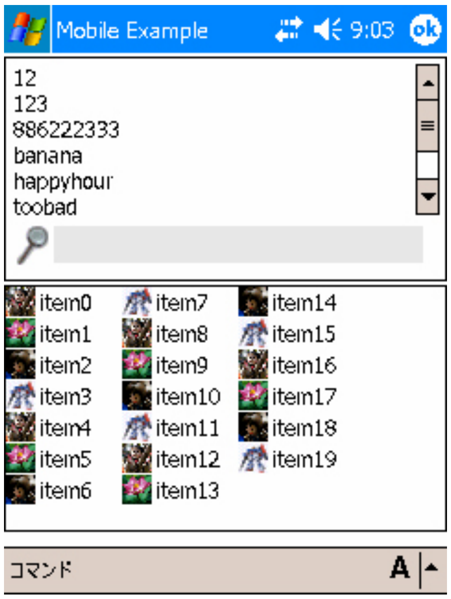
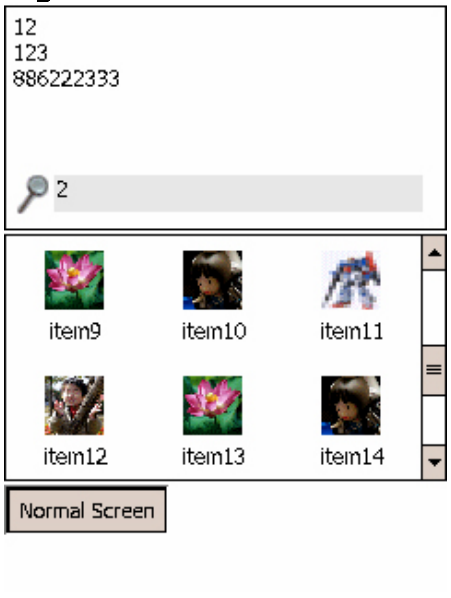


Figure 6. Full-screen MobileShell



---

eJFace

eJFace is a platform-independent UI tool kit that depends on eSWT. eJFace provides a set of components and help utilities that simplify the development of eSWT-based applications (by wrapping the eSWT widgets), just like JFace does for SWT. In fact, eJFace is a strict subset of JFace, so it shares many similarities with its cousin. eJFace provides support for resource management, viewers, actions, and preference pages. There are tutorials in Resources covering JFace that can aid in your eJFace endeavors.

Viewer types available in eJFace:

- `CheckBoxTableViewer`
- `CheckBoxTreeViewer`
- `ComboViewer`

- `ListViewer`
- `TableViewer`
- `TreeViewer`

---

eWorkbench

eWorkbench allows eRCP applications to run simultaneously inside a single workbench window, similar to how it is in RCP. eWorkbench clients provide views for specific display scenarios, and eWorkbench automatically decides which view to use based on the mobile device in use. In eWorkbench, there is no concept of perspectives -- it can be considered that there is only one perspective applications share -- due to the applicability of the concept to embedded devices.

eWorkbench application development

There are a couple steps (familiar to RCP developers) for creating an eWorkbench application. The process is similar to creating an Eclipse RCP application using the concept of contributions.

Step 1: Define your views

eWorkbench lets you define three types of views that extend `org.eclipse.ui.part.ViewPart`. The normal view is required; the other two views are optional.

- **Normal:** The default view
- **Large:** The view used when the display is large
- **Status:** The view used when the display is small

Now we create a sample view to use.

Listing 4. Sample view

```
public class DefaultView extends ViewPart {

        public void createPartControl(Composite parent) {
        //create a composite with fill layout to host a label
        Composite composite = new Composite(parent, SWT.NONE);
        composite.setLayout(new FillLayout());

        // create a label
        Label label = new Label(composite,SWT.CENTER);
        label.setText("Hello eWorkbench!"); }

        public void setFocus() {}
}
```

Let Eclipse know we have views available by using the extension point mechanism (see Listing 5).

Listing 5. plugin.xml

```
<extension point="org.eclipse.ui.views">
        <view allowMultiple="false"
                category="org.eclipse.ercp.eworkbench.viewCategory"
                class="com.ibm.ercp.application.views.DefaultView"
                icon="icons/sample.gif"
                id="com.ibm.ercp.application.defaultView"
                name="Sample DefaultView"/>
</extension>
```

Step 2: Define your eWorkbench contribution

To be considered an eWorkbench application, you have to extend the org.eclipse.ercp.eworkbench.applications extension point and provide some information (see Listing 6 for an example):

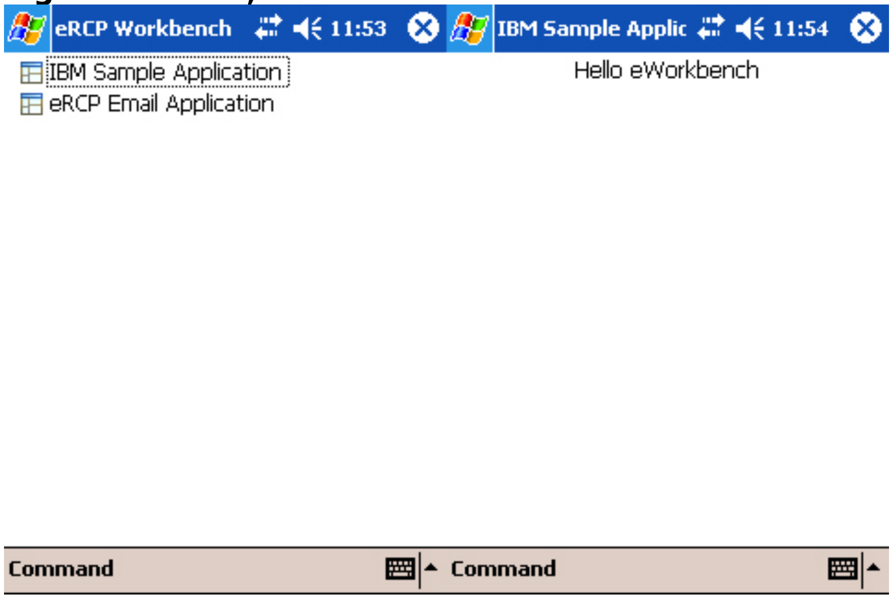
- **id:** The unique identifier that will represent your eWorkbench application
- **name:** The name of your application (displayed in the workbench)
- **views:** The views that your application will support (normal, large, support)

Listing 6. plugin.xml

```
<extension point="org.eclipse.ercp.eworkbench.applications">
        <application id="com.ibm.ercp.application" name=\
        "IBM Sample Application" singleton="true">
                <views normal="com.ibm.ercp.application.views.normal" />
        </application>
</extension>
```

Figure 7 shows a side-by-side screenshot of the eWorkbench application list, followed by the launch of our sample application.

Figure 7. Hello, eWorkbench



eUpdate

An advantage RCP offers is the ability to update plug-ins from a centralized server using the update manager interface. There are also features associated with the update manager, such as scheduled updates. Also, by using the Open Services Gateway Initiative (OSGi) and plug-ins, features can be installed dynamically. eRCP's answer to these advantages is eUpdate, which at the time of this writing is being developed heavily.

eUpdate provides the logic and UI for plug-ins wishing to contain update management capabilities. You can write your own update functionality with the help of these plug-ins or by using an eUpdate workbench application that provides a complete GUI for configuring update-related information.

Conclusion

This article introduced the embedded Rich Client Platform (eRCP) and its various components, providing example code that includes the sample eWorkbench application and other code listings as self-contained examples.

Download

Description	Name	Size	Download method
Sample code	os-ecl-rcp-com.ibm.ercp.application_1.0.0.jar	25KB	HTTP

Information about download methods

Resources

Learn

- Learn more about the embedded Rich Client Platform.
- Learn more about using JFace through "How to use the JFace Tree Viewer" at Eclipse.org.
- Find out more about using eSWT with a few SWT Snippets.
- Check out the Eclipse Foundation and its many projects.
- Learn more about the Open Services Gateway Initiative at OSGi.org.
- Visit developerWorks' Eclipse project resources to learn more about Eclipse.
- Stay current with developerWorks technical events and webcasts.
- Visit the developerWorks Open source zone for extensive how-to information,

tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- See the latest Eclipse technology downloads at IBM alphaWorks.
- Innovate your next open source development project with IBM trial software, available for download or on DVD.

Discuss

- Get involved with eRCP development by checking out the eRCP mailing lists.
- Seek help at the eRCP newsgroups first.
- The Eclipse newsgroups has lots of resources for people interested in using and extending Eclipse.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the authors

Chris Aniszczyk is a software engineer at IBM Lotus focusing on OSGi related development. He is an open source enthusiast at heart, and he works on the Gentoo Linux distribution and is a committer on a few Eclipse projects (PDE, ECF, EMFT). He's always available to discuss open source and Eclipse over a frosty beverage.

Uriel Liu is a software developer at the IBM China Software Development Lab and works in WED client technology. He is also a committer on the eRCP project.