Introduction to Eclipse's Rich Client Platform

Jeff Gunther
President
Intalgent
www.intalgent.com

Session TS-5979

Goal

Learn how to architect and build client-side Java™ applications using the Eclipse Rich Client Platform

Agenda

Overview of the Eclipse Rich Client Platform (RCP) and Its Place in the Development Landscape

Introduction to the Basic Steps to Create an RCP Application

Exploration of the Generic Workbench and the Core User Interface Components

Agenda

Overview of the Eclipse Rich Client Platform (RCP) and Its Place in the Development Landscape

Introduction to the Basic Steps to Create an RCP Application

Exploration of the Generic Workbench and the Core User Interface Components

Overview of Eclipse

The Maturation of Eclipse Project

- Eclipse is described as “…A kind of universal tool platform—an open extensible IDE for anything and nothing in particular”
- An Open Source, independent platform managed by the Eclipse Foundation
- Before the release of 3.0, Eclipse was traditionally only thought of as a Java integrated development environment (IDE)

What Is the RCP?

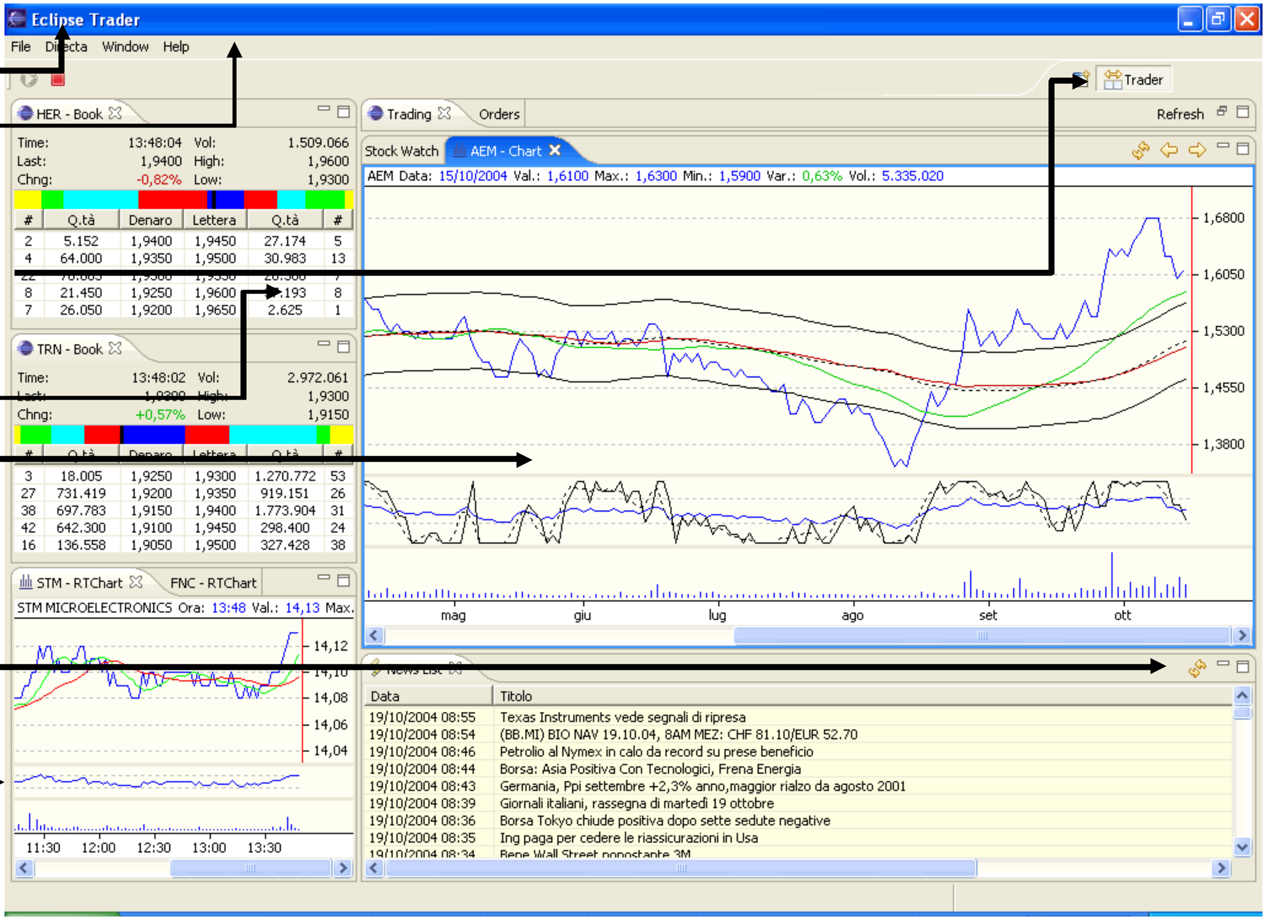
Details of the Eclipse RCP

- The minimal set of plug-ins needed to build a rich client application is collectively known as the Eclipse RCP
- RCP provides a generic Eclipse workbench that can be extended for a custom application
- An RCP application consists of at least one custom plug-in that uses the interface components of the Eclipse IDE

Elements of the Generic Workbench

User Interface Components of the RCP

- Workbench
- Menu bar
- Shortcut bar
- View
- Editor
- Tool bar
- Perspective



HER - Book

Time: 13:48:04 Vol: 1.509.066
Last: 1,9400 High: 1,9600
Chng: -0,82% Low: 1,9300

#	Q.tà	Denaro	Lettera	Q.tà	#
2	5.152	1,9400	1,9450	27.174	5
4	64.000	1,9350	1,9500	30.983	13
22	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
8	21.450	1,9250	1,9600	[illegible].193	8
7	26.050	1,9200	1,9650	2.625	1

TRN - Book

Time: 13:48:02 Vol: 2.972.061
Last: 1,9300 High: 1,9300
Chng: +0,57% Low: 1,9150

#	Q.tà	Denaro	Lettera	Q.tà	#
3	18.005	1,9250	1,9300	1.270.772	53
27	731.419	1,9200	1,9350	919.151	26
38	697.783	1,9150	1,9400	1.773.904	31
42	642.300	1,9100	1,9450	298.400	24
16	136.558	1,9050	1,9500	327.428	38

STM MICROELECTRONICS Ora: 13:48 Val.: 14,13 Max.

AEM Data: 15/10/2004 Val.: 1,6100 Max.: 1,6300 Min.: 1,5900 Var.: 0,63% Vol.: 5.335.020

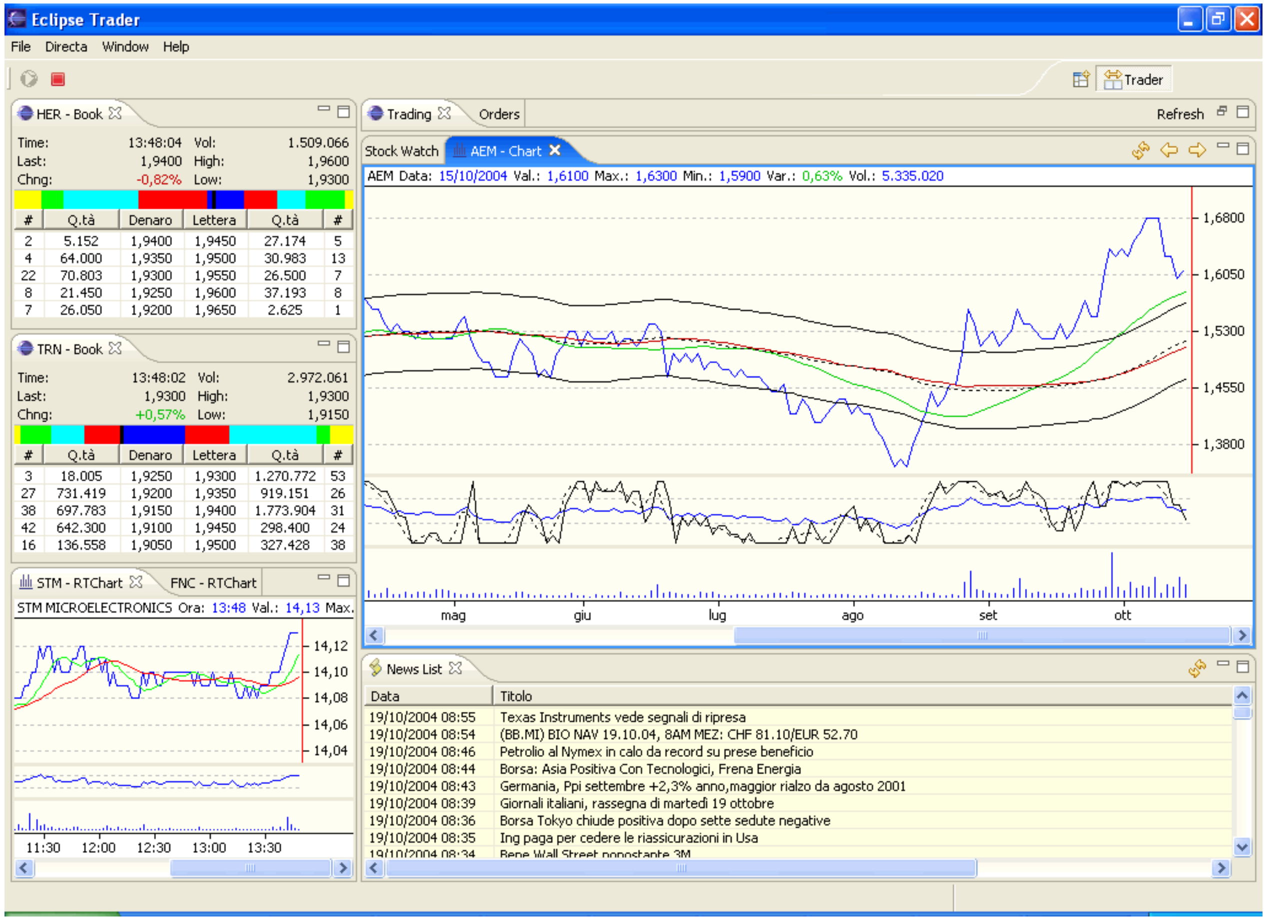
Data	Titolo
19/10/2004 08:55	Texas Instruments vede segnali di ripresa
19/10/2004 08:54	(BB.MI) BIO NAV 19.10.04, 8AM MEZ: CHF 81.10/EUR 52.70
19/10/2004 08:46	Petrolio al Nymex in calo da record su prese beneficio
19/10/2004 08:44	Borsa: Asia Positiva Con Tecnologici, Frena Energia
19/10/2004 08:43	Germania, Ppi settembre +2,3% anno,maggior rialzo da agosto 2001
19/10/2004 08:39	Giornali italiani, rassegna di martedì 19 ottobre
19/10/2004 08:36	Borsa Tokyo chiude positiva dopo sette sedute negative
19/10/2004 08:35	Ing paga per cedere le riassicurazioni in Usa
19/10/2004 08:34	Bene Wall Street nonostante 3M

Technology Comparison

Factor	HTML Clients	Applet	Standalone Rich Clients
User Interface	Moderate	Sophisticated	Sophisticated
Offline Support	No	No	Yes
Interactivity	Browser Limited	Browser Limited	Unlimited
Usability	Limited	Moderate	Sophisticated
Bandwidth Usage	Fixed	Variable	Variable

Example of an RCP Application

Eclipse Trader—Online Stock Trading System



HER - Book

Time: 13:48:04 Vol: 1.509.066
Last: 1,9400 High: 1,9600
Chng: -0,82% Low: 1,9300

#	Q.tà	Denaro	Lettera	Q.tà	#
2	5.152	1,9400	1,9450	27.174	5
4	64.000	1,9350	1,9500	30.983	13
22	70.803	1,9300	1,9550	26.500	7
8	21.450	1,9250	1,9600	37.193	8
7	26.050	1,9200	1,9650	2.625	1

TRN - Book

Time: 13:48:02 Vol: 2.972.061
Last: 1,9300 High: 1,9300
Chng: +0,57% Low: 1,9150

#	Q.tà	Denaro	Lettera	Q.tà	#
3	18.005	1,9250	1,9300	1.270.772	53
27	731.419	1,9200	1,9350	919.151	26
38	697.783	1,9150	1,9400	1.773.904	31
42	642.300	1,9100	1,9450	298.400	24
16	136.558	1,9050	1,9500	327.428	38

STM MICROELECTRONICS Ora: 13:48 Val.: 14,13 Max.

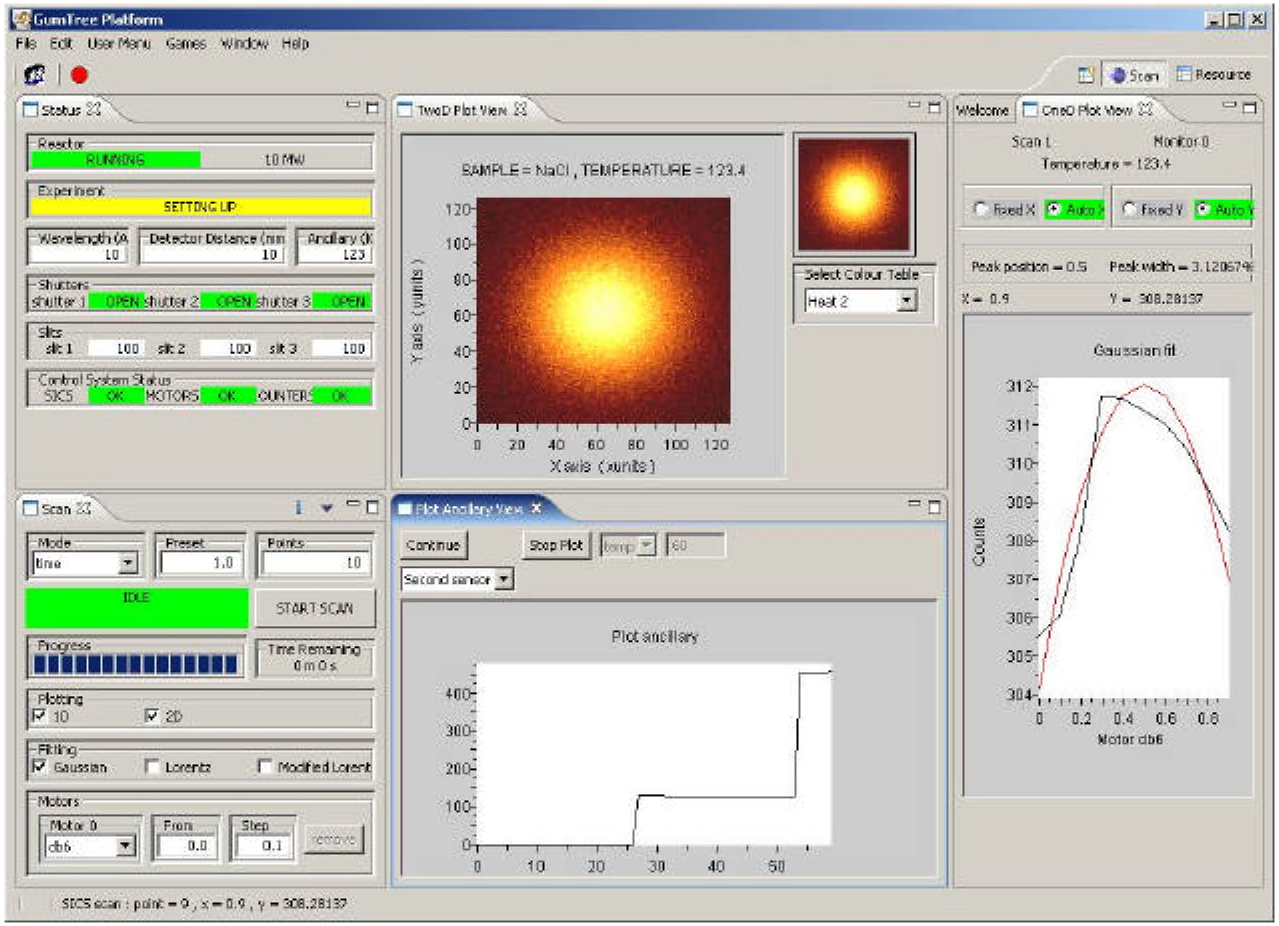
AEM Data: 15/10/2004 Val.: 1,6100 Max.: 1,6300 Min.: 1,5900 Var.: 0,63% Vol.: 5.335.020

News List

Data	Titolo
19/10/2004 08:55	Texas Instruments vede segnali di ripresa
19/10/2004 08:54	(BB.MI) BIO NAV 19.10.04, 8AM MEZ: CHF 81.10/EUR 52.70
19/10/2004 08:46	Petrolio al Nymex in calo da record su prese beneficio
19/10/2004 08:44	Borsa: Asia Positiva Con Tecnologici, Frena Energia
19/10/2004 08:43	Germania, Ppi settembre +2,3% anno,maggior rialzo da agosto 2001
19/10/2004 08:39	Giornali italiani, rassegna di martedì 19 ottobre
19/10/2004 08:36	Borsa Tokyo chiude positiva dopo sette sedute negative
19/10/2004 08:35	Ing paga per cedere le riassicurazioni in Usa
19/10/2004 08:34	Bene Wall Street nonostante 3M

Example of an RCP Application

GumTree—Instrument Control Program



Reactor: RUNNING 10 MW
Experiment: SETTING UP
Wavelength (A: 10 | Detector Distance (mm: 10 | Ancillary (K: 123
Shutters: shutter 1 OPEN shutter 2 OPEN shutter 3 OPEN
Slits: slit 1 100 slit 2 100 slit 3 100
Control System Status: SICS OK MOTORS OK OUNTERS OK

SAMPLE = NaCl , TEMPERATURE = 123.4
Y axis (yunits) / X axis (xunits)
Select Colour Table: Heat 2

Scan 1 Monitor 0
Temperature = 123.4
Fixed X [✓ Auto X] Fixed Y [✓ Auto Y]
Peak position = 0.5 Peak width = 3.1206748
X = 0.9 Y = 308.28137
Gaussian fit
Counts / Motor db6

Mode: time | Preset: 1.0 | Points: 10
IDLE
Progress | Time Remaining 0 m 0 s
Plotting: [✓ 1D] [✓ 2D]
Fitting: [✓ Gaussian] Lorentz Modified Lorentz
Motors: Motor 0: db6 | From: 0.0 | Step: 0.1

Plot ancillary

SICS scan : point = 9 , x = 0.9 , y = 308.28137

Why Use the Rich Client Platform?

Benefits of the RCP

- Elegant and extensible plug-in architecture
- Highly customizable workspace and user interface components
- Good interoperability with other technologies
- Scalable from desktops to embedded devices
- Wide cross-platform support
- Transparent to the end user

Core RCP Components

Technology Components of the RCP

- Standard Widget Toolkit (SWT)—Provides developers a platform-independent API that is tightly integrated with the operating system’s native windowing environment
- JFace Toolkit—Platform-independent user interface API that extends and interoperates with SWT; includes a variety of components and utility classes
- Eclipse Runtime—Provides the foundation for plug-ins, extension points and extensions
- Generic Workbench—Multi-window environment for managing views, editors, perspectives, actions, wizards, preference pages, etc.

Optional RCP Components

Technology Components of the RCP

- Help—Web based interface with support for full text indexing and dynamic content
- Update manager—Framework to discover and install updated versions of plug-ins and extensions
- Text/Forms—Frameworks for constructing text editors and forms
- Welcome page—Initial greeting upon application startup

Optional RCP Components (Cont.)

Technology Components of the RCP

- Cheat sheets—Guides used to walk users through a long running, multi-step task
- Resources—Workspace for managing projects, folders, and files
- Console—Extendable console view
- Outline and properties—Extendable outline and property views

Optional RCP Components (Cont.)

Technology Components of the RCP

- Graphical Editing Framework (GEF)—Framework for building graphical editors; Includes Draw2D, a vector graphics framework
- Eclipse Modeling Framework (EMF)—Framework for modeling and code generation
- Service Data Objects (SDO)—Framework that simplifies and unifies data application development in a service oriented architecture (SOA)

Agenda

Overview of the Eclipse Rich Client Platform (RCP) and Its Place in the Development Landscape

Introduction to the Basic Steps to Create an RCP Application

Exploration of the Generic Workbench and the Core User Interface Components

Steps to Create an RCP Application

High Level Steps to Create an Application

- Identify Extension Points
- Define the Plug-In Manifest
- Implement an **`Application`** Class
- Implement a **`WorkbenchAdvisor`** Class
- Implement a **`WorkbenchWindowAdvisor`** Class
- Implement supporting Extensions
- Export and deploy the application

The Plug-In Development Environment

Plug-In Perspective Within the Eclipse IDE

- The Eclipse IDE contains a specialized perspective to create and package an Eclipse RCP application
- To access this perspective:
    - Launch Eclipse 3.1
    - Select **Window > Open Perspective > Other** from the menu bar
    - Choose **Plug-in Development**

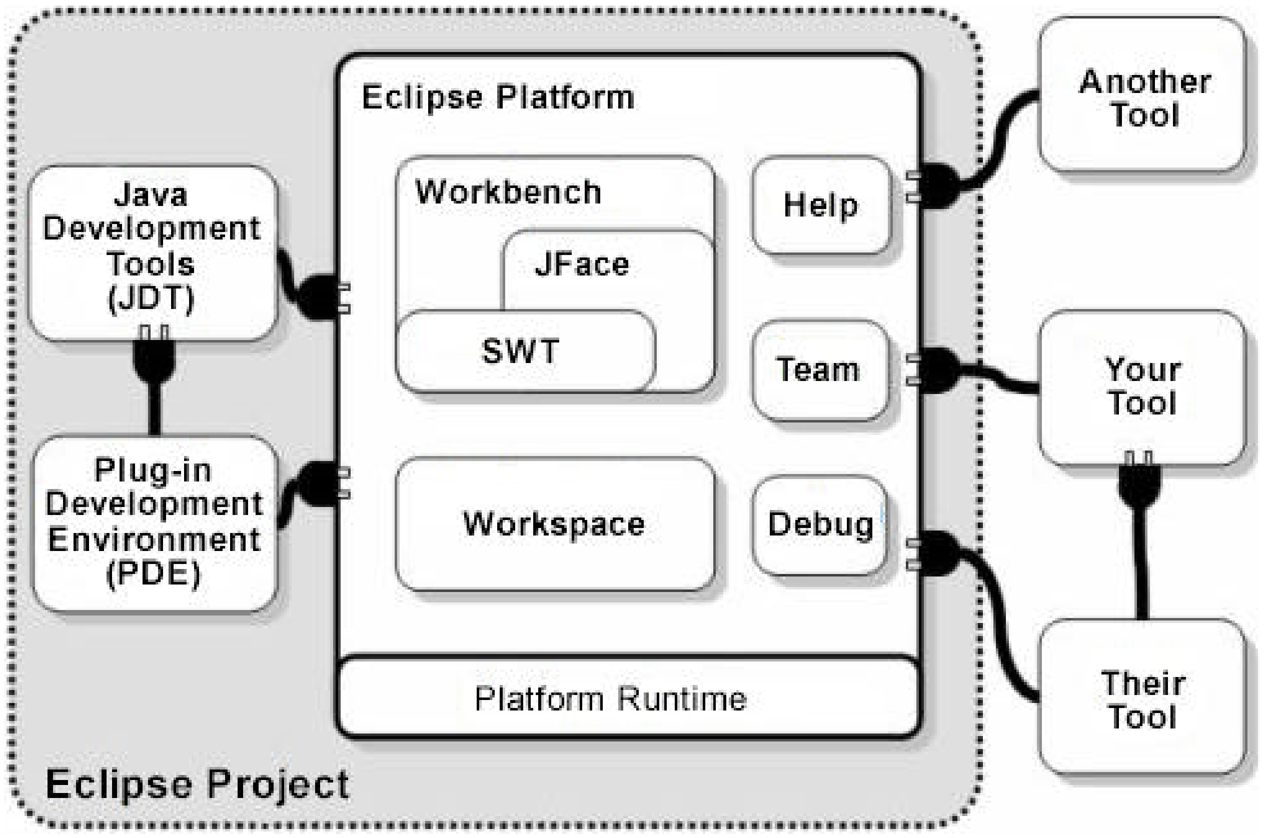
Understanding Extensions

Introduction to Extensions

- Platform is extremely extensible through the use of a relatively small runtime kernel and its elegant plug-in architecture
- New functionality is added to the runtime kernel through the use of plug-ins
- A plug-in can define its own set of extension points that other developers can utilize within their own plug-ins or RCP applications

Eclipse Plug-In Architecture

Component Relationships



Understanding the Plug-In Manifest

Introduction to the Plug-In Manifest

- The New Plug-in Project wizard will generate a plug-in manifest
- The manifest is an XML document that is responsible for defining:
  - Resources
  - Dependencies
  - Extensions and extension points
- The plug-in manifest is always located within the project's root directory
- The file is named "plugin.xml"

The Applications Extension Point

Introduction to the Applications Extension Point

- The first extension is defined through the **`org.eclipse.core.runtime.applications`** extension point. This extension is used to declare the entry point for an RCP application
- The Application class implements the **`IPlatformRunnable`** interface
- Responsible for the lifecycle of the application

Using the Application Extension Point

Extension Point and Supporting Class

```
<extension id="application"
   point="org.eclipse.core.runtime.applications">
   <application>
      <run
         class="com.intalgent.google.Application">
      </run>
   </application>
</extension>
```

```
public class Application implements IPlatformRunnable {
   public Object run(Object args) throws Exception{
      …
   }
}
```

Implementing an Application Class

```
public class Application implements IPlatformRunnable{
  public Object run(Object args) throws Exception{
    Display d = PlatformUI.createDisplay();
    try {
      WorkbenchAdvisor a = new ApplicationWorkbenchAdvisor();
      int c = PlatformUI.createAndRunWorkbench(d,a);
      if (c == PlatformUI.RETURN_RESTART) {
        return IPlatformRunnable.EXIT_RESTART;
      }
      return IPlatformRunnable.EXIT_OK;
    } finally {
      d.dispose();
    }
  }
}
```

Configuring the Workbench

Supporting Classes to Configure the Workbench

- Three base classes are used to configure and control the workbench:
  - **`WorkbenchAdvisor`** class is used to configure the workbench
  - **`WorkbenchWindowAdvisor`** class is used to configure the workbench window
  - **`ActionBarAdvisor`** class is used to configure the action bars of the workbench
- The New Plug-in Project wizard will automatically create these classes within the project

Implementing a `WorkbenchAdvisor` Class

```
public class ApplicationWorkbenchAdvisor
      extends WorkbenchAdvisor {

  public WorkbenchWindowAdvisor
      createWorkbenchWindowAdvisor(
            IWorkbenchWindowConfigurer configurer) {
      return new
      ApplicationWorkbenchWindowAdvisor(configurer);
  }
…
```

Implementing a `WorkbenchAdvisor` Class (Cont.)

```
…
   public void initialize(IWorkbenchConfigurer
        configurer) {
        super.initialize(configurer);
        configurer.setSaveAndRestore(true);
   }

   public String getInitialWindowPerspectiveId() {
        return Perspective.ID;
   }
}
```

Implementing a WorkbenchWindowAdvisor Class

```
public class ApplicationWorkbenchWindowAdvisor
      extends WorkbenchWindowAdvisor {

  public ApplicationWorkbenchWindowAdvisor
            (IWorkbenchWindowConfigurer configurer) {
            super(configurer);
  }

  public ActionBarAdvisor
            createActionBarAdvisor(IActionBarConfigurer
            configurer) {
      return new ApplicationActionBarAdvisor(configurer);
  }
…
```

Implementing a `WorkbenchWindowAdvisor` Class (Cont.)

```
…
   public void preWindowOpen() {
           IWorkbenchWindowConfigurer configurer =
               getWindowConfigurer();
          configurer.setInitialSize(new Point(400, 300));
          configurer.setShowCoolBar(false);
         configurer.setTitle("Google");
       }
}
```

Implementing the ActionBarAdvisor

```
public class ApplicationActionBarAdvisor extends
                   ActionBarAdvisor {
  private IWorkbenchAction aboutAction;
  public ApplicationActionBarAdvisor
      (IActionBarConfigurer configurer) {
      super(configurer);
  }

  protected void
      makeActions(IWorkbenchWindow window) {
      aboutAction = ActionFactory.ABOUT.create(window);
  }
…
```

Implementing the ActionBarAdvisor (Cont.)

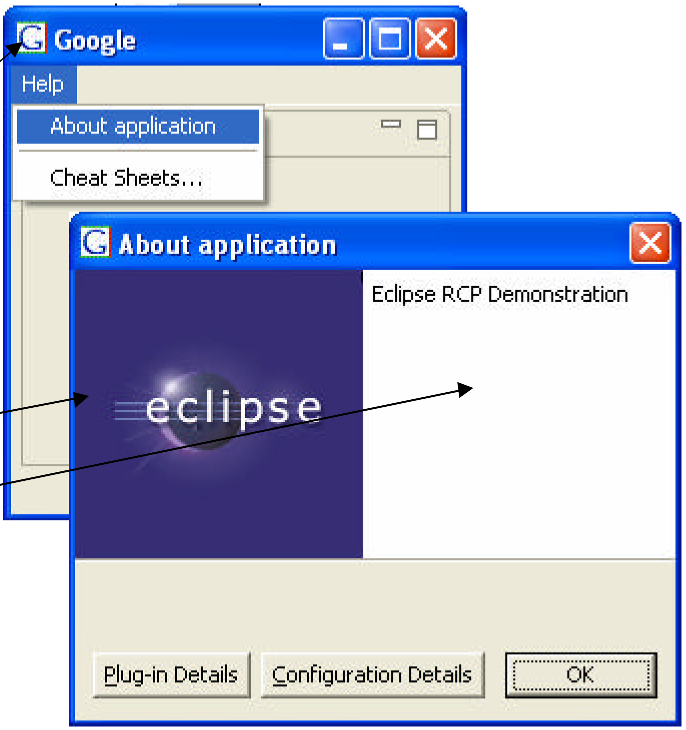
```
…
   protected void fillMenuBar(IMenuManager menuBar) {
        IMenuManager help = new MenuManager("Help“,
                IWorkbenchActionConstants.M_HELP);
        help.add(aboutAction);
        menuBar.add(help);
   }
}
```

Application Branding

Adding Branding to the Workbench

- The **`org.eclipse.core.runtime.products`** extension point is used to add specific branding to the workbench

```
<extension id="product"
   point="org.eclipse.core.runtime.products">
   <product name="application"
      application="com.intalgent…">
      <property name="appName"
         value="Google"/>
      <property name="windowImage"
         value="google.gif"/>
      <property name="aboutImage"
         value="logo.gif"/>
      <property name="aboutText"
         value="Eclipse RCP Demonstration"/>
   </product>
</extension>
```



Eclipse RCP Demonstration

Agenda

Overview of the Eclipse Rich Client Platform (RCP) and Its Place in the Development Landscape

Introduction to the Basic Steps to Create an RCP Application

Exploration of the Generic Workbench and the Core User Interface Components

Overview of Perspectives

Exploring the Role of Perspectives

- Perspectives within the workbench are a visual container for all opened views and editors
- It's important to take into account the following design considerations:
  - The workbench only displays a single perspective at a time, so it's important to group logical and functional areas
  - A view/editor cannot be shared between different perspectives

Creating a Basic Perspective

Exploring the Steps to Create a Perspective

Creating a perspective is a two-step process:

- The plug-in manifest needs to be modified to include a new extension that uses the **`org.eclipse.ui.perspectives`** extension point
- Using the attributes from the new extension point, a perspective class needs to be created

```
public class Perspective implements IPerspectiveFactory {

        public static final String ID =
                "com.intalgent.google.perspective";

        public void createInitialLayout(IPageLayout layout) {
                layout.setEditorAreaVisible(false);
        }
}
```

Perspective Controls

Exploring the Mechanism to Control Perspectives

- Methods to manipulate perspectives:
  - `IWorkbench.showPerspective()`
  - `IWorkbench.openWorkbenchWindow()`
  - `IWorkbenchPage.setPerspective()`
  - `IWorkbenchWindow.close()`
  - `IWorkbenchPage.savePerspective()`
- Methods to locate and access perspectives:
  - `IWorkbenchPage.getPerspective()`
  - `IWorkbenchPage.getOpenPerspectives()`

Overview of Views

Exploring the Role of Views

- Views within the workbench are visual containers that allow users to display or navigate resources of a particular type
- It's important to take into account the following design considerations:
  - A view's responsibility is to display data from your domain model, it's important to group similar types of objects into the view
  - The number of views that any application will have is largely dependent on the application's size and complexity

Creating a Basic View

Exploring the Steps to Create a View

Creating a view is a two-step process:

- The plug-in manifest needs to be modified to include a new extension that uses the **`org.eclipse.ui.views`** extension point
- Using the attributes from the new extension point, a view class needs to be created

View Controls

Exploring the Mechanisms to Control Views

- Methods to manipulate views:
  - **`IWorkbenchPage.showView()`**
  - **`IWorkbenchPage.hideView()`**
  - **`IWorkbenchPage.bringToTop()`**
  - **`ViewPart.saveState()`**
  - **`ViewPart.restoreState()`**
- Methods to locate and access views:
  - **`IWorkbench.getViewRegistry()`**
  - **`IWorkbenchPage.getViewReferences()`**
  - **`IWorkbenchPage.findViewReference()`**
  - **`IWorkbenchPage.findView()`**
  - **`IWorkbenchPage.getActivePart()`**
  - **`IWorkbenchPage.getActivePartReference()`**
  - **`IWorkbenchPartSite.getId()`**
  - **`IViewSite.getSecondaryId()`**

Overview of Actions

Exploring the Role of Actions

- Actions within the workbench are commands that can be triggered by the user of an application
- In general, actions fall into three distinct categories:
  - Buttons
  - Items within the tool bar
  - Items within the menu bar

Creating a Basic Action

Exploring the Steps to Create an Action

Creating an action is a two-step process:

- The plug-in manifest needs to be modified to include a new extension that uses the **`org.eclipse.ui.viewActions`** extension point
- Using the attributes from the new extension point, an action class needs to be created

Action Controls

Exploring the Mechanisms to Control Actions

- Methods to manipulate actions:
  - **`IWorkbenchPage.closeAllEditors()`**
  - **`IWorkbenchPage.closeEditor()`**
  - **`IWorkbenchPage.closeEditors()`**
  - **`IWorkbenchPage.bringToTop()`**
  - **`IWorkbenchPage.openEditor()`**
- Methods to locate and access actions:
  - **`IWorkbenchPage.getActiveEditor()`**
  - **`IWorkbenchPage.getDirtyEditors()`**
  - **`IWorkbenchPage.getEditorReferences()`**
  - **`IWorkbenchPage.getDirtyEditors()`**

Java WebStart

Deploying RCP Applications via Java WebStart

- Eclipse 3.1 introduced the ability to deploy RCP application via Java WebStart (JWS)
- Deploying an RCP application via JWS is a three-step process:
  - Package each plug-in in a separate JAR file
  - Since the application requires full permissions, each JAR file must be signed
  - Package all the JAR files into single WAR file
- Your JNLP file need to use the class: **`org.eclipse.core.launcher.WebStartMain`**

DEMO

Exploring the Google Application

Summary

- The Eclipse Rich Client Platform delivers developers a polished framework for creating elegant, cross-platform applications
- As the development community begins to understand and utilize the RCP within their own applications, it’s going to be exciting to see how this platform is extended and evolves

More Information

- Download the companion source code package for the demonstrated RCP application
  http://www.intalgent.com/javaone/
- Eclipse Rich Client Platform Overview
  http://www.eclipse.org/rcp/
- EclipseZone
  http://www.eclipsezone.com

References

Resources, Articles, and Presentations Referenced

- Eclipse 3.1 Help Contents
- EclipseCon 2005 Presentation: Eclipse RCP (J. McAffer and N. Edgar)
- EclipseCon 2005 Presentation: Developing for the Rich Client Platform (N. Edgar and Pascal Rapicault)
- EclipseCon 2005 Presentation: Packaging, Deploying and Running Rich Client (E. Burnette)
- Eclipse.org Article: Using Perspectives in the Eclipse UI (D. Springgay)
- Eclipse.org Article: Creating an Eclipse View (D. Springgay)

Q&A

Introduction to Eclipse's Rich Client Platform

Jeff Gunther
President
Intalgent
www.intalgent.com

Session TS-5979