Developing for the Rich Client Platform

Turning your application in to a “Really Cool Product”

Nick Edgar
Pascal Rapicault

IBM Rational Software
Ottawa

Eclipse Platform Committers
To contact us: news://news.eclipse.org/eclipse.platform.rcp
Your Infrastructure

- To get the most from this tutorial, you need to bring a laptop capable of running Eclipse and developing plug-ins.

- Before the tutorial, you need to have the following software loaded on your machine:
  - Eclipse SDK 3.1M5  http://www.eclipse.org/downloads
  - Eclipse RCP SDK 3.1M5
  - Eclipse RCP Delta Pack 3.1 M5
  - A Jabber client  http://www.jabber.org

- Eclipse should be installed and running.
- The RCP SDK and RCP Delta pack should be kept as zip.
- Your jabber client should be installed and running with an account.
What is eclipse RCP?

- “Really Cool Platform”?
- “Rich Client Platform”
  A Platform for building Client applications with Rich functionality

History
- Hackers in the eclipse community started building non-IDE apps on 2.1
- In 3.0 we made this real by factoring out the IDE aspects from the workbench
Where can you encounter Eclipse RCP

IBM Lotus Workplace

Eclipse Trader

Reference
- EclipseCon’05: Eclipse RCP Everywhere (J.-M. Lemieux, J. McAffer)
RCP by the numbers (as of 3.1 M5)

- Download size: ~ 5 Mb
- Unzipped size: ~ 6 Mb

- VM size of a Java hello world (not eclipse-based): ~ 8 Mb
- VM size of an headless eclipse-based hello world: ~ 9 Mb

- Time before your application code runs: ~ 650ms

- Number of plug-ins in the RCP binary: 10
What is in RCP?

IDE personality

Generic Workbench

Resources

JFace

Runtime

SWT

OSGi

Help (optional)  Update (optional)  Text (optional)  IDE Text  Compare  Debug  Search  Team/CVS
Outline

- **Section I – The basic concepts**
  - Eclipse application
  - RCP application
  - Workbench advisors
  - Simple branding

- **Section II – The workbench specifics**
  - Window / page
  - Perspective
  - View / Editors
  - Actions, key bindings
  - Communication across parts
  - Access to the application model

- **Section III – Extra RCP plug-ins, advanced topics**
  - Help
  - Update / provisioning
  - Advanced branding
  - Third party libraries
  - User Authentication
Section I – The basic concepts
What is an eclipse application?

1. An extension to the `org.eclipse.core.runtime.applications` extension point.

   ```xml
   <extension id="application"
     point="org.eclipse.core.runtime.applications">  
     <application>
       <run class="hyperbola.Hyperbola"/>
     </application>
   </extension>
   ```

2. A class implementing `IPlatformRunnable`. It is the main.

   ```java
   public class Hyperbola implements IPlatformRunnable {
     public Object run(Object args) throws Exception {
       System.out.println("Hello");
     }
   }
   ```
The application name is composed of the extension id prefixed by the plug-in id.
My first RCP application

⇒ An RCP application is an eclipse application where the workbench runs the main event loop.

Overall life cycle of the application

```java
public class Hyperbola implements IPlatformRunnable {
    public Object run(Object args) throws Exception {
        try {
            Display d = PlatformUI.createDisplay();
            WorkbenchAdvisor wa = new MinimalAdvisor();
            PlatformUI.createAndRunWorkbench(d, wa);
            return new Integer(0);
        } finally {
            if (d!=null)
                d.dispose();
        }
    }
}
```

- Create the display
- Create a workbench advisor
- Run the workbench
- Application return code
- Dispose the display
Configuring the Workbench and its Windows

- `WorkbenchAdvisor`, `WorkbenchWindowAdvisor`, `ActionBarAdvisor`
  - *strategy* objects to configure the workbench and its windows
  - provide hook methods called at strategic points during the workbench life cycle
The `WorkbenchAdvisor` and its configurer

⇒ Advises on how the workbench should be configured

```java
public class HyperbolaWorkbenchAdvisor extends WorkbenchAdvisor {

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

    public String getInitialWindowPerspectiveId() {
        return HyperbolaPerspective.ID;
    }

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

Enable the workbench state save mechanism

Defines the initial perspective

Factory method for the window advisor
The `WorkbenchWindowAdvisor` and its configurer

⇒ Advises on how the window should be configured

- **Title**
  - `IWorkbenchWindowConfigurer.setTitle()`

- **Cool bar**
  - `IWorkbenchWindowConfigurer setShowCoolBar()`

- **Menu bar**
  - `IWorkbenchWindowConfigurer setShowMenuBar()`

- **Status bar**
  - `IWorkbenchWindowConfigurer setShowStatusLine`
  - `IWorkbenchWindowConfigurer setShowPerspectiveBar()`
  - `IWorkbenchWindowConfigurer setShowFastViewBars()`
  - `IWorkbenchWindowConfigurer setShowProgressIndicator()`

**Note** Not shown here:
- `IWorkbenchWindowConfigurer.setShellSize()`
- `IWorkbenchWindowConfigurer.setShellStyle()`
The `WorkbenchWindowAdvisor` and its configurer (sample)

```java
public HyperbolaWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

    HyperbolaWorkbenchWindowAdvisor(IWorkbenchWindowConfigurer configurer) {
        super(configurer);
    }

    public void preWindowOpen() {
        IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
        configurer.setTitle("Hyperbola");
        configurer.setInitialSize(new Point(275, 475));
        configurer.setShowProgressIndicator(true);
        configurer.setShowPerspectiveBar(false);
    }

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

Layout and appearance settings

Factory method for the action bar advisor
The ActionBarAdvisor and its configurer

 CreateUser actions and use them in the various bars

---

Creation of actions
makeActions()

Cool bar
fillCoolBar()
ICoolBarManager

Menu bar
fillMenuBar()
IMenuManager

Status bar
fillStatusBar()
IStatusLineManager
The ActionBarAdvisor and its configurer (sample)

```java
public class HyperbolaActionBarAdvisor extends ActionBarAdvisor {
    private IWorkbenchAction quitAction;

    public HyperbolaActionBarAdvisor(IActionBarConfigurer configurer) {
        super(configurer);
    }

    protected void makeActions(IWorkbenchWindow window) {
        quitAction = ActionFactory.QUIT.create(window);
        register(quitAction);
    }

    protected void fillMenuBar(IMenuManager menuBar) {
        IMenuManager fileMenu = new MenuManager("&Hyperbola", IWorkbenchActionConstants.M_FILE);
        menuBar.add(fileMenu);
        fileMenu.add(quitAction);
    }

    ...
```
Structure of the advisors and configurers

Regular plug-in API

IWorkbench

IWorkbenchWindow

IActionBarConfigurer

IWorkbenchConfigurer

HyperbolaWorkbenchAdvisor

HyperbolaWorkbenchWindowAdvisor

HyperbolaActionBarAdvisor

Privileged API

Workbench side

Client side
Workbench and advisors life cycle (partial)

The shell is available as of `postWindowCreate()`
Branding – the basics

Branding consists in giving your product the colors of your company

Extension-point: org.eclipse.core.runtime.products

```
<extension id="hyperbolaProduct" point="org...products">
  <product name="Hyperbola">
    <application="org.eclipse.rcp.hyperbola.application">
      <property name="windowImage" value="online.gif"/>
    </application>
  </product>
</extension>
```

Default application to execute
Branding – the basics (cont’d)

About dialog

```xml
<property name="aboutImage"
  value="image.gif" />

<property name="aboutText"
  value="Hyperbola Chat client" />
```

Default preferences (in a Java property file)

```xml
<property name="preferenceCustomization"
  value="plugin_customization.ini" />
```

In `plugin_customization.ini` file:

```
org.eclipse.ui/SHOW_TRADITIONAL_STYLE_TABS=false
org.eclipse.ui/SHOW_TEXT_ON_PERSPECTIVE_BAR=false
```

References

- Product editor in Eclipse 3.1
- Eclipse.org article: Branding your application (A. Eisdness and P. Rapicault)
Producing an RCP application – the product editor
Layout of an RCP application in the file system

- **hyperbola/**
  - **hyperbola.exe**
  - **hyperbola.ini**

- **configuration/**
  - **config.ini**

- **plugins/**
  - `<plug-ins from the rcp base>`
  - `<hyperbola plug-ins>`

- **jre/**

- **Branded eclipse.exe and its optional partner file allowing to set VM and VM arguments**

- **Controls the product to run, the bundles to run, the splash screen, and other aspect of the platform**

- **Optional JRE**

- **Your plug-ins and their prerequisites**
Alternate layout of an RCP application in the file system

- `hyperbol a/`
  - `hyperbol a. exe`
  - `hyperbol a. ini`
- `configuration/`
  - `config. ini`
- `hyperbol a- content/`
  - `eclipse/`
    - `plugins/`
      - `<hyperbol a. plug-ins>`
  - `links/`
    - `hyperbol a. link`
- `plugins/`
  - `<plug-ins from the RCP base>`
- `jre/`

This layout allows for multiple plug-in folders

- **Your plug-ins**
- **Link file whose content refers to the root of the folders containing an eclipse folder**
- **RCP plug-ins**
The config.ini

- Java properties file read on startup and merged with the system properties
- Controls many aspects of the platform (locations, start level, ...)
- Used in the branding to declare the splash screen and the product to run

```ini
# The product to run
eclipse.product = org.eclipsercp.hyperbola.product

# The splash screen to display
osgi.splashPath = platform:/base/plugins/org.eclipsercp.hyperbola

# The list of bundles to run
osgi.bundles = org.eclipse.core.runtime@2:start, org.eclipse.ui, org.eclipse.swt, org.eclipse.swt.win32, ...
```
Exercise

- Fill in the main of the application

- Modify the advisors
  - Hook actions such as File / Exit, Help / About
  - Show the toolbar and the status bar

- Brand your product

- Export your product
Exercise – Self-hosting concepts

**Workspace**

**Target** (Preferences > PDE > Target platform)

```
d:/RCPSDK/
  eclipse/
    plugins/
      <plugs-ins from the rcp base>
      <plugs-ins from the rcp delta pack>
```

=

**Your eclipse application**
*(runtime-workbench)*

**Note**
- This can be further controlled using the plug-ins tab in the launch configuration
Section II – The workbench specifics
Workbench structure

Workbench (1)
  Window (0 – N)
  Page (0 – 1)
  Perspective (0 – N), 1 active
  View (0 – N)
  Editor (0 – N)
A window contains a page, which contains an arrangement of views and editors whose layout is defined by a perspective.

**Appearance**

- `IWorkbenchWindowConfigurer`

**Controlling (opening/closing)**

- `IWorkbench.openWorkbenchWindow()`
- `IWorkbenchWindow.close()`

**Introspection / tracking**

- `org.eclipse.ui.IWorkbench.getActiveWorkbenchWindow()`
- `org.eclipse.ui.IWorkbench.getWorkbenchWindows()`
- `org.eclipse.ui.IWindowListener.addWindowListener()`
- `org.eclipse.ui.IPageListener.addPageListener()`
- `org.eclipse.ui.IPageService.getActivePage()`
- `org.eclipse.ui.IWorkbenchWindow.getActivePage()`
Perspective

⇒ Controls the layout of views and editors

- Defined programmatically in the perspective factory (IPerspectiveFactory)

```java
public void createInitialLayout(IPageLayout layout) {
    String editorArea = layout.getEditorArea();
    layout.setEditorAreaVisible(false);
    IFolderLayout left = layout.createFolder("left", IPageLayout.LEFT,
                                            0.25f, editorArea);
    left.addView(MyView.VIEW_ID);
}
```

- Perspective extensions (org.eclipse.ui.perspectiveExtensions)
  - Allow plug-ins to extend perspectives
Perspective – control and lifecycle

Controlling

I Workbench.showPerspective()
I Workbench.openWorkbenchWindow()
I WorkbenchPage.setPerspective()
I WorkbenchWindow.close()

Saving

I WorkbenchPage.savePerspective()

Introspection / tracking

I WorkbenchPage.getPerspective()
I WorkbenchPage.getOpenPerspectives()
org.eclipse.ui.IPerspectiveListener3

Reference

- Eclipse.org article: Using Perspectives in the Eclipse UI (D. Springgay)
Views

⇒ A part of the workbench page outside of the editor area

Typical roles for views:
- Allow overall navigation of the application’s model (e.g. the Package Explorer)
- Support navigation in the editor (e.g. Outline)
- Provide additional context-sensitive info (e.g. Properties)
Views – standalone

⇒ A view that can’t be docked with others

Declared in the perspective factory

```java
layout.addStandaloneView(HelloRCPView_ID, false, IPageLayout.TOP, .95f, layout.getEditorArea());
```

Id of the view
Flag to show the title bar
Views – non-moveable

Declared in the perspective factory

```java
public void createInitialLayout(IPageLayout layout) {

    layout.addView(View1.ID_VIEW1, IPageLayout.TOP,
            IPageLayout.RATIO_MAX,
            IPageLayout.ID_EDITOR_AREA);

    IViewLayout viewLayout = layout.getViewLayout(View1.ID_VIEW1);

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

    layout.addView(View1.ID_VIEW1, IPageLayout.TOP, IPageLayout.RATIO_MAX, IPageLayout.ID_EDITOR_AREA);

    IViewLayout viewLayout = layout.getViewLayout(View1.ID_VIEW1);

    viewLayout.setCloseable(false);

}

Declared in the perspective factory
Views – multiple instance

In the plugin.xml

```xml
<extension point="org.eclipse.ui.views">
  <view allowMultiple="true" class="demoViews.View1" name="Non closable view" id="DemoViews.view1"/>
</extension>
```

When opening the view

```java
window.getActivePage().showView(View1.ID_VIEW1, var, IWorkbenchPage.VIEW_VISIBLE);
```

The secondary id of the view
Views – sticky

⇒ A view that stays open across perspective switches

In the plugin.xml

```xml
<stickyView id="id.myview"
    location="RIGHT"
    closeable="true"
    moveable="false"/>
```
Views – positioning (single instance case)

⇒ Controlling the position of the view when it opens

Declared in the perspective factory

```java
IFolderLayout folder = layout.createFolder("demoViews", IPageLayout.TOP, 0.5f, layout.getEditorArea());
folder.addPlaceholder("DemoViews.view1");
```

ID of the views that will be opened in this folder
Views – positioning (multiple instance case)

⇒ Controlling the position of the view when it opens

Declared in the perspective factory

```java
IFolderLayout folder = layout.createFolder("demoViews", IPageLayout.TOP,
0.5f, layout.getEditorArea());

folder.addPlaceholder("DemoViews.view1:*");
```

ID of the views that will be opened in this folder

Pattern matching the secondary ID
Views – title bar

- Content of the title bar – (org.eclipse.ui.part.WorkbenchPart – ViewPart)
  
  ```java
  setPartName()
  setTitleToolTip()
  setTitleImage()
  setContentDescription()
  ```

- Indicate things are happening in the view – only through jobs
  
  ```java
  getSite().getAdapter(IWorkbenchSiteProgressService.class)
  ```

**Notes**

- To hide the title bar, use a standalone view.
- To have a title bar that goes across the whole view, use a standalone view.
Views – control and lifecycle

Controlling

- IWorkbenchPage.showView()
- IWorkbenchPage.hideView()
- IWorkbenchPage.bringToTop()

Tracking

- org.eclipse.ui.IPartListener2
- org.eclipse.ui.IPartService

Introspection

- IWorkbench.getViewRegistry()
- IWorkbenchPage.getViewReferences()
- IWorkbenchPage.findViewReference()
- IWorkbenchPage.findView()
- IWorkbenchPage.getActivePart()
- IWorkbenchPage.getActivePartReference()
- IViewSite.getId()
- IViewSite.getSecondaryId()

Saving

- To save the widgets state
  - ViewPart.saveState()
  - ViewPart.restoreState()

- To save the underlying model
  - ISaveablePart

Reference

- Eclipse.org article: Creating an Eclipse View (D. Springgay)
Editors

Files not required !!!

References
- API: `org.eclipse.ui.part.EditorPart` and `org.eclipse.ui.IEditorInput`
- Article: http://www.jroller.com/page/Zhou/20040215#eclipse_editors_not_tied_to
Editors – title bar

- Content of the title bar – (org.eclipse.ui.part.WorkbenchPart - EditorPart)

  - setPartName()
  - setTitleToolTip()
  - setTitleImage()
  - setContentDescription()

- Title bar spanning the whole editor area
  
  See bug 84706
Editors – control & lifecycle

Controlling

- `IWorkbenchPage.closeAllEditors()`
- `IWorkbenchPage.closeEditor()`
- `IWorkbenchPage.closeEditors()`
- `IWorkbenchPage.bringToTop()`
- `IWorkbenchPage.openEditor()`

Introspection

- `IWorkbenchPage.getActiveEditor()`
- `IWorkbenchPage.getDirtyEditors()`
- `IWorkbenchPage.getEditorReferences()`
- `IWorkbenchPage.getDirtyEditors()`
- `IEditorReference.isDirty()`
- `IWorkbenchPartSite.getId()`

Tracking

- `org.eclipse.ui.IPartListener2`
Editors – misc.

- Saving / restoring editors state
  - Editor input should be persistable (implement `IPersistableElement`)
  - Restoration of the content should be lazy and the editor input should be lightweight

- Hiding/showing the editor area after the fact

  ```java
  IWorkbenchPage.setEditorAreaVisible(boolean)
  ```

- Misc.
  - `EditorInput` must implement `equals()` and `hashCode()`

References
- Plug-ins: org.eclipse.ui.workbench.texteditors and its prerequisites
- Example: http://www.eclipse.org/rcp/main.html#text_editor_example
- Other plugins: org.eclipse.ui.forms, org.eclipse.gef
Editors or Views

<table>
<thead>
<tr>
<th></th>
<th>Editors</th>
<th>Views</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Position</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary area</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Around primary area</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td><strong>Open/save life cycle</strong></td>
<td>Yes</td>
<td>No*</td>
</tr>
<tr>
<td><strong>Input object</strong></td>
<td>Yes</td>
<td>No*</td>
</tr>
<tr>
<td><strong>Contribute to main menu</strong></td>
<td>Yes</td>
<td>No*</td>
</tr>
<tr>
<td><strong>Rearrangeable</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*: It is an out-of-the-box no, however see the section on views for solutions

Reference
Actions

⇒ An action abstracts the differences between a menu item and a toolbar item

Makes it easier to reorganize the UI

- Has behavior: `run()`
- Describes its own enablement (enabled property)
- Describes its own presentation attributes (text, image, tooltip)
Actions (sample)

class AddContactAction extends Action {

    public AddContactAction(Session s) {
        setId("hyperbola.action.addContact");
        setText("Add &Contact...");
        setEnabled(s.isConnected());
        // After that you want to track the connection state
    }

    public void run() {
        // Prompt for contact details
    }
}
Retargetable actions

⇒ It allows the current part to provide the behavior (handler) for an action added elsewhere (e.g. cut, copy, paste).

```java
public static final ActionFactory ADD_CONTACT =
    new ActionFactory("hyperbola.action.chat"){

    public IWorkbenchAction create(IWorkbenchWindow window) {
        RetargetAction action = new RetargetAction(getId());
        action.setText("Add &Contact...");
        action.setToolTipText("Add Contact...");
        action.setImageDescriptor(...);
        window.getPartService().addPartListener(action);
        action.setActionDefinitionId("hyperbola.command.addContact")
        return action;
    }
};
```

**Note**
- Text and tool tip can be overridden by the handler if the action implements LabelRetargetAction.
Retargetable actions (hooking a handler from a part)

class RosterView extends ViewPart {
    IAction addContactAction;

    void hookActions() {
        addContactAction = new AddContactAction(getSession());
        getViewSite().getActionBars().setGlobalActionHandler("hyperbola.action.chat", addContactAction);
    }
}
Declarative actions

- They add onto menus and toolbars previously defined by the application, editors and views

- Extension points: org.eclipse.ui.actionSets, editorActions, viewActions, popMenus

- Give flexibility for progressive exposure to functionality.

- Can be associated with perspectives and parts, filtered by activities, etc.

- Code only run when action selected. Enablement rules can be defined declaratively. Allows lazy plug-in activation.

- Restriction: they are not designed to build on each other.

Note
- The new commands support in 3.1 relaxes this restriction
Declarative actions (sample)

```xml
<extension point="org.eclipse.ui.actionSets">
  <actionSet
    label="Contacts"
    id="org.eclipsesrc.cp.hyperbola.contactsActionSet">
    <menu
      id="hyperbola.menu.contacts"
      label="&Contacts"
      path="additions"/>
    <action
      id="hyperbola.action.addContact"
      definitionId="hyperbola.command.addContact"
      label="Add &Contact"
      icon="icons/add_contact.gif"
      retarget="true"
      menubarPath="hyperbola.menu.contacts/additions"
      toolbarPath="mainToolbar/contacts"/>
  </actionSet>
</extension>
```
Actions, programmatically or declaratively?

- Define a minimal skeleton programmatically in the advisor.
  - Include only the menus and actions that will always be visible.

- Define action sets for other contributions.

- For views and editors, define their actions programmatically, but allow for extension.

- When views need to add to main menus and toolbars, use
  org.eclipse.ui.actionSets and
  org.eclipse.ui.actionSetPartAssociations.

**Note**
- Action sets can enable progressive exposure and enhance UI stability due to the flexibility of perspective- and part-associations.

**Reference**
- Eclipse.org article: Contributing Actions to the Eclipse Workbench (S. Arsenault)
Key bindings

Key bindings associate a key sequence with a command.

- To associate an action with a command, use

  ```java
  IAction.setActionDefinitionId(String commandId)
  ```

- Actions added in code must be registered with the key binding service.

  ```java
  addContactAction = JabberActionFactory.ADD_CONTACT.create(window);
  register(addContactAction);
  ```

- Registration is done automatically for actions added declaratively.
Key bindings (sample)

```xml
<extension point="org.eclipse.ui.commands">
  
  <category
    name="Hyperbola"
    description="Hyperbola commands"
    id="hyperbola.commands" />

  <command
    name="Add a contact"
    description="Add a contact to your roster"
    categoryId="hyperbola.commands"
    id="hyperbola.command.addContact" />

  <keyConfiguration
    name="Hyperbola Key Configuration"
    description="Hyperbola Key Configuration"
    id="hyperbola.commands.keyConfiguration" />

  <keyBinding
    commandId="hyperbola.command.addContact"
    keySequence="CTRL+N"
    keyConfigurationId="hyperbola.commands.keyConfiguration" />

</extension>
```
Communication across parts

Direct communication

- **View to view**
  - Use the APIs to open and hide view
- **View to editor**
  - Use the APIs to open editor with the appropriate input

Tracking the selection

- **To publish:**
  - `IWorkbenchSite.setSelectionProvider(ISelectionProvider)`
  - `IWorkbenchPart.getSite()`
- **To subscribe:**
  - `ISelectionService` and `ISelectionListener`
  - `IWorkbenchPage extends ISelectionService`
  - `IWorkbenchWindow.getSelectionService()`

Tracking part lifecycle and activation

- **To subscribe:**
  - `IPartService` and `IPartListener[2]`
  - `IWorkbenchPage extends IPartService`
  - `IWorkbenchWindow.getPartService()`

Tracking window lifecycle and activation

- **To subscribe:**
  - `IWindowListener`
  - `IWorkbench.addWindowListener(IWindowListener)`
Accessing the application Model

From Views

- When the view is opened programmatically
  - `setInput(((HelloView)showView(...)).setInput(...))`

- When the view is opened by the user
  - Access the model in `createControlPart()`
    (i.e.: get the current selection)
  - For views with multiple instances: keep a map of secondary id to input

From an Action

- Track the current selection
- Take an instance from the model in the constructor

From Editors

- Give input when opening the editor
  - `openEditor(page.openEditor(myInput, "hyperbola.editor"))`

Static Fields

- For example the resources plug-in:
  `ResourcesPlugin.getWorkspace()`
Saving / restoring the application state

You must save your model yourself!

- Workbench
  - Need to explicitly enable workbench state persistence (see `IWorkbenchConfigurer`)
  - Preferences should be saved explicitly on change
  - Editors and views, see the appropriate sections

Be Lazy
Locations – where can I store my data?

**Instance**  
Application data (Platform.getInstanceLocation())  
• Writeable, controlled by the application.  
• osgi.instance.area

**Configuration**  
Configuration being run (Platform.getConfigurationLocation())  
• Writeable. May have a parent location.  
• osgi.configuration.area  
• osgi.sharedConfiguration.area

**User**  
Data specific to a user (Platform.getUserLocation())  
• osgi.user.area

**Special values**

@none, @noDefault, @user.home, @user.dir

**Notes**

- The eclipse install folder is accessible from Platform.getInstallLocation() and is read only. The associated system property is osgi.install.area.
- Information read-only and lock from the Location objects are advisory only.
- Default values can be specified in the config.ini
Exercise

- Create a login prompt
  - Analyze command line arguments to skip authentication
  - Set some locations

- Create a “chat” action

- Make the roster view non-closable, non-multiple, without title

- Modify the status bar show the server and the number of buddies online (ui / model integration)

- When a user is double-clicked in the roster view, open the user property dialog (communication across parts)

- In the roster view, highlight the user whose chat view is active (communication across parts)
Section III – Extra RCP plug-ins, advanced topics and other cool stuff

1. Helping your user
2. Provisioning and deploying
3. Advanced / extreme branding
4. Using third party libraries
5. User authentication
1. Helping your users
Eclipse Help infrastructure – 101

- HTML and XML based system
- Dynamic content generation
- Contextual help
- Search engine

Help UI
User interface and dialogs

Help Core
API to access the documents

---

### plugin.xml

```xml
<extension point="org.eclipse.help.toc">
  <toc primary="true">
    <toc file="doc/wizard.html"/>
    <toc file="doc/tipsAndTricks.xml"/>
  </toc>
</extension>
```

### guide.xml

```xml
<toc label="Hyperbola User Guide">
  <topic label="Getting Started">
    <anchor id="gettingstarted"/>
  </topic>
  <topic label="Commands">
    <href="doccmds.html"/>
  </topic>
</toc>
```
Eclipse Help infrastructure – UI

Building a help plug-in

In this example, we assume that a documentation author has already supplied you with the raw documentation in the form of HTML files. The granularity and structure of these files is completely up to the documentation team. Once the documents are ready, the plug-in and topics can be done in Eclipse.

We start by assuming that the documentation is provided in the following tree:

```
html/
  concepts/
    concept1.html
    concept1_1.html
    concept1_2.html
  tasks/
    task1.html
    task2.html
    task3_1.html
    task3_2.html
  ref/
    ref1.html
    ref2.html
```

Required plug-ins

- org.apache.ant
- org.apache.lucene
- org.eclipse.help.appserver
- org.eclipse.help.base
- org.eclipse.help.ui
- org.eclipse.help.webapp
- org.eclipse.tomcat
Eclipse Help infrastructure – Core API

- Lightweight API

- Define a hierarchical table of contents, giving access to the help topics and their content

- Define content for context-sensitive help

- Plug-in: `org.eclipse.help`
  - See the class `org.eclipse.help.HelpSystem`

Reference

- Help: Plugging in Help
2. Provisioning and deploying
Updating your application

Feature-based

- Update UI
  Reusable dialogs

Bundle-based

- Update Core
  APIs to programmatically update

- OSGi API
  Lowest level API to install/update plug-ins
An update site is an URL where you can find a site.xml

- No special server is required!
- Update site delivers “features”
- A Feature references plug-ins it manages

**feature.xml**

```
<feature id="hyperbolafeature"
  label="hyperbola-feature"
  version="1.0.0">
  ...
  <plugin id="org.eclipsercp.hyperbola"
    download-size="0"
    install-size="0"
    version="1.0.0.qualifier"/>
  ...
</feature>
```

**typical layout of an update site**

```
site.xml
features/
  hyperbolafeature.jar
plugins/
  org.eclipsercp.hyperbola.jar
  org.eclipsercp.hyperbola.ui.jar
```

**Reference**

- EclipseCon’05: Creating, Packaging, [...] Features in Eclipse 3.0 (S. Minocha, P. McCarthy)
Updating your application – Update UI

Required plug-ins
org.eclipse.update.ui and its prerequisites

Reference
- Eclipse.org article: How to Keep Up To Date (D. Glozic and D. Birsan)
Updating your application – the update core API

- **Plugin:** org.eclipse.update.core
  - Package: org.eclipse.update.operation

### Installing a feature

```java
InstallCommand cmd = new InstallCommand("myFeature", "1.0.0",
                                          http://myCompany/product, null, "false");

cmd.run(new NullProgressMonitor());

cmd.applyChangesNow();
```

---

**Reference**

Updating your application – the OSGi API

```java
// to install
Bundle[] toRefresh = new Bundle[1];
toRefresh[0] = context.installBundle("reference:file:d:/aBundle");
refreshPackages(toRefresh);

// to uninstall
toRefresh[0].uninstall();
```

To predict the state of the system (`org.eclipse.osgi.service.resolver`) See PlatformAdmin service, more precisely the State class

**Note**
- When using this API, it is recommended to download the files manually and use a reference: URL, otherwise OSGi copies the jars into the configuration area.

**Reference**
Java WebStart

- Eclipse 3.1 can be Java WebStarted
  - It requires full permission on the client

- Preparing for Java WebStart
  - Jar up your plug-ins
  - Sign your plug-ins

- The main is `org.eclipse.core.launcher.WebStartMain`

- Values from the `config.ini` goes in the JNLP file except:
  - `osgi.splashPath` use the JNLP mechanism
  - `osgi.bundles` not necessary. Implicit by the jars listed in the JNLP file

Reference
- EclipseCon’05: Packaging, Deploying and Running Rich Client (E. Burnette)
3. Advanced / extreme branding
Intro support

- It provides the “welcome page” for a product.

- HTML / CSS or SWT / Forms based

- It supports two modes:
  - Full mode
  - Standby mode

- Can run actions to drive the UI

Reference
- Eclipse.org article (to be published): Introducing Intro: How to create Welcome pages [...] (M. Faraj)
Themes

⇒ General mechanism provided by the workbench to help plug-ins provide uniform look and feel to their components

- A theme has an ID
- Declare categorized colors, fonts, and other data items (identified with IDs)
- Assign specific values to these within a named theme
- To access the values use JFace color and font registries and track the change using change listeners

- Workbench uses this for its own colors
- Other parts (views and editors) can do so as well

To set a theme:

```java
IWorkbench.getThemeManager().setCurrentTheme();
```

bug 84738
The presentation API

-> The presentation controls the appearance of editors, views and other components in the workbench.

- Relation themes / presentations
  - A presentation should define its colors and fonts in a theme category
  - A theme category can be bound to a presentation

Reference
- EclipseCon’05: Presentations API - the look and feel of Eclipse (J.M. Lemieux, S. Xenos)
- Example: http://www.eclipse.org/rcp/main.html#text_editor_example
- Example: R2.1 presentation (plugin org.eclipse.ui.presentations.r21)
Non rectangular window

Handled by the workbench window advisor

Shell creation and customization

```java
WorkbenchWindowAdvisor.postWindowCreate()
```

Filling the window with toolbar, menu bar, etc...

```java
WorkbenchWindowAdvisor.createWindowContents()
```
4. Using third party libraries
Packaging third party libraries

- **Recommended way:**
  - Create one or many bundles !!!
  - Examples:
    - junit.jar → Junit plugin.
    - Hibernate 2.1.7 → log4j, junit, xalan, xerces, jta, … plugins

- **Less recommended ways**
  - Embed the library in your own plug-in
  - Add the library on the `bootclasspath`
  - Set the parent loader of the framework – and place the library accordingly
    
    ```
    osgi.parentClassloader = {boot | ext | app | fwk }
    ```

**Reference**
- (Potential) PDE support in 3.1 for the recommended way
Frequent problems with third party libraries

- Context classloaders
  - In the library:

    ```java
    Thread.currentThread().getContextClassLoader().getResource()
    ```

- The solution / workaround

  ```java
  Thread current = Thread.currentThread();
  ClassLoader oldLoader = current.getContextClassLoader();
  try {
    current.setContextClassLoader(getClass().getClassLoader());
    // call library code here
  } finally {
    current.setContextClassLoader(oldLoader);
  }
  ```

- Property files
5. User authentication
Authentication – when the application starts

```java
public class Application implements IPlatformRunnable {
    public Object run(Object args) throws Exception {
        try {
            // Do authentication here
            Display d = PlatformUI.createDisplay();
            // Or here if you need to have a display
            WorkbenchAdvisor wa = new MinimalAdvisor();
            PlatformUI.createAndRunWorkbench(d, wa);
            return new Integer(0);
        } finally {
            if (d != null)
                d.dispose();
        }
    }
}
```

Authentication – before the application starts

- The authentication bundle
  - Do the full authentication in the start method
    - This is usually bad practice, but here we need to be synchronous
  - On authentication failure, change the application to run to be an empty one.
  - On authentication success, set the start level to 6

- Other settings
  - Add the authentication bundle on the `osgi.bundles` list and set its start level
    ```
    osgi.bundles = org.eclipse.core.runtime@1:start,
    authentication@2:start
    ```
  - Set the start level of the framework to the value to which the authentication bundle has been set
    ```
    osgi.startlevel = 2
    ```
  - Ensure that the default start level of other bundles is greater

**Note**
- This technique can also be used to do user based provisioning and/or to change the application to run
Turning your RCP application into a domain-specific platform

- Define API to access your model
- Expose extension-points
- Other UI API?
  - Menu bar and toolbar paths
  - View IDs
  - Action IDs
- Build a community

References
- Eclipse.org article: How to Use the Eclipse API? (J. des Rivieres)
- EclipseCon’05 session: API First (J. des Rivieres)
- EclipseCon’04 session: Eclipse APIs: Lines in the Sand (J. des Rivieres)
The end…
F.A.Q.
Productization issue – Accessibility and NL

- Translation:
  - Plugin.xml – plugin.properties
  - Code – regular java techniques
  - NL fragments – to provide after the fact translation
  - org.eclipse.osgi.util.NLS mechanism

- Accessibility:
  - The workbench is already accessible

References
- Eclipse help: Locale specific files
- Eclipse.org article: Designing Accessible Plug-ins in Eclipse (T. Creasey)
Because of Eclipse multiple classloaders, you may have trouble to restore objects from a plug-in that is not in your prerequisites.

To work around that, annotate the output stream:

```java
class EclipseOutputStream extends ObjectOutputStream {
    PackageAdmin pkgAdmin; //initialize it in the constructor

    protected void annotateClass(Class clazz) throws IOException {
        super.annotateClass(clazz);
        Bundle declaringBundle = pkgAdmin.getBundle(clazz);
        String bundleName = null;
        if (declaringBundle != null) {
            bundleName = declaringBundle.getSymbolicName();
        } else {
            bundleName = "bootClasspath";
        }
        this.writeUTF(bundleName);
    }
}
```

Write in the stream the bundle that loaded the class. It will be read by `resolveClass()` of EclipseInputStream.
What does it take to make my plug-ins dynamic?

- **Good practices**
  - Keep your objects for yourself and do not offer API for other plug-ins to store their objects in your plug-in
  - Track extension and extension-points
  - Clean-up after yourself (unregister listeners, free OS resources)

- **Runtime facilities:**
  - `org.eclipse.core.runtime.dynamic.helpers`
  - `org.eclipse.core.runtime.IRegistryChangeListeners`
  - `org.eclipse.core.runtime.InvalidRegistryObjectException`
  - Dynamic capabilities markup in the manifest.mf

- **OSGi facilities**
  - Services, declarative services

**References**
- Platform Core webpage: Markup to express bundle dynamic capabilities.
How do I do a system tray icon / item?

http://dev.eclipse.org/viewcvs/index.cgi/%7Echeckout%7E/platform.swt-home/dev.html
Can I reuse the resources plug-in?

- Yes – however it only provides you with the concepts of workspace, projects, markers, and it is bound to the file system.

- The resources navigator is **not** in the resources plug-in.

- Some other interesting views are in the IDE.
How do I access a file in my plug-in?

- In the given bundle

  ```java
  Bundle.getEntry("images/run.gif")
  Bundle.getEntries("images")
  ```

- In the given bundle and its fragments, supports translation

  ```java
  Platform.find(bundle, new Path("images/hello.txt"))
  Platform.find(bundle, new Path("$nl$/file.txt"))
  ```
What is the licensing model?

References
- EclipseCon’05: Getting your Plug-in Legal: a Primer for Eclipse Developers (I. Heffan)
- news://news.eclipse.org/eclipse.foundation
How do I filter contributions?

- Activities – An **activity** is a logical grouping of function that is centered around a certain kind of task. The contributions made available to the users are filtered using the enabled activities.

- It can also be adapted to hide unwanted contributions from other plug-ins.

**References**
- EclipseCon’05: Addressing UI Scalability in Eclipse (K. Horne)
- EclipseCon’05: Techniques for Seamless Integration: A Talk from the Trenches (J. Jones)
RCP Application and/or plug-ins to the IDE (or other RCP apps)?

- The IDE is an RCP application whose target audience is developers.
  - The main difference is that you do not have privileged access to the workbench through advisors

- An IDE and an RCP app are both made of plug-ins 😊

- The main recommendations are:
  - Do not put “business” logic into the advisor
  - Do not depend too much on the look and feel
  - Isolate the concerns of your application
  - You may have to write an integration plug-in to hook in some things in the IDE and adapt the workflows

Reference
- EclipseCon’05 : Eclipse RCP Everywhere (J.-M. Lemieux, J. McAffer)
What is the shape of a plug-in – the alternatives?

1. Plug-in folder containing the code jar
   
   ```
   pl ugi ns/
   org. eclipse. ui_3. 1. 0/
   ui.jar
   pl ugi n. xml
   ```

2. One jar containing the code jar
   
   ```
   pl ugi ns/
   org. eclipse. ui_3. 1. 0. jar
   (the jar contains)
   ui.jar
   pl ugi n. xml
   ```

3. Plug-in folder containing the class files “dot” is the classpath
   
   ```
   pl ugi ns/
   org. eclipse. ui_3. 1. 0/
   org/ eclipse/ ui /Ui.class
   org/ eclipse/ ui /Workbench.class
   pl ugi n. xml
   ```

4. One jar containing the classes “dot is the classpath”
   
   ```
   pl ugi ns/
   org. eclipse. ui_3. 1. 0. jar
   (the jar contains)
   org/ eclipse/ ui /Ui.class
   org/ eclipse/ ui /Workbench.class
   pl ugi n. xml
   ```

**Note**

- 1 and 4 are recommended. 2 can’t be tooled since java compilers do not support nested jars and 3 will result in too many files.
How do I reuse workbench preference pages?

See bug 73587
Widgets in action bars

- To populate your cool bars and status bars with text fields, drop down, etc.

1. Create a contribution item (by extending org.eclipse.jface.action.ContributionItem or ControlContribution)

2. Add this contribution item to the manager (for example MenuManager)

Reference
- Hyperbola code: SearchContribution or StatusActionGroup
MANIFEST.MF or plugin.xml

If you are targeting 2.1 the plugin.xml must be used

If you are targeting 3.0: the plugin.xml is recommended since there is no good support to edit 3.0 manifest

If you are targeting with one plug-in 3.0 and 3.1, the plugin.xml is recommended

If you are targeting 3.1, the manifest.mf is recommended since there will be appropriate tooling for it
How do I acquire an OSGi service?

- **ServiceTracker**
  
  ```java
  ServiceTracker tracker;
  tracker = new ServiceTracker(bundleCtx, "service.interface", null);
  tracker.open()
  tracker.getService()
  ```

- **BundleContext.getServiceReference**
  
  ```java
  ServiceReference packageAdminRef;
  packageAdminRef = ctx.getServiceReference(PackageAdmin.class.getName());
  if (packageAdminRef != null)
  ctx.getService(packageAdminRef);
  ```

- **Declarative services API**
  
  See bug 43890
Using flat look forms support in my editors and views?

- Use the plug-in: `org.eclipse.ui.forms`

```java
public void createPartControl(Composite parent) {
    toolkit = new FormToolkit(parent.getDisplay());
    form = toolkit.createScrolledForm(parent);
    form.setText("Hello, Eclipse Forms");
}
```

References