Building Commercial Quality Plug-ins for Eclipse

Dan Rubel, Chief Technology Officer
Eric Clayberg, VP of Product Development
Instantiations, Inc.
March 17, 2008
Course Materials

- All course materials are available:
  - From Eclipse USB flash drive
  - From http://www.instantiations.com/EclipseWorld2008/
  - File Name: QualityEclipse_v2.x.x_for_Eclipse3.3.zip

- USB flash drive
  - Copy contents of USB drive to local drive
  - Installation instructions on Slide #35 (Lab 1)
Overview

- Introduction
- Eclipse Architecture
- Plug-in Structure
- Aspects of an Eclipse based product
  - Actions, Views, Editors, …
- Discussion of related topics sprinkled throughout
  - Adapter Factory, Parameterized Extensions, OS Resources, …
  - Early Startup, Internal Classes, …
- Serviceability
  - Performance, Logging, Tracing, …
- Building your application
- Ready for Rational

Some slides derived from EPL-licensed Eclipse project briefing materials and slides on the IBM developerWorks web site.
Who Are We?

- Dan Rubel
  - Chief Technology Officer for Instantiations
- Eric Clayberg
  - V.P. of Product Development for Instantiations
- Dan & Eric
  - First used Java in 1996; Eclipse since 2000
  - Co-authors of several Eclipse articles for *WebSphere Advisor* magazine
  - Co-authors of *Eclipse: Building Commercial Quality Plug-ins*
  - Co-developers & architects of VA Assist Enterprise, CodePro and over a dozen other commercial software products
Instantiations

- Founded in 1997, Headquarters in Portland, OR
- Leading edge development tools for professional Java developers
  - VA Assist Enterprise (Smalltalk & Java)
  - CodePro Product Line (AnalytiX, Profiler, PlusPak, etc.)
  - WindowBuilder Pro (SWT, Swing & GWT Designer)
  - RCP Developer (Help Composer, RCP Packager)
  - WindowTester Pro (SWT & Swing)
- Extensive Eclipse experience
  - One of first IBM partners briefed on Eclipse in 1999
  - Technical development since January 2000 (>8 years)
  - First commercial Eclipse & WSAD add-on (Nov. 2001)
  - First product certified as “Ready for WebSphere Studio”
  - Eclipse Foundation member & major contributor
  - Eclipse Projects: KOI (Collaboration), Pollinate (Beehive)
Eclipse: Building Commercial Quality Plug-ins
Second Edition

- Publisher: Addison Wesley
- 3rd Edition due May 2008
- http://www.qualityeclipse.com/
- 854 pages
- Authors:
  - Eric Clayberg
  - Dan Rubel
- Series Editors:
  - Erich Gamma
  - Lee Nackman
  - John Wiegand
- Forwards by:
  - Skip McGaughey
  - Simon Archer

<table>
<thead>
<tr>
<th>1</th>
<th>Using Eclipse Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A Simple Plug-in Example</td>
</tr>
<tr>
<td>3</td>
<td>Eclipse Infrastructure</td>
</tr>
<tr>
<td>4</td>
<td>SWT Widgets</td>
</tr>
<tr>
<td>5</td>
<td>JFace Viewers</td>
</tr>
<tr>
<td>6</td>
<td>Actions</td>
</tr>
<tr>
<td>7</td>
<td>Views</td>
</tr>
<tr>
<td>8</td>
<td>Editors</td>
</tr>
<tr>
<td>9</td>
<td>Resources Change Tracking</td>
</tr>
<tr>
<td>10</td>
<td>Perspectives</td>
</tr>
<tr>
<td>11</td>
<td>Dialogs &amp; Wizards</td>
</tr>
<tr>
<td>12</td>
<td>Preference Pages</td>
</tr>
<tr>
<td>13</td>
<td>Property Pages</td>
</tr>
<tr>
<td>14</td>
<td>Builders, Markers &amp; Natures</td>
</tr>
<tr>
<td>15</td>
<td>Implementing Help</td>
</tr>
<tr>
<td>16</td>
<td>Internationalization</td>
</tr>
<tr>
<td>17</td>
<td>Extension Points</td>
</tr>
<tr>
<td>18</td>
<td>Features &amp; Branding</td>
</tr>
<tr>
<td>19</td>
<td>Product Build</td>
</tr>
<tr>
<td>20</td>
<td>Advanced Topics</td>
</tr>
</tbody>
</table>
Ready for Rational

“The Ready for IBM Rational software validation offering helps IBM Business Partners demonstrate that their product offerings integrate with key products in the IBM Software Development Platform (SDP).”

Qualification

- Business Partner must be commercial members of IBM PartnerWorld

Certification Process

- Requirements document and checklist
- Live demonstration of product

Benefits

- Use of the Ready for IBM Rational software mark
- An entry in IBM’s Global Solutions Directory (GSD)
- Participation on the Plug-in Central Web site
- Other technical and marketing benefits to be defined over time
Eclipse Workbench

- Menu Bar
- Page
- Editor
- Tool Bar
- Views
- Workbench Window
Eclipse Architecture Overview
Eclipse Architecture Details

- **Plug-in** - smallest unit of Eclipse function
  - Promotes modular architecture
  - Code can be rearranged to meet different needs
  - Big example: HTML editor
  - Small example: Action to create zip files

- **Extension point** - named entity for collecting “contributions”
  - Promotes loose coupling between plug-ins resulting in more flexibility
  - Facilitates other tools to be used within the platform
  - Easily extend existing platform - no need to wait for new product releases
  - Example: extension point for workbench preference UI

- **Extension** - a contribution
  - Example: specific HTML editor preferences
Eclipse Plug-in Architecture

- Each plug-in
  - Depends on a set of other plug-ins
  - Contains Java code libraries and other files
  - May export APIs for downstream plug-ins
  - Lives in its own plug-in subdirectory
  - Defines 1 or more extensions
  - Optionally declares new extension points

- Details spelled out in the plug-in manifest
  - Manifest declares contributions
  - Code implements contributions and provides API
  - plugin.xml file in root of plug-in subdirectory
Extension points

- Structured way to allow new functionality
  - Facilitates loose coupling between plugins
  - Schema describes extension format

- External tools extend the platform
  - Java Development Tooling (JDT)
  - Plug-in Development Environment (PDE)

- Extension points may have a corresponding API interface
  - Describes what should be provided in the extension
Basic Workbench Extension Points

View Action  Editor Actions  Action Set Actions

Resource: sample1.readme  -  Eclipse SDK

File  Edit  Navigate  Search  Project  Run  Readme  Window  Help

1. SECTION 1
This text is a placeholder for the content.
1.1 Subsection
This text is a placeholder for the content.

1. SECTION 2
2.1 Subsection
2.2 Subsection

Outline  Tasks  Readme Sections

Content Outline  Custom View  Trim

SAMPLE README FILE

1. SECTION 1
This text is a placeholder for the content.
1.1 Subsection
This text is a placeholder for the content.
What's in a plug-in?

- META-INF/MANIFEST.MF
  - Manifest that describes plug-in
- plugin.xml
  - Manifest that describes plug-in
- plugin.properties
  - Plugin-in properties
- A JAR file
  - An archive with the plug-in code
- about.html
  - Textual description of the plug-in
Plug-in Structure

OSGI Manifest
- name
- identifier
- classpath
- required plug-ins

Plug-in Manifest
- extensions
- extension-points

Required Plug-ins
Plug-in Development Environment (PDE)

- Specialized tools for developing Eclipse plug-ins
- Built atop Eclipse Platform and JDT
  - Implemented as Eclipse plug-ins
  - Using Eclipse Platform, JDT APIs and extension points
- Included in Eclipse Project releases
  - Separately installable feature
  - Part of Eclipse SDK drops
- Goals:
  - Make it easier to develop Eclipse plug-ins
  - Support self-hosted Eclipse development
Plugging-in Creation

New Project
Select a wizard
Create a Plug-in Project

New Plug-in Project
Plug-in Project
Create a new plug-in project

Plug-in Content
Enter the data required to generate the plug-in.

Plug-in Properties
- Plug-in ID: com.quality.eclipsefavorites
- Plug-in Version: 1.0.0
- Plug-in Name: Favorites Plugin
- Plug-in Provider: Quality Eclipse
- Classpath:

Plug-in Options
- Generate an activator, a Java class that controls the plug-in's life cycle
- Activator: com.quality.eclipsefavorites.Activator
- This plug-in will make contributions to the UI

Rich Client Application
Would you like to create a rich client application? (Yes/No)
Action Creation

Create actionSet containing menus and actions
Action Set – Customize Perspective

Can be turned on or off by user
**Action Declaration**

- **actionSet** contains:
  - menus
  - and/or
  - actions

---

```xml
<extension
  point="org.eclipse.ui.actionSets">
  <actionSet
    label="Favorites Action Set"
    visible="true"
    id="com.qualityeclipse.favorites.workbenchActionSet">
    <menu
      label="Favorites"
      path="additions"
      id="com.qualityeclipse.favorites.workbenchMenu">
      <separator
        name="content">
      </separator>
      <separator
        name="additions">
      </separator>
    </menu>
    <action
      id="com.qualityeclipse.favorites.openFavoritesView"
      label="Open Favorites View"
      icon="icon/sample.gif"
      tooltip="Open the favorites view in the current workbench page"
      memberPath="com.qualityeclipse.favorites.workbenchMenu/content"
      toolbarPath="Normal/additions"
      class="com.qualityeclipse.favorites.actions.OpenFavoritesViewActionDelegate">
      // Implementation details...
    </action>
  </actionSet>
</extension>
```

---

```java
public class OpenFavoritesViewActionDelegate implements IWorkbenchWindowActionDelegate {
    // Implementation details...
    public void run(IAction action) {
        // Implementation details...
    }
}
```
Action Location

**path**
Where menu will appear
- "additions" = top level menu
- "menu-id/group-id"
- "menu-id/additions"

**menubarPath**
Where menu item will appear
- "menu-id/group-id"
- "menu-id/additions"
Action Delegate

Action
- Class in Eclipse plug-in
- Contains display state
- Instantiated when displayed

Action Delegate
- Class in your plug-in
- Contains execution behavior
- **Instantiated when selected**
- IWorkbenchWindowActionDelegate

Lazy plug-in loading
Action Display State

Display State
- Text
- Image
- Tool tip
- Visibility
- Enablement

Defined in Manifest

Refined in Delegate

```xml
<extension>
    <point key="org.eclipse.ui.actionSets">
        <actionSet>
            <label>Favorites ActionSet</label>
            <visible>true</visible>
            <id>com.qualityeclipse.favorites.workbenchActionSet</id>
        </actionSet>
    </point>
</extension>
```

```java
public class OpenFavoritesViewActionDelegate implements IWorkbenchWindowActionDelegate {
    @Override
    public void run(IAction action) {
        // Open the favorites view.
    }
}
```
Action Visibility and Enablement

Two places where display behavior can be defined:

**Plug-in manifest**
- Visibility – logical expression
  - objectClass
  - objectState (IActionFilter)
  - pluginState
  - systemProperty
- Enablement – logical expression
  - objectClass
  - objectState (IActionFilter)
  - pluginState
  - systemProperty

**Action Delegate**
- selectionChanged(…)
  - Refine display state
  - Must execute quickly
  - If too slow then assume enabled
- run(…)
  - Guard code to check enabled

Your plug-in need not be loaded but **cannot express all display behavior**

Can contain any display behavior but **not executed until action selected**

Push display behavior into manifest whenever possible
Delegates not instantiated until menu item is selected

- `init(IAction action)`
  - Called immediately after object instantiated
- `selectionChanged(IAction action, ISelection selection)`
  - Not called until delegate instantiated when action first selected
  - Called every time selection changes
  - Should update display state of action
  - **Must execute quickly**
- `run(IAction action, Event event)`
  - Called when users selects the action
  - Use `setRedraw(false/true)` to reduce flickering
- `dispose()`
  - Cleanup delegate… last method called on this object
IActionDelegate2 and IExecutableExtension

- Implement IExecutableExtension
- **Unstructured** data in manifest is passed to Action Delegate

```xml
<action
    label="Open Favorites View"
    icon="icons/sample.gif"
    tooltip="Open the favorites view in the current workbench page"
    menubarPath="com.qualityeclipse.favorites.workbenchMenu/content"
    toolbarPath="Normal/additions"
    id="com.qualityeclipse.favorites.openFavoritesView"
    class="myPackage.MyActionDelegate;one two three">
</action>
```

```java
public class MyActionDelegate
    implements IActionDelegate, IExecutableExtension
{
    public void setInitializationData(IConfigurationElement config, String propertyName, Object data)
        throws CoreException
    {
        // perform initialization here
    }
}
```

Parameterize any class specified in the plug-in manifest such as actions delegates, views, editors, etc…
IActionDelegate2 and IExecutableExtension

- Implement IExecutableExtension
- **Structured** data in manifest is passed to Action Delegate

```xml
<action>
  <label>Open Favorites View</label>
  <icon>icons/sample.gif</icon>
  <tooltip>Open the favorites view in the current workbench page</tooltip>
  <menuPath>com.quality.eclipse.favorites.workbenchMenu/content</menuPath>
  <toolbarPath>Normal/<additions>
  <id>com.quality.eclipse.favorites.openFavoritesView</id>
  <class>myPackage.MyActionDelegate</class>
  <parameter name="p1" value="one"/>
  <parameter name="p2" value="two"/>
  <parameter name="p3" value="three"/>
</action>
```

```java
public class MyActionDelegate
    implements IActionDelegate, IExecutableExtension
{
    public void setInitializationData(IConfigurationElement config, String propertyName, Object data)
        throws CoreException
    {
        // perform initialization here
    }
}
```

Parameterize any class specified in the plug-in manifest such as actions delegates, views, editors, etc…
Top Level Menu

We just defined a top level menu

- Great visibility - show off a product
- Quick user access to your product’s functionality

But what if every product did this?

- Define as visible in the plug-in manifest
  - Menu is visible when product is first installed
- Provide preference to hide menu
  - New preference in your product’s preference page
  - IWorkbenchPage.hideActionSet(…) – hide menu from current perspectives
  - IActionSetDescriptor.setInitiallyVisible(…) – hide menu in future perspectives
Context Menu Declaration

popupMenu contains:

menus and/or actions

```xml
<extension>
  <objectContribution>
    <objectClass>org.eclipse.core.resources.IResource</objectClass>
    <objectAdapter>true</objectAdapter>
    <objectPart>
      <menu>
        <label>Favorites</label>
        <path>additions</path>
        <id>com.qualityeclipse.favorites.popupSubMenu</id>
        <groupMarker>
          <name>content</name>
        </groupMarker>
        <separator name="additions"/>
        <separator/>
        <menu>
          <action id="com.qualityeclipse.favorites.addToFavorites">
            <label>Add</label>
            <icon>icons/sample.gif</icon>
            <tooltip>Add the currently selected resource(s) to the Favorites view</tooltip>
            <menuPath>com.qualityeclipse.favorites.popupSubMenu</menuPath>
            <enableCondition="/"
            <class">com.qualityeclipse.favorites.actions.AddToFavoritesActionDelegate</class>
          </action>
          <separator/>
          <groupMarker>
            <name>actions</name>
          </groupMarker>
          <separator/>
          <menu>...
        </menu>
      </menu>
    </objectPart>
  </objectContribution>
</extension>
```

```java
public class AddToFavoritesActionDelegate implements IObjectActionDelegate {

    /**
     * Add the selection to the Favorites view.
     */
    public void run(IAction action) {
        ...
    }
}
```
Context Menu Location

**path**
Where sub-menu will appear
- “additions” = in context menu
- “menu-id/group-id”
- “menu-id/additions”

**menubarPath**
Where menu item will appear
- “menu-id/group-id”
- “menu-id/additions”

Lazy plug-in loading
Context Menu Delegate

**Action**
- Class in Eclipse plug-in
- Contains display state
- Instantiated when displayed

**Action Delegate**
- Class in your plug-in
- Contains execution behavior
- **Instantiated when selected**
- IObjectActionDelegate

**Lazy plug-in loading**
Context Menu Display State

**Display State**
- Text
- Image
- Tool tip
- Visibility
- Enablement

**Defined in Manifest**

**Refined in Delegate**

**Visible in any view based upon current selection**

```
public class addToFavoritesActionDelegate implements IObjectActionDelegate {
    /**
     * Add the selection to the Favorites view.
     * @param action
     * @return true or false
     */
    public boolean run(IAction action) {
        ...
    }
}
```
View Specific Actions

- Object specific menus appear in any view based upon current selection
- View specific menus appear only in one specific view
- Similar declaration, but delegate must implement IViewActionDelegate
Editor Specific Actions

- Object specific menus appear in any view based upon current selection.

- Editor specific menus appear only when specific editor is active.

- Similar declaration, but delegate must implement IEditorActionDelegate.
Actions – Ready for Rational

- Action labels should conform to workbench usages
  - New and Delete – creation and disposal of an object
  - Add and Remove – manipulating collections of object
- Action icons should be
  - 16 x 16 (preferred), 24 x 24, 32 x 32
  - 1 pixel border along left and bottom
- Reduce visibility and enablement as much as possible
  - Lazy loading
  - Reduce UI clutter
- Action should enable quickly
  - If cannot determine action visibility and enablement quickly then enable the action optimistically
- Make actions reversible when possible
  - Single action should correlate to single undo
Lab 1 – Installing Tools & Samples

1. Unzip the QualityEclipse_v2.x.x_for_Eclipse3.3.zip file to any place that is convenient
2. Start Eclipse and access the Update Manager via Help > Software Updates > Find and Install...
3. Select Search for new features to install
4. Click New Local Site and navigate to the “/QualityEclipse/eclipse” directory created in step #1
5. Click OK in the Edit Local Site dialog
6. Make sure that “QualityEclipse/eclipse” is the only item checked and click Finish
7. Make sure that the top level item in the resulting Updates/Install wizard is checked
8. Accept the license and click Next
9. Click Finish on the last page of the Install wizard
10. Click Yes to restart Eclipse, and the QualityEclipse tools and samples should then be available for use.
Lab 1 – Actions

- Select the Category “EclipseCon 2008 Labs”
- Select “Lab 0 – Setup”
- Click the import toolbar button

At any time during the lab, you can compare your work with the expected result
Lab 1 – Advanced Search – 1

Eclipse provides excellent search facility, but only searches workspace and referenced plug-ins.

Problem:
How to search all Eclipse plug-ins?

Solution 1:
Create one binary project for each Eclipse plug-in

Solution 2:
Use plug-in view to “Add to Java Search”

Solution 3:
Create a reference project containing all Eclipse plug-ins

Wizard for creating reference projects:
www.qualityeclipse.com/tools
Lab 1 – Advanced Search – 2

- Select **Navigate > Open Type...** to open the **Open Type** dialog

- Type "Plugin" and select **org.eclipse.core.runtime.Plugin** to open an editor on that class
Repeat this process but type "Team" and you will not find any classes.

We know there are many Eclipse classes starting with "Team", but where are they?

Our "lab" project only references a few Eclipse plug-ins and only those plug-ins are searched.
Lab 1 – Advanced Search – 4

- Create a reference project to index all Eclipse plug-ins
- Select File > New > Project... to open the New Project wizard
Lab 1 – Advanced Search – 5

- Select **QualityEclipse Tools > New Reference Project**
- Click **Next** then click **Finish**
- **New Reference** project is created
Lab 1 – Advanced Search – 6

- Select **Navigate > Open Type...** to open the **Open Type** dialog
- Now type "Team" and you will all of the Eclipse classes that start with "Team"
Lab 1 – Adding a context menu action 1

- Open the plug-in manifest editor
- Click the “Extensions” tab
- Click the “Add” button to add an extension
Lab 1 – Adding a context menu action 2

- Enter “*popup” in the filter field
- Select the popupMenus extension
- Click Finish
Lab 1 – Adding a context menu action 3

- Right-click on the new extension
- select New > objectContribution
Lab 1 – Adding a context menu action

- Click Browse
- Enter “IResource” in the filter field
- Select IResource in the list
- Click OK
- If IResource does not appear, then add dependency and save editor then try selecting IResource again
Lab 1 – Adding a context menu action

- Right-click and select New > action
Lab 1 – Adding a context menu action

- Enter a label for the new Action
- Click on the class field label to open the New Java Class dialog
Lab 1 – Adding an action class

- Enter the name of the class to be created
- Click Add to open the Interface Selection dialog
- Enter “iobj” in the filter field, select IObjectActionDelegate
- Click OK and then Finish
Lab 1 – Plug-in dependency warning

- After saving your changes, a warning **may** appear
- Double-click on the warning to open the editor
- Class is not in plug-in’s dependency list so the Eclipse compiler cannot find that class
Lab 1 – Finding which plug-in contains a class

- Select Navigate > Open Type
- Enter class name in filter field
- Select class in list
- Plug-in identifier appears in path
Lab 1 – Resolving plug-in dependency problem

- Open plug-in manifest editor
- Switch to Dependencies page
- Click Add button
- Enter “*resource” in filter field
- Select plug-in from list
- Click OK and save editor content
Lab 1 – Using auto import

- Open action delegate editor
- Locate run(...) method
- Enter “MessageDialog”
- Press Ctrl-Space to display possible imports
- Select MessageDialog import and press Enter
Lab 1 – Using auto completion

- Place cursor after `MessageDialog`
- Enter "openi"
- Select `openInformation` from list and press Enter
Lab 1 – Completing the new action

- Enter the code defining the action’s behavior

```java
public void setActivePart(IAction action, IWorkbenchPart targetPart) {
    // TODO Auto-generated method stub
}

public void run(IAction action) {
    MessageDialog.openInformation(null, "My Action", "It works!");
}

public void selectionChanged(IAction action, ISelection selection) {
    // Todo
}
```
Lab 1 – Add Object Action for IJavaElement

Repeat prior steps to add Plug-in dependency (save editor after adding dependency) then new object action.
Lab 1 – Testing the new context menu action

- Launch a runtime workbench
- Create a new plug-in project
- Right-click on build.properties
- Select My Object Action
Lab 1 – Completed

Compare your work with the expected result

- Select “Lab 1 – Actions”
- Click Compare to see differences
Views

- Views provide information on some object
- Views augment editors
  - Example: Outline view summarizes content
- Views augment other views
  - Example: Properties view describes selection

- Extension point for new types of views
- Eclipse Platform includes many standard views
  - Examples: Resource Navigator, Outline, Problems, Search, …
- View API and framework
  - Views can be implemented with JFace viewers
View Creation

Create a view in the plug-in manifest
View Declaration

- Declaration contains categories and views

```xml
<extension point="org.eclipse.ui.views">
    <category
        name="Quality Eclipse"
        id="com.qualityeclipse.favorites"/>
    <view
        category="com.qualityeclipse.favorites"
        name="Favorites"
        icon="icons/sample.gif"
        id="com.qualityeclipse.favorites.views.FavoritesView"
        class="com.qualityeclipse.favorites.views.FavoritesView"/>
</extension>
```
View Content Provider

Content Provider
- Consumes “input” object
- Produces all possible objects

In this case, it produces an array of person objects

Input object may be dependent upon current selection in page

Describing a table based view but anything is possible
View Filters

Viewer Filters
- Consume all possible objects
- Produce objects to be displayed

View menu with “Filter” action to select which filters are used
View Sorter

Viewer Sorter
- Consume unordered objects
- Produce ordered objects

TableViewSorter
- One comparator for each column
- Tracks order in which comparators are used
- Click column to sort, click again to reverse sort
- Click column A then B to sort by B then A
View Label Provider

Label Provider
- Consumes object and column
- Produces string/image to display

To display workbench objects, see
- WorkbenchLabelProvider
- WorkbenchPartLabelProvider
View Actions

- Create actions not delegates
  - Subclass of Action – no need to implement IActionDelegate unless you plan to reuse them elsewhere as contributions

- Three places actions can appear:
  - Special separator for contributions
  - Group – “additions”
View Context Menu – Initialization

- Remove all actions each time menu is displayed
  ```java
  menuMgr.setRemoveAllWhenShown(true);
  ```

- Register so others can contribute
  ```java
  getSite().registerContextMenu(menuMgr, viewer);
  ```

- Add menu listener to build menu dynamically
  ```java
  menuMgr.addMenuListener(new IMenuListener() {
      public void menuAboutToShow(IMenuManager m) {
          … build menu …
      }
  });
  ```
View Context Menu – Dynamically Building

- Add local actions specific to the view
- Add global actions such as Cut, Copy, Paste, …
- Add special group so others can contribute actions
  
  ```java
  menuMgr.add(new Separator(IWorkbenchActionConstants.MB_ADDITIONS));
  ```

Local

Global

Contributions
View Context Menu – Global Actions

Hook Global Actions to enable edit key bindings – Ctrl-X for Cut

```java
private void hookGlobalActions() {
    getViewSite().getActionBars().setGlobalActionHandler(IWorkbenchActionConstants.DELETE, removeAction);
    getViewSite().getActionBars().setGlobalActionHandler(IWorkbenchActionConstants.COPY, copyAction);
    getViewSite().getActionBars().setGlobalActionHandler(IWorkbenchActionConstants.CUT, cutAction);
    getViewSite().getActionBars().setGlobalActionHandler(IWorkbenchActionConstants.PASTE, pasteAction);
}
```
Typically, global edit actions manipulate the clipboard

- Convert selected items to other formats – IResource
- Associate a handler with that format – ResourceTransfer.getInstance()

```java
public class CopyFavoritesAction extends Action {
    public void run() {
        IFavoriteItem[] items = view.getSelectedFavorites();
        try {
            view.getClipboard().setContents(
                new Object[] { asResources(items), asText(items), },
                new Transfer[] { ResourceTransfer.getInstance(), TextTransfer.getInstance(), });
        } catch (SWTError error) {
            ...
        }
    }
}
```
View Tool Bar

- Add Actions to Tool Bar
  
  ```java
  getViewSite().getActionBars().getToolBarManager().add(removeAction);
  ```

- Add selection listener to update enablement
  
  ```java
  viewer.addSelectionChangedListener(new ISelectionChangedListener() {
      public void selectionChanged(SelectionChangedEvent event) {
          removeAction.setEnabled(!event.getSelection().isEmpty());
      }
  });
  ```
View Pulldown Menu

- Similar to context menu, except
  - Not based upon selection
  - Not dynamically built – just add the actions during menu initialization
  - No selection listener – actions always enabled
View Drag and Drop

Drag Source
- Associate a drag source with the control
  DragSource source = new DragSource( viewer.getControl(), DND.DROP_COPY);

- Set the transfer types (similar to clipboard manipulation)
  source.setTransfer( new Transfer[] { TextTransfer.getInstance(), ResourceTransfer.getInstance() });

- Add a drag listener
  source.addDragListener(new DragSourceListener() {…});

Drop Target
- Associate a drop target with the control
  DropTarget target = new DropTarget( viewer.getControl(), DND.DROP_MOVE | DND.DROP_COPY);

- Set the transfer types (similar to clipboard manipulation)
  target.setTransfer( new Transfer[] { ResourceTransfer.getInstance(), JavaElementTransfer.getInstance() });

- Add a drop listener
  target.addDropListener(new DropTargetListener() {…});
View – Saving Local State Information

Local state information typically includes:

- References to objects being displayed
- Current sort order
- Currently enabled filters
- Currently selected objects

Local state information typically stored using mementos:

- `saveState(IMemento memento)` – called by Eclipse during shutdown
  
  ```java
  mem = memento.createChild("SortInfo");
  mem.putInteger("SortColumn", info.columnIndex);
  ```

- `init(IViewSite site, IMemento memento)` – called during view initialization
  
  ```java
  IMemento mem = memento.getChild("SortInfo");
  if (mem != null) info.columnIndex = mem.getInteger("SortColumn");
  ```
View – Saving Global State Information

Global state information typically includes:

- Model information shared by all views of this type

Global state information typically stored in plug-in metadata area:

- Use XML file for highly structured data
- Lazily load information when needed
- Save information during plug-in shutdown
  ```java
  public void stop(BundleContext context) {
      ... save information ...
  }
  ```
- ISaveParticipant for snapshots throughout Eclipse session
  ```java
  public void saving(ISaveContext context) {
      ... save information ...
  }
  ```
View – Managing OS Resources

Motto: If you create it, then you dispose it

Some Java objects have associated OS resources
- Images, Fonts, Colors
- Potential memory leak – must be disposed properly

Cache OS Resources where they are used
- Label provider
- View
- Plug-in
- Product

Some resources are provided for you
- Workbench images, System fonts and colors
- Eclipse ensures proper disposal – do not dispose yourself
View – Managing Images

Motto: If you create it, then you dispose it

Plug-in manifest
- Images declared in plug-in manifest are managed by Eclipse
- Let Eclipse manage the images whenever possible

Workbench images
- PlatformUI.getWorkbench().getSharedImages().getImage(…);
- Managed by Eclipse – Do not dispose

Label Providers sometimes cache images
- Dispose images when label provider is disposed
- Make sure label provider itself is disposed

ImageRegistry in AbstractUIPlugin
- May override initializeImageRegistry to pre-cache images
- Override stop(…) method to dispose images when plug-in exits
View – Managing Images

Motto: If you create it, then you dispose it

ImageDescriptor
- Describes an image
- Pure Java object – no OS component

Image
- Actual image used for display
- Has OS component – must be disposed properly

Caching

public Image getImage(ImageDescriptor imageDescriptor) {
    Image image = (Image) imageMap.get(imageDescriptor);
    if (image == null) {
        image = imageDescriptor.createImage();
        imageMap.put(imageDescriptor, image);
    }
    return image;
}

public void dispose() {
    Iterator iter = imageMap.values().iterator();
    while (iterator.hasNext())
        ((Image) iter.next()).dispose();
    imageMap.clear();
}
View – Managing Fonts

Motto: If you create it, then you dispose it

Workbench Fonts
- JFaceResources – Default, Banner, Dialog, Header, Text, Viewer

Local Fonts
- font = new Font(display, “Arial”, 14, SWT.NORMAL);
- Has OS component – must be disposed properly

Caching Fonts
- Label Provider
- View
- Editor
View – Managing Colors

Motto: If you create it, then you dispose it

Workbench Colors

- JFaceResources.getColorRegistry()
- JFaceColors – banner background, banner foreground, error text, ...

Local Colors

- color = new Color(display, reg, green, blue);
- Has OS component – must be disposed properly

Caching Colors

- Label Provider
- View
- Editor
Linking Views – Selection Provider

Provide selection to others
- `getSite().setSelectionProvider(viewer);`
- `viewer -or- other object implementing ISelectionProvider`

Provide adaptable objects
```
public class MyModelClass ..... implements IAdaptable {
{
    private IResource resource;
    ..... 
    public Object getAdapter(Class type) {
        if (type.isInstance(this))
            return this;
        if (type.isInstance(resource))
            return resource;
        return Platform.getAdapterManager().getAdapter(this, type);
    }
}
```
Linking Views – IAdaptable

Adapting MyModelClass to IResource
- `object.getAdapter(MyModelClass.class);`
- MyModelClass implement IAdaptable

But… Adapting IResource to MyModelClass… how?
- `object.getAdapter(IResource.class)`
- Implement how?
- IResource has no knowledge of MyModelClass
- Cannot change base code
- So… create an Adapter Factory
Linking Views – IAdapterFactory

Adapter Factory

```java
public class MyAdapterFactory implements IAdapterFactory {
    public Class[] getAdapterList() {
        ... return array of types this factory adapts to ...
        return new Class[] {MyModelClass.class};
    }

    public Object getAdapter(Object object, Class type) {
        ... adapt object to type -or- return null ...
        if (object instanceof MyModelObject && IResource.class.equals(type))
            return ((MyModelObject) object).getResource();
        return null;
    }
}
```

Register Adapter Factory when the plug-in starts up

```java
Platform.getAdapterManager().registerAdapters(…);
```

Unregister Adapter Factory when the plug-in shuts down

```java
Platform.getAdapterManager().unregisterAdapters(…);
```
Linking Views – IWorkbenchAdapter2

- Provides additional display information for the workbench

IWorkbenchAdapter
- Text
- Image

IWorkbenchAdapter2
- Foreground Color
- Background Color
- Font
Linking Views – Selection Listener

Selection change listener
- `getSite().getPage().addPostSelectionListener(…);`
- `getSite().getPage().removePostSelectionListener(…);`

Getting the selection
- Check instanceof IStructuredSelection
- Extract objects from selection

Adapting selected objects
Object `obj = selectedObjects[index];`
if (obj instanceof IAdaptable)
  `obj = (IResource) obj.getAdapter(IResource.class);`
if (obj instanceof IResource)
  `mySelectedObjects.add(obj);`

… change selection in view based on mySelectedObjects …
Views – Ready for Rational

- **Save Immediately**
  Actions in a view have immediate effect on the model
  There is no “save” action in a view

- **One or more views?**
  If a view contains more than one control,
  it may be advisable to split it up into two or more views.

- **View initialization**
  When a view first opens, derive the input from the perspective state

- **Selection oriented actions**
  Context menu should contain selection oriented actions
  Presentation oriented actions may appear in tool bar and pulldown menu

- **Register context menus**
  Register all context menus so that it may be automatically extended by the platform

- **View global actions from menu bar**
  If view supports global actions, they must be executable from Edit menu

- **Persist view state**
  Persist the state of each view between sessions
Lab 2 – Setup

- Select the Category “EclipseCon 2008 Labs”
- Select “Lab 1 – Actions”
- Click the import toolbar button

At any time during the lab, you can compare your work with the expected result.
Lab 2 – Creating a new view 1

- Open the plug-in manifest editor
- Switch to the Extensions page
- Click Add
Lab 2 – Creating a new view 2

- Enter "*view" in the filter field
- Select the views extension-point
- Select Sample View
- Click Next then Finish
Lab 2 – Review the generated code

- New view extension generated
- Next perspectiveExtension generated so view will automatically appear in the resource perspective
- New class generated to implement behavior for the new view
Lab 2 – Test the new view

- Launch runtime workbench
- Select Window > Show View > Other...
- Select Sample Category > Sample View
- Click OK and new view appears
Lab 2 – Test the new view’s context menu

- Sample View appears
- Right click on any item and see popup menu with 2 items

- Close runtime workbench
Lab 2 – Completed

- Compare your work with the expected result
- Select “Lab 2 – Views”
- Click Compare to see differences
Views versus Editors

Views and Editors differ in purpose and lifecycle

<table>
<thead>
<tr>
<th></th>
<th>View</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Display information</td>
<td>Modify information</td>
</tr>
<tr>
<td>Lifecycle</td>
<td>Actions have immediate effect on the model</td>
<td>Follows the editor lifecycle open – modify – save – close</td>
</tr>
</tbody>
</table>
Editors

- Editors appear in workbench editor area
- Contribute actions to workbench menu and tool bars
- Open, edit, save, close lifecycle
- Open editors are stacked by default

- Extension point for contributing new types of editors
- Eclipse includes many standard editors
  - Eclipse Platform includes simple text file editor
  - JDT provides Java source file editor
- Windows only: embed any OLE document as editor
- Extensive text editor API and framework
Editor Creation

Create an editor in plug-in manifest
Editor Declaration

- Declaration
  - Name
  - Identifier

- Editor
  - Edit Controls
  - Context Menu

- Editor Contributor
  - Top Level Menu
  - Tool Bar Buttons
  - Global Edit Actions

```xml
<extension point="org.eclipse.ui.editors">
  <editor
    id="com.qualityclipse.favorites.editors.PropertiesEditor"
    extensions="properties"
    name="Properties Editor"
    icon="icons/sample.png"
    contributorClass="com.qualityclipse.favorites.editors.PropertiesEditorContributor"
    class="com.qualityclipse.favorites.editors.PropertiesEditor">
  </editor>
</extension>
```
Editor Lifecycle

Open
- `setInitializationData(…)` – called with arguments specified in plug-in manifest
- `createPartControl(Composite parent)` – create the editor controls
- `init(IEditorSite site, IEditorInput input)` – initialize the editor’s content
- `setFocus()` – called when editor becomes the active part

Modify
- `firePropertyChanged(PROP_DIRTY)` – should be called when content modified
- `isDirty()` – should return true when content modified but not saved

Save
- `doSave(IProgressMonitor monitor)` – should save content
- `doSaveAs()` – prompt user and call `doSave(…)`
- `isSaveAsAllowed()` – returns true if user can save content to another location

Close
- `dispose()` – called while Eclipse is closing the editor
Editor Context Menu – Initialization

- Remove all actions each time menu is displayed
  ```java
  menuMgr.setRemoveAllWhenShown(true);
  ```

- Register so others can contribute
  ```java
  getSite().registerContextMenu(menuMgr, tree);
  ```

- Add menu listener to build menu dynamically
  ```java
  menuMgr.addMenuListener(new IMenuListener() {
    public void menuAboutToShow(IMenuManager m) {
      ... build menu ...
    }
  });
  ```
Editor Context Menu – Build Dynamically

- Add local actions specific to the editor
- Add global actions such as Cut, Copy, Paste, …
- Add special group so others can contribute actions
  
  ```java
  menuMgr.add(new Separator(IWorkbenchActionConstants.MB_ADDITIONS));
  ```
Editor Contributor

Manages tool bar buttons, menus and menu items

- **init(…)**
  - Called when first created
  - Initialize actions

- **setActiveEditor(…)**
  - Called when editor becomes active or inactive
  - Contributor should insert and remove menus and toolbar buttons as appropriate

- **dispose()**
  - Called when contributor is no longer needed
  - Contributor should release OS resources created during init(…)
Editor – Global Actions

Typical Global Actions

- Cut
- Copy
- Paste
- Select All

When editor becomes active, hook global actions

```java
public void setActiveEditor(IEditorPart part) {
    actionBars.setGlobalActionHander(
        IWorkbenchActionConstants.CUT, textEditor.getAction(ITextEditorActionConstants.CUT)
    );
    ... etc ...
}
```
Editor – Ready for Rational

- Editor lifecycle
  - Modifications made in an editor must follow an open-save-close lifecycle
- Single editor per input
  - Only one editor instance per editor input within a perspective
- Separate editors for each input
  - It must be possible to open separate editor instances for each input
- Accessing global actions
  - Global actions must be executable from the window’s Edit menu
- Registering editor menus
  - Register all context menus so that others may contribute actions
- Close editor when object deleted
  - If file is deleted and the editor is unmodified, then editor should automatically close
- Unsaved editor modifications
  - If file is deleted and the editor is modified, then editor should warn user
- Editor outline view
  - Editor should provide an outline model for the Outline view
- Synchronize with outline view
  - Notification about location in editor and outline view should be two-way
- Synchronize with external changes
  - If modifications occur outside the workbench, user should be prompted to override or accept
Perspectives

- Perspectives are arrangements of views and editors
- Different perspectives suited for different user tasks
- Users can quickly switch between perspectives
- Task orientation limits visible views, actions
  - Scales to large numbers of installed tools
- Perspectives control
  - View visibility
  - View and editor layout
  - Action visibility
- Extension point for new perspectives
- Eclipse includes standard perspectives
  - Resource, Java, Debug, Team Synchronization, …
- Perspective API for programmatic access
Perspectives

org.eclipse.ui.perspectiveExtensions extension-point

- Add a view to an existing perspective
  - Relative – to another view
  - Ratio – percentage of other view’s area
  - Relation – to the specified view: stacked, left, right, top, bottom
  - Initially visible or not
- Add a shortcut
  - To a view
  - To another perspective
- Add an Action Set
Perspective Factory

- Declaration
  - Name
  - Identifier
  - Icon

- Factory
  - Views
  - Actions
Perspectives – Ready for Rational

- Create perspectives for long lived tasks
  A new perspective should be created when a group of related non-modal tasks

- New view for existing perspective
  Augment an existing perspective rather than creating a new perspective with only a few views

- Perspectives without editors
  If it is undesirable to have editors in the perspective then hide the editor area

- Adding perspective actions to window menu
  Populate the window menu bar with actions appropriate to the tasks to be completed in that perspective
Dialogs

- Three basic areas
- Avoid modal when possible
- Typical icons in message dialog

- Error
- Warning
- Information
- Question
Details Dialog

**Serviceability**
Ready for Rational requires that all error messages generated by a product appear in a dialog with a details area

- Error
- Provider Information
- Stack Trace
Dialogs – Ready for Rational

- **Initial dialog focus**
  When a dialog opens, set the initial focus to one of its controls

- **Serviceability**
  Display errors in an error dialog with details area showing the product provider along with additional error information

- **Use wizards for multi-step tasks**
  Use a wizard for tasks consisting of many steps that must be performed in a specific order
Wizards

- Used for modal task consisting of many steps that must be performed in a specific order
- Layout should be preserved

Error Messages

Wizard page defines controls in content area

Navigation Buttons
Wizards – Ready for Rational

- Initial wizard focus
  When a wizard opens, set the initial focus to one of its controls
- Wizard look and feel
  Follow the platform wizard look and feel
- Wizard field initialization
  Seed the wizard fields using the current workbench state
- Wizard data validation
  Validate the wizard data in tab order
- Wizard browse button
  Use an edit field and browse button whenever an existing object is referenced
- Open new file in editor
  If a new file is created, then open the file in an editor when wizard closes
- New project switches perspective
  If a new project is created, change the active perspective to suit the project type
- Show new object
  If a single new object is created, select and reveal the new object in the appropriate view.
Preference Page – Creation

Create a preference page in the plug-in manifest
Preference Page – Declaration

- Declaration
  - Name
  - Identifier
  - Category (for nested pages)

- Class
  - Fields
  - Layout
  - Validation

- Field Editors
  Eclipse provides several field editors plus you can define your own
Preference Page – Field Editors

Eclipse provides several pre-built field editors

- **Boolean**
  - ![Boolean Field Editor](image)

- **Color**
  - ![Color Field Editor](image)

- **Directory**
  - ![Directory Field Editor](image)

- **Files**
  - ![Files Field Editor](image)

- **Font**
  - ![Font Field Editor](image)

- **Integer**
  - ![Integer Field Editor](image)

- **Scale**
  - ![Scale Field Editor](image)

- **String**
  - ![String Field Editor](image)

You are not limited to these… you can define whatever you like
Preference Page – Nested

- Root page shows product graphic and basic information
- Nested pages show additional detail
Preference API

- Access via plug-in class
  - MyPlugin.getInstance().getPluginPreferences()

- Simple key/value based storage
  - Boolean, Integer, String, etc

- Preference values
  - **Current** – each preference has a current value. same as default value if not specified
  - **Default** – each preference has a default value specified programmatically or using a file same as default-default value if not specified
  - **Default-Default** – hard coded value in the Eclipse preference system used if no current or default value is specified

- Specifying default values
  - Programmatically via initializeDefaultPluginPreferences() method
  - Using a file named “pref_store.ini” in the plug-in’s root directory

- Saving preferences
  - AbstractUIPlugin automatically saves preferences on plug-in shutdown
  - **Plugin class – must override stop(...) to explicitly store preferences**

- If you have more complex highly structured data, then store values in a file in your plug-in metadata directory

  ```java
  file = MyPlugin.getInstance().getStateLocation().append("data.xml").toFile();
  ```
Preferences – Ready for Rational

- Preferences dialog use
  Global options will be exposed within the Preferences dialog

- Preferences dialog misuse
  Expose the local options for a particular view, editor, or window in the view itself via a menu or tool item… not in a preference page
Property Pages

- Declaration
  - Object Type
  - Name
  - Identifier

- Class
  - Fields
  - Layout
  - Validation

Same as preferences, but for resources rather than entire workspace

Property pages can be reused as preference pages
Property View

Properties can be displayed in the properties view

Currently selected object must implement IPropertySource
Properties – Ready for Rational

- Properties views for quick access
  Use the Properties view to edit the properties when quick access is important

- Properties dialog for expensive calculations
  Use a Properties dialog to edit the properties that are expensive to calculate

- Properties dialog for complex relationships
  Use a Properties dialog to edit the properties of an object that have complex relationships to one another

- Properties dialog contains superset of items
  Properties dialogs should contain the superset of items shown in the properties view
Extension Points

Extension-points are an excellent way to provide structured decoupled product enhancements

- Plan for lazy loading and initialization
  Require basic static extension information in the manifest name, identifier, description, etc… so that the extending plug-in need not be loaded

- Object creation
  Use IConfigurationElement.createExecutableExtension(…) to create instances that are specified in plug-in manifest so that parameterization can occur

- Provide interface and base class
  Classes specified in plug-in manifest should implement some interface defined in your plug-in so that you can communicate with that instance in an implementation independent manner
  Provide an abstract base class for others to extend as a convenience

- Exception handling
  Wrapper use of IConfigurationElement.createExecutableExtension(…) in an exception handler that logs information about the exception and gracefully degrades
Extension Points

Extension and Extension Behavior

Extension Point and Abstract Base Class

Schema and Documentation
Extension Points – Schema

Create an extension-point schema and documentation so that consumers understand required elements

- Elements
- Name
- Identifier
- Icon
- Description
- …
Performance and Lazy Loading

- Plug-ins are loaded only when needed
  - Divide your product into chunks that can be lazily loaded
  - Use extension-points to decouple functionality
  - Provide as much information in plug-in manifest so plug-in need not be loaded until the last possible moment

- Current
  - Plug-ins loaded lazily but not unloaded until session ends

- Future
  - Plug-ins will be unloaded when no longer needed
  - What defines “no longer needed”?
Responsive UI

Push work out of the UI thread to improve the user experience

- **Jobs API**
  - Infrastructure for “running in background” dialog
  - “Always run in background” workbench preference
  - Specify root resource so jobs don’t collide
  - Specify priority so most important jobs get done first

- **Issues**
  - Deadlock – specific root resource when job is created
  - UI Thread – cannot update UI on job thread… use asyncExec(…)

- **Applies to:**
  - Long running Actions – interacting with external resource – CVS
  - View display code – label decorations calculated in background
  - Editor interaction – parse checking performed in background
Builders, Markers, Natures

Builders are an alternative to Actions if you want something to “always happen”

- Builders
  Perform some operation or conversion (e.g. compilation, validation, etc)

- Markers
  Used by the builder to report information to the user

- Natures
  Used to associate a project with one or more builders

But …. scoped to project, not workspace
Builders, Markers, Natures

- Declaration
- Nature
- Builder

- Nature
- Adds builder to project
- Removes builder from project

- Builder
- Performs operation
Marker Declaration

- Declaration
  - Name
  - Identifier
  - Description
  - Marker supertype

- Resolution
  - a.k.a. “Quick Fix”

- Marker Object
  
  Used by Builders, but can also be used elsewhere
Builders, Markers, Natures – Ready for Rational

- User builders to convert resources
  Any product that converts resources from one format into another where the resources are synchronized, such as compilers, must use the build APIs and builders extension-point

- Builders must be added by natures
  A builder must be added to a project by a nature

- Do not replace existing builders
  Plug-ins cannot replace the builders associated with project natures provided by the workbench or other vendors

- Do not misuse the term “build”
  The term “build” should not be overloaded to have a meaning other than the build processing triggered using the workbench build APIs

- Mark builder-created resources as “derived”
  Resources created by builders should be identified as derived when they are not source
  `IResource.setDerived(true);`

- Use IResourceProxy objects when possible
  Use IResourceProxyVisitor rather than IResourceVisitor when visiting all resources in a project
Resource Change Listeners

Alternative to actions or builders
if you want something to “always happen”

ResourcesPlugin.getWorkspace().addResourcesChangeListener(
    new IResourceChangeListener() {
        public void resourceChanged(event) {
            ... process event ...
        }
    }
)

Scoped to workspace not project

But… plug-in must be loaded for resource change listener to work

Ready for Rational - Requirement #1
Add an IResourceChangeListener to the workspace if you need to be aware of or react to changes in resources.
Early Startup

Sometimes you need aggressive startup… how?

- `org.eclipse.ui.startup` extension point
- Must implement `org.eclipse.ui.IStartup`
- Don’t use it unless you need it
- Plug-in should have small footprint and fast startup

What about plug-ins that *may* need early startup?

**Problem:**
if extension is specified then plug-in is always started

**Solution:**
create a plug-in to manage early startup
Early Startup

**Solution:** tiny plug-in that provides startup for others

Each plug-in can programmatically specify whether they need early startup.
Logging and Tracing

Need to turn on tracing in a deployed application… how?

1) Add tracing to your plug-in

   if (Platform.getDebugOption("com.instantiations.eclipse.startup/debug")) {
       ... log information ...
   }

2) Create .options file in your plug-in’s root directory

   - Contains the tracing options for your plug-in
   - Not read by system… for documentation purposes only

3) Modify primary .options file

   - Located in same directory as eclipse.exe file
   - Read by Eclipse on startup
   - Properties file format… key = value
   - Append key to turn on debugging… plug-in-id/debug-option key format

Sample .options file
com.instantiations.eclipse.startup/debug = true
com.instantiations.eclipse.startup/startup = true
com.instantiations.eclipse.startup/startupRequest = true
Logging and Tracing

Our tracing preference page
- Reads .options files for our plug-ins
- Modifies the primary .options file
Logging and Tracing – Ready for Rational

- Provide tracing
  
  Provide Eclipse run-time tracing support with product-specific filters

- Plug-in .options file
  
  Include an appropriate .options file in the directory for each plug-in where the .option file contains the various trace options.

- Provide error logging
  
  All plug-ins included as part of an Extension will support error logging through the Eclipse error logging facility.

- Error Message Numbering
  
  Implement an error message numbering convention to assist in the process of problem determination, isolation and resolution.
Internal Classes

Eclipse separates classes into two groups:

Public API
- Code maintained across multiple versions of Eclipse… 2.1, 3.0, 3.1, 3.2, 3.3, 3.4
- Compatibility layer maintained by Eclipse.org

Internal classes
- Package name contains “internal”
- Can change radically across versions of Eclipse

Suggestions:
- Stick to Public API whenever possible
- Consider another approach
- Ask on newsgroups for suggestions
- If no public API… submit code modification to Eclipse
Internal Classes – Access

If you need to access internal code:
- Call method directly
- Create utility class in same package
- Copy code into your own plug-in
- Subclass
- Create a plug-in fragment containing utility class

Only consider this as a short term fix of last resort.

Long term solution:
- Modify Eclipse itself in the smallest way possible
- Submit the code to Eclipse.org
- Include explanation
Internal Classes – Ready for Rational

Non-public Eclipse interfaces and classes may not be used without the approval of IBM. The use of internal interfaces and classes is discouraged as they are subject to change without notice.

Approval to use a non-public Eclipse interface or class will be given based on the pre-test report submitted with your Ready for IBM Rational software application.

In other words

- Use them at your own risk
- Don’t expect to obtain Ready for Rational certification with them unless you provide a good reason and submit changes to Eclipse so that in the future you can use some new public API
Help

Help provided using the native Eclipse help browser
Help Component

- Help books are HTML webs
- Extension points for contributing
  - entire books
  - sections to existing books
  - F1-help pop ups
- Eclipse Platform contributes
  - “Workbench User Guide”
  - “Platform Plug-in Developer Guide” (APIs)
  - F1-help for views, editors, dialogs, …
- JDT and PDE contribute their own help
- Help mechanisms available to all plug-ins

- Help search engine based on Apache Lucene
- Headless help server based on Apache Tomcat
Help – Structure

- Create help documentation in a separate plug-in
- Plug-in manifest references “toc” files

```xml
<extension point="org.eclipse.help.toc">
  <toc file="toc.xml" primary="true"/>
  <toc file="tocgettingstarted.xml"/>
  <toc file="tocreference.xml"/>
</extension>
```

- “toc” files (table of contents) reference documentation pages

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?NLS TYPE="org.eclipse.help.toc"?>
<toc label="Favorites Guide" topic="html/toc.html">
  <topic label="Getting Started">
    <anchor id="gettingstarted"/>
  </topic>
  <topic label="Reference">
    <anchor id="reference"/>
  </topic>
</toc>
```

- Zip documentation pages into “doc.zip” file located in plug-in root
Help – Internationalization

- Place translated “toc” files and documentation pages into specially named subdirectories
  
  nl/<language>
  
  nl/<language>/<country>

  where <language> and <country> represent two-letter codes for language and country

  Eclipse help system looks for “toc” files and documentation pages in order from most specific (language/country) to most general (plug-in root directory)

- Translated files may be contributed using fragments
Help – Infopop (F1)

Context sensitive help appearing when user presses F1
Help – Infopop Context ID

- Help context identifier
  `<plug-in-id>.<local-id>`

- Associate programmatically with view
  `WorkbenchHelp.setHelp(viewer.getControl(), "com.qualityeclipse.favorites.favorites_view");`

- Associate programmatically with actions
  `WorkbenchHelp.setHelp(copyAction, "com.qualityeclipse.favorites.copyAction");`

- Associate in plug-in manifest with actions
  `<action
    name = "Open Favorites View"
    ...`
  `helpContextId = "favorites_view"/>`
Help – Infopop Content

- Plug-in manifest points to context file
  
  ```xml
  <extension point="org.eclipse.help.contexts">
    <contexts file="contexts.xml" plugin="com.qualityeclipse.favorites"/>
  </extension>
  ```

- Contexts file
  
  ```xml
  <contexts>
    <context id="favorites_view">
      <description>This is the Favorites view.</description>
      <topic href="html/gettingstarted/favorites_view.html" label="Using the Favorites View"/>
      <topic href="html/gettingstarted/installation.html" label="Installing the Favorites View"/>
      <topic href="html/reference/preferences.html" label="Favorites View Preferences"/>
    </context>
    <context id="copyAction">
      <description>This command copies a Favorites item from the view.</description>
      ... etc ...
    </context>
  </contexts>
  ```

Infopop content for “Open Favorites View” action
Help – Programmatic Access

- `setHelp (control, context-id)`
  Associate help context with a particular UI control

- `setHelp (action, context-id)`
  Associate help with a particular action

- `displayHelp ()`
  Open the Eclipse help browser

- `displayHelp (context-id)`
  Display context sensitive help for the given context
Help – Cheat Sheets

- Walk the user through a series of required and optional steps
- Automatically launch appropriate tools and actions
- Create using the org.eclipse.ui.cheatsheets.cheatSheetContent extension-point
Help – Ready for Rational

- Existing documentation is not modified
  Do not delete or replace existing documentation during the installation of your product into an existing workbench install

- Provide help through the workbench help system
  Help for the plug-in’s workbench UI should be provided through the workbench help system integrated into the workbench.

- Implement active help
  Provide links rather than instructions when possible.
  For example, rather than telling the user
  select File > Import
  then select External Plug-ins and Fragments,
  provide a link called
  “Click here to open the Import External Fragments wizard.”

- Use Cheat Sheets properly
  Provide an overview of tasks.
  Illustrate only one task per step.
  Provide a help link with each step.
Lab 3 – Help

- Select the Category “EclipseCon 2008 Labs”
- Select “Lab 2 - Views”
- Click the import toolbar button

At any time during the lab, you can compare your work with the expected result.
Lab 3 – Create Help – 1

Goal: Create a help page for the Sample view

- Select File > New > Project... to open the New Project wizard
- In the New Project wizard, select Plug-in Project and click Next
Lab 3 – Create Help – 2

- Enter “lab.help” in the Project name field, then click Next

- Change the Plug-in Name to “Lab Help Plug-in”, then click Next
Lab 3 – Create Help – 3

- Select **Custom plug-in wizard**, then click **Next**

- Select only the **Help Table of Contents** choice, then click **Next**
Lab 3 – Create Help – 4

- Select the **Primary** checkbox, then click **Finish**
Lab 3 – Create Help – 5

- When the wizard closes, the `lab.help` project is created and contains the following files and directories

  (a) **html** directory - holds all html documentation page files

  (b) **toc.xml** file - the primary help topic entry for this help plug-in

  (c) other **toc*.xml** files - the help sub-topic entries for this help plug-in
Lab 3 – Test Help – 1

**Goal:** Test the help created in the prior section

- Launch the runtime workspace
- Open the Eclipse **Help** window by selecting **Help > Help Contents**
Lab 3 – Test Help – 2

- Select the **Sample Table of Contents** to see the new help content

- Close the **Help** window and runtime workbench
Lab 3 – Completed

- Compare your work with the expected result
- Select “Lab 3 - Help”
- Click Compare to see differences
Internationalization

- Eclipse Platform is internationalized
- Many translations available including the following languages
  - English
  - German
  - Spanish
  - Italian
  - French
  - Portugese (Brazil)
  - Japanese
  - Korean
  - Chinese (Traditional)
  - Chinese (Simplified)
- Translations live in plug-in fragments
  - Separately shippable
- Internalization mechanisms available to all plug-ins
Internationalization – Plug-in Manifest

Plug-in manifest

```xml
<plugin
    id="com.qualityeclipse.favorites"
    name="%favorites.plugin.name"
    version="1.0.0">

    <extension point="org.eclipse.ui.views">
        <category
            name="%favorites.category.name"
            id="com.qualityeclipse.favorites"/>
        <view
            name="%favorites.view.name"
            icon="icons/sample.gif"
            ... etc ...
    </extension>

plugin.properties

# Contains translated strings for the Favorites plug-in
favorites.plugin.name=Favorites Plug-in
favorites.category.name=Quality Eclipse
favorites.view.name=Favorites

Translated files should be named

plugin_<language>.properties
    - or -
plugin_<language>_<country>.properties

Standard German file would be named:
plugin_de.properties

Standard French file would be named:
plugin_fr.properties
Internationalization – String Extraction

Eclipse includes a powerful string externalization tool that will do most of the work

private static final String CUT = “Cut”;

is replaced with

private static final String CUT = ViewMessages.getString(
    “FavoritesView.Cut”); //$NON-NLS-1$
Internationalization – String Extraction

Think carefully about how you write your code, since it can cause more strings that need externalization than is strictly necessary.

- Replace single-character strings with single characters
- Look for opportunities to reuse keys rather than creating new ones
- Use message binding to allow more flexibility during translation

Message binding example:

```java
// Bad, we don’t want to externalize “Count (“
label.setText(“Count (“ + count + “)“);

// Good, we can externalize “Count”
label.setText(“Count” + “ (“ + count + ‘)’); //$NON-NLS-2$

// Better, use binding patterns where possible.
label.setText(MessageFormat.format(“Count (%1)“, new String[] {count}))
```

“Count (%1)” in one language may be “the (%1) count” in another
Internationalization – String Extraction

Translated properties files should be named

<basename>_<language>_<country>.properties
- or -
<basename>_<language>_<country>.properties

where <language> and <country> represent the two-letter codes used to signify the language and country, and <basename> is the Name of the original properties file.

For example:

ViewMessages.properties
ViewMessages_de.properties German translation
ViewMessages_fr.properties French translation
Internationalization – Other Resources

Other resources such as

- HTML
- XML
- INI

are placed into

nl/<language>
- or -
nl/<language>/<country>

directories, where <language> and <country> represent the two-letter codes used to signify the language and country
Internationalization – Tips

- To make sure that you have externalized all your plug-in’s strings change “Usage of non-externalized strings” option from “Ignore” to “Warning”.

- Codes can be found at
  - Language codes www.unicode.org/onlinedat/languages.html
  - Country codes www.unicode.org/onlinedat/countries.html

- Translated files can be placed into plug-in fragments to further reduce program size

- If the fragment JAR file contains resources (and no classes), optimization is possible by adding type = “resource”. For example:

  <library name="nl1.jar" type="resource"/>

Features and Branding

A Feature …

- Ties together multiple plug-ins as a single unit
- Provides product branding such as
  - About pages
  - Splash screen
- Installable using the Update Manager
Product Information

Window image

Splash screen

Welcome pages

About product info

About feature info
Product Information

- Primary feature controls product information
  - Splash screen
  - Window image
  - About product info
  - Initial welcome page
  - Default perspective
  - Preference default overrides

- All features can provide
  - Welcome page
  - About feature info
Update Manager

- Can be used to discover, download and install features and plug-ins
- Supported by modular nature of Eclipse
  - Makes it easy to install additional features and plug-ins, and to update existing ones
- Installing or updating features and plug-ins requires adding files to Eclipse
  - Deleting files is never required as different versions of plug-ins and features can co-exist in Eclipse.
Features and Branding – Ready for Rational

- **Do not override product branding**
  Extension installations should not override the existing product branding either by overwriting the existing product branding feature or by launching using the Eclipse feature option.

- **Branded feature visibility**
  At least one branded feature must be visible in the About product and About product Features dialogs.

- **Include attribution information**
  A provider’s features and plug-ins must include appropriate attribution information (company name, version ID, name) in the attribution dialogs started using the Feature Details and Plug-in Details… buttons found on the About product dialog.

- **about.html file contents**
  The plug-in must include an about.html file in the plug-in install directory containing at least:
  a. The Eclipse attribution, using the following text and URL:
     “This offering is based on technology from the Eclipse Project” http://www.eclipse.org
  b. Any attribution required by any dependent technology used by the plug-in (as defined by the provider of that technology)
  c. Any other legal information that the provider is mandated to provide

- **Splash screen restrictions**
  The display of a splash screen for a feature is restricted to when the software is installed in an evaluation or demonstration mode. Once a license is purchased, the software must be automatically modified during the application of the license to remove the display of a second-level splash screen.
Building your product

- Use Ant to automate the build process
- Break the Ant script into separate targets based on supported versions of Eclipse
- Headless Ant Example

```java
java -cp \eclipse\startup.jar org.eclipse.core.launcher.Main -clean -noupdate -application org.eclipse.ant.core.antRunner -data \eclipse\workspace -verbose -file headless_test.xml %* >headless_out.txt 2>&1
```
Deploying your product into an existing installation

**Option #1:**
- Place your product directly into the Eclipse directory

**Option #2:**
- Place your product into separate directory
- Create link file telling Eclipse where to find your product
  - “links” subdirectory
  - File extension “.link”
- Tip: create file of reverse-links pointing back to Eclipse
- Ready for Rational requirement
Supporting multiple versions of Eclipse

Why not ship a single binary if

- Public API used by your plug-in has not changed
- Code compiles against each version successfully

Because there are

- Hidden runtime compatibility issues
Hidden runtime compatibility issues

Your code:
public void yourMethod () {
    foo(0);
}

Eclipse 3.3 API:
public void foo (int value) {
    ... some operation ...
}

Eclipse 3.4 API:
public void foo (long value) {
    ... some operation ...
}

Code compiled against one Eclipse ...
... will not run in another
Building against multiple versions of Eclipse

Approach #1: Code Branches
or one project per version of Eclipse

- Standard approach
- Everything compiled

But

- No single code base
- Lots of merging
Building against multiple versions of Eclipse

Approach #2: Façade Pattern

- Shared code
- Single binary

But

- Related functionality not together
- Hidden runtime compatibility issues
Building against multiple versions of Eclipse

Approach #3: Preprocessor used during build process

- Related functionality kept in close proximity
- One binary per version of Eclipse
- Shared code
PreProcessor

Source translated during build before compilation

Related code kept in close proximity

Eclipse 3.1 specific code

Eclipse 3.0 specific code
Fragments

Can help smooth differences between versions of Eclipse
For example:

- Backported two jobs classes
- APIs are the same… internals are different
Classpath Translation Tools

Specialized Ant task for translating a project’s classpath so that it can be compiled against a different version of Eclipse at build time

For example:

```xml
<eclipsetools.getclasspath
  buildmap="WSAssistSource"
  config="eclipse34"
  binroot="${build.bin.34}"
  property="classpath34"/>
```

For download and more information, see http://www.qualityeclipse.com/
Ready for Rational Wrap-up

- What to expect
  - 2 – 8 weeks of work depending on size of application
  - Lots of back and forth to work out details
  - Potentially lots of work to clean up internal API usage
- Prepare and submit document well in advance
  - Requires at least a week of work
  - Provide as much detail as possible
  - Include lots of screen shots
  - Will likely require several iterations
- Go into live session with “no surprises”
  - Document everything
  - If you have any questions or issues, ask questions well in advance
  - Do a dry run of everything you plan to show the day before
instantiations
Build Quality Software

Leading-edge tools for professional Java developers

WindowBuilder Pro
RCP Developer
CodePro AnalytiX