Commands in Eclipse: some advanced patterns
About the Speaker

- Paul Webster
  - Senior Software Developer, Platform UI Committer
  - Works for IBM Rational Software since he joined the Eclipse™ Platform UI team in May 2005
  - Works with part lifecycle, commands/handlers, keybindings, and menu contributions
  - Eclipse 4 and the 4.2 Workbench
Overview

- Overview of the technical implementation of the Command Framework
- Relationship between commands, handlers, and menu contributions
- Contributing to the application state
- Common patterns used when exposing functionality to the user using commands
Commands

- A command is an abstraction of some semantic behaviour.
- A command is not an implementation of that behaviour.
- A command is not the visual presentation of that behaviour.

Refresh

- RefreshFiles (activePartId == org.e.u.navigator)
- RefreshConnections (activePartId == org.e.dtp.databases)
Commands - State

- A command provides its enabled state: isEnabled()
- The state of the command is the state of its active handler
- Commands also allow arbitrary state to be attached to them
Handlers

- A handler implements one behaviour for a command.
- The active handler controls the command’s enabled state.
- Handlers are provided an application context in their execute(*) method.

```
Commands ——> Handlers
  |
  |
  |
Refresh ——> RefreshFiles (activePartId == org.e.u.navigator)
  |
  |
  |
Refresh ——> RefreshConnections (activePartId == org.e.dtp.databases)
```
Handlers – Pick One

- The 2 most common kinds of handlers:
  - Global: think Exit or About
  - Local: think Refresh or Print

- One or zero are chosen
Say we want the refresh command to refresh information in the active view:

- Command Service
- Handler Service
- Application Context
- Source
  - activePartId

Refresh

- RefreshFiles
  - (activePartId == org.e.u.navigator)

- RefreshConnections
  - (activePartId == org.e.dtp.databases)
Keybindings

- A keybinding allows a key sequence to execute a command:
  - F5: Refresh
  - ALT+SHIFT+Q, C: Show View (Console)
- A keybinding might execute a different command depending on the state of the application
Menu Contributions

- Menus and Toolbars are important for exposing functionality to the user.
- In Eclipse, menu contributions specify the presentation of commands to the user.
Services

- All programmatic interaction with the menu/command framework is done through services.
- These are not OSGi™ services.
Variable Sources

- The global application context is kept up to date by various source providers, managed by the IEvaluationService.
Pattern 2: Using a parameter

- Part of the command definition
- Passed to the handler
- From a menu item or key shortcut

```java
public Object execute(ExecutionEvent event) throws ExecutionException {
    String parameter = event.getParameter(VIEW_ID);
    System.out.println("Show Label: "+ parameter);
    return null;
}
```
Pattern 3: The toggle

- You want a toggle command when you want the user to have a visual cue to the state of the system: Build Automatically

- The system expects the toggle state to use the `org.eclipse.ui.commands.toggleState` ID.

```xml
<command categoryId="org.eclipse.ui.category.edit"
    defaultHandler="org.eclipse.example.commands.MonitorCountHandler"
    id="org.eclipse.example.commands.monitorCount"
    name="Monitor Count">
    <state class="org.eclipse.ui.handlers.RegistryToggleState:false"
        id="org.eclipse.ui.commands.toggleState"/>
</command>
```

- It's the handler's responsibility to change the command state

```java
public Object execute(ExecutionEvent event) throws ExecutionException {
    Command command = event.getCommand();
    boolean shouldStop = HandlerUtil.toggleCommandState(command);
    // do something
    return null;
}
```
Pattern 4: Choices using radio buttons

- Providing the user with a small set of choices to pick from is commonly done with radio buttons
- The command needs the radio state
- The command also needs a parameter (to identify a state)

```xml
<command defaultHandler="org.eclipse.example.commands.ViewPageHandler"
    id="org.eclipse.example.commands.viewPage"
    name="View Page">
    <commandParameter id="org.eclipse.ui.commands.radioStateParameter"
        name="State" optional="false"/>
    <state class="org.eclipse.ui.handlers.RadioState:page"
        id="org.eclipse.ui.commands.radioState"/>
</command>
```
Pattern 5: Extending a toolitem with a dropdown

- Usually a command that takes a parameter
- The command contribution to the toolbar should specify an id
- The dropdown is filled in using menu:id

```xml
<command id="org.eclipse.example.commands.perspectiveList"
    commandId="org.eclipse.example.commands.setPerspective"
    icon="images/editor_area.gif"
    style="pulldown">
</command>

//..........
<menuContribution allPopups="false"
    locationURI="menu:org.eclipse.example.commands.perspectiveList">
// fill in dropdown
</menuContribution>
```
Pattern 6: A dynamic section of a menu

- You can place a dynamic contribution in any menu
- Provide a list of 0 or more contributions when the menu is shown
- IWorkbenchContribution can supply you with an IServiceLocator

```xml
<menuContribution locationURI="menu:org.eclipse.example.commands.perspectiveList">
  <dynamic class="org.eclipse.example.commands.DynamicPerspectiveList" id="org.eclipse.example.commands.perspectiveList.dynamic"/>
  <separator name="other" visible="true"/>
  // more contributions
</menuContribution>
```

```java
CommandContributionItemParameter parm = new CommandContributionItemParameter(locator, desc.getId(), SET_PERSP_ID, CommandContributionItem.STYLE_PUSH);
HashMap<String, String> cmdParms = new HashMap<String, String>();
    cmdParms.put(SetPerspectiveHandler.PERSP_ID, desc.getId());
parm.parameters = cmdParms;
parm.icon = desc.getImageDescriptor();
parm.label = desc.getLabel();
return new CommandContributionItem(parm);
```
Pattern 7: Operating on a selected object

- The context menu is a convenience whose focus is the selected object(s)
- In most cases this will be an IStructuredSelection … any analysis must iterate through the selection.
- The expression can be applied to activeWhen, enabledWhen, or visibleWhen
- When dealing with the Package Explorer, one must adapt

```xml
  <with variable="activeMenuSelection">
    <count value="++"/>
    <iterate ifEmpty="false">
      <adapt type="org.eclipse.core.resources.IResource">
        <test property="org.eclipse.core.resources.contentTypeId"
          args="kindOf"
          value="org.eclipse.core.runtime.text"/>
      </adapt>
    </iterate>
  </with>
```
Pattern 8: The search bar

- Simple controls can be placed in a toolbar
- The contribution is filling in a Composite
- The contribution has to be prepared to deal with orientation

```xml
<menuContribution locationURI="toolbar:org.eclipse.ui.main.toolbar?
        after=org.eclipse.example.commands.toolbar">
    <toolbar id="org.eclipse.example.commands.searchToolbar"
        label="Command Search Toolbar">
        <control id="org.eclipse.example.commands.search"
            class="org.eclipse.example.commands.SearchContribution"/>
    </toolbar>
</menuContribution>
```
Pattern 9: The property tester

- Variables exposed through the system are objects
- Sometimes we need to examine more attributes in activeWhen, enabledWhen, and visibleWhen
- Property testers allow properties of those objects to be examined
- They also allow more complex tests than the XML permits

```
<propertyTester
  id="org.eclipse.core.resources.filePropertyTester"
  class="org.eclipse.core.internal.propertytester.FilePropertyTester"
  namespace="org.eclipse.core.resources"
  properties="contentTypeTypeId"
  type="org.eclipse.core.resources.IFile"/>
// then it is used
<test property="org.eclipse.core.resources.contentTypeId"
  args="kindOf"
  value="org.eclipse.core.runtime.text"/>
```
Pattern 10: Command Framework interactions

- We want our contributions to be able to access functionality
- We want our interactions to keep the application state up to date

```
public Object execute(ExecutionEvent event) throws ExecutionException {
    IWorkbenchWindow window = HandlerUtil.getActiveWorkbenchWindowChecked(event);
    ICountService service = (ICountService) window.getService(ICountService.class);
    System.out.println("New value: "+ service.incrementCount());
    return null;
}
```
More Information

- You can get general information from the wiki:
  - http://wiki.eclipse.org/Command_Core_Expressions
  - http://wiki.eclipse.org/Menu_Contributions

- You can see examples of this:
  - http://github.com/paulweb515/eclipseExamples

- You can contact us and ask questions:
  - On IRC: irc://freenode.net/#eclipse
  - On the newsgroup: http://www.eclipse.org/forums/
    Eclipse Projects > Eclipse Platform
    Eclipse Projects > Rich Client Platform
Give Feedback on the Sessions

1. Sign In: www.eclipsecon.org

2. Select Session Evaluate

3. Vote
   -1  0  +1
Legal Notices

- Copyright © IBM Corp., 2007-2012. All rights reserved. This presentation and the source code in it are made available under the EPL, v1.0.

- Eclipse and the Eclipse logo are trademarks of Eclipse Foundation, Inc.

- IBM and the IBM logo are trademarks or registered trademarks of IBM Corporation, in the United States, other countries or both.

- Other company, product, or service names may be trademarks or service marks of others.

- THE INFORMATION DISCUSSED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, AND IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, SUCH INFORMATION. ANY INFORMATION CONCERNING IBM'S PRODUCT PLANS OR STRATEGY IS SUBJECT TO CHANGE BY IBM WITHOUT NOTICE.