Building Eclipse Plug-Ins

Dan Rubel, Chief Technology Officer
Eric Clayberg, VP of Product Development
Overview (condensed from a 6 hour tutorial)

Topics
> Introduction
> Eclipse Architecture
> Commands
> Views
> Editors
> Perspectives
> Modal Windows
> Advanced Topics
> Help and Internationalization
> Features, Branding, & Update
> Build and Deploy
> Ready for Rational

Labs
> Hello World
> Commands
> Views
> Perspectives
> Preference Pages
> Help

Content derived from Eclipse Plug-ins (3rd edition)
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 Plug-ins* (3rd edition Dec 2008)
- Co-developers & architects of VA Assist Enterprise, CodePro and over a dozen other commercial software products
Founded in 1997, headquarters in Portland, OR

Market leading tools for professional Java developers
- Automated Software Code Quality and Security
- UI Construction and Testing Tools

9 years of extensive Eclipse experience
- Active Eclipse Foundation member and major contributor.

Leveraging the best of open source and proprietary technologies to enable the creation of better enterprise software

10,000+ Customers. Customer focused. Customer driven.
Awards and Acknowledgements

2006 - 2008
Private 100
Fastest Growing
Oregon
Companies

#14 Fastest
Growing
Technology
Company
2006

“Because our software is used to drive key business decisions, it’s critical that we deliver high-quality software, and implementing CodePro AnalytiX has allowed us to automate the process of enforcing programming standards and best practices.”
Rich Main, Director,
Java Development Environments, SAS

“WindowBuilder delivers the kind of GUI building productivity that we use to have before we converted to Java.”
Sally Rich
Checkfree Solutions

“RCP Developer is the first product to bring comprehensive application construction, GUI testing and packaging of rich-client applications to Eclipse RCP. We are delighted that Instantiations is delivering this important technology to the growing Eclipse ecosystem.”
Mike Milinkovich, Executive Director
Eclipse Foundation

“Since we started using WindowTester, tests that took 2–3 weeks to write previously can now be done in 2–3 days.”
Steve Tocco
Director of Quality Assurance, BEA
Strategic Product Lines

**CodePro AnalytiX™**
Automated Software Quality and Security Tools for Code Analysis, Measurement and Improvement

**WindowBuilder™ Pro**
WYSIWYG development of Java GUIs for SWT, Swing and GWT

**WindowTester™ Pro**
Automated testing of Java GUIs for SWT and Swing
Professional Services

> One of the most experienced teams of Eclipse, Java and Smalltalk experts

> Planning assistance for build and deployment

> QuickQuality live mentoring boot-camp

> Architectural, design and development mentoring

> Custom software engineering

“The members of the professional services team at Instantiations are truly top-tier technologists. Not only did they integrate right into our team, bringing impressive experience and mentoring skills to bear, they brought along enough big-picture aptitude to understand our business problems, our pressures, and what project success meant for us.”

Jesse Watson
Manager, Engineering
Varolii Corporation
Eclipse Plug-ins (3rd edition)

Publisher: Addison Wesley
http://www.qualityeclipse.com/
928 pages
Authors:
Eric Clayberg
Dan Rubel
Series Editors:
Erich Gamma
Lee Nackman
John Wiegand
Forwards by:
Skip McGaughey
Simon Archer

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Using Eclipse Tools</td>
</tr>
<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>Commands &amp; 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>Building a Product</td>
</tr>
<tr>
<td>20</td>
<td>GEF</td>
</tr>
<tr>
<td>21</td>
<td>Advanced Topics</td>
</tr>
</tbody>
</table>
Eclipse Workbench
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
What's in a plug-in?

- **META-INF/MANIFEST.MF**
  - Describes plug-in dependencies

- **plugin.xml**
  - Describes plug-in extensions

- **plugin.properties**
  - Extracted translatable strings

- **A JAR file**
  - An archive with the plug-in code

- **about.html**
  - Textual description of the plug-in

Plug-ins can be packaged as directories (old) or jar files (new)
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
Commands
Commands

- One or more menu contributions per command
- One or more handlers per command
- Only one “active” handler at a time

```java
public class PasteHandler extends AbstractHandler {
    public Object execute(ExecutionEvent event)
        throws ExecutionException {
        // Perform Paste operation
        return null;
    }
}
```
Commands

Menu Contribution

UI Representation

Cut Ctrl-X
Copy Ctrl-C
Copy Qualified Name
Paste Ctrl-V
Delete Delete
Select All Ctrl-A

Category

Command

"Paste"

Handler

Concrete Behavior

```java
public class PasteHandler extends AbstractHandler {
    @Override
    public Object execute(ExecutableEvent event) throws ExecutionException {
        return null;
    }
}
```

Keybinding

"Ctrl-V"
Eclipse Plug-ins (3rd edition)

**Category**

**Command**

**Menu**

**Handler**

**User Interface**

```xml
<extension point="org.eclipse.ui.commands">
  <category name="Favorites">
    <description>Commands related to the Favorites View</description>
    <id name="com.qualityeclipse.favorites.commands.category"/>
  </category>
  <command categoryId="com.qualityeclipse.favorites.commands.category">
    <name>Open Favorites View</name>
    <description>Open the Favorites view if it is not already visible</description>
    <id name="com.qualityeclipse.favorites.commands.openView"/>
  </command>
</extension>

<extension point="org.eclipse.ui_menus">
  <menuContribution locationURI="menu:org.eclipse.ui.main.menu?after=toolbars">
    <menu id="com.qualityeclipse.favoritesmenus.favoritesMenu">
      <label>Favorite</label>
      <mnemonics='U'>
        <command commandId="com.qualityeclipse.favorites.commands.openView">
          <id name="com.qualityeclipse.favoritesmenus.openFavoritesMenu"/>
          <icon name="images/sample.gif"/>
          <mnemonic='O'>
        </command>
      </mnemonics>
    </menu>
  </menuContribution>
</extension>

<extension point="org.eclipse.ui_handlers">
  <handler commandId="com.qualityeclipse.favorites.commands.openView">
    <class name="com.qualityeclipse.favoriteshandlers.OpenFavoritesViewHandler"/>
  </handler>
</extension>

public class OpenFavoritesViewHandler extends AbstractHandler {
  public Object execute(ExecutionEvent event) throws ExecutionException {
    // Open the favorites view...
    return null;
  }
}

```

Image of Eclipse UI with menu and action selection.
Menu

Menu Contribution

UI Representation

<table>
<thead>
<tr>
<th>Action</th>
<th>Keybinding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Ctrl+X</td>
</tr>
<tr>
<td>Copy</td>
<td>Ctrl+C</td>
</tr>
<tr>
<td>Copy Qualified Name</td>
<td>Ctrl+Shift+C</td>
</tr>
<tr>
<td>Paste</td>
<td>Ctrl+V</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete</td>
</tr>
<tr>
<td>Select All</td>
<td>Ctrl+A</td>
</tr>
</tbody>
</table>

Category

Command

Abstract Binding Point

"Paste"

Handler

Concrete Behavior

```java
public class PasteHandler extends AbstractHandler {
    public Object execute(Object exe) throws ExecutionException {
        // Perform Paste operation
        return null;
    }
}
```

Keybinding

“Ctrl-V”
Menu Contribution

- locationURI
- visibleWhen
- expression
Menu Contribution – locationURI

menuContribution locationURI="[scheme]:[id]?[argument-list]"

To specify the Eclipse menu bar:

menu:org.eclipse.ui.main.menu?after=additions

The Eclipse menu bar unique ID

The location within the menu bar where the contribution should be added
Menu Contribution – locationURI

menuContribution locationURI="[scheme]:[id]?[argument-list]"

menu:org.eclipse.ui.main.menu?after=additions

**scheme** = the 'type' of the UI component into which the contributions will be added

- menu = the main application menu or view pull-down menu
- popup = a context menu in view or editor
- toolbar = the main application toolbar or toolbar in view
Menu Contribution – locationURI

menuContribution locationURI="[scheme]:[id]?[argument-list]"

menu:org.eclipse.ui.main.menu?after=additions

id = the unique identifier of menu, popup or toolbar into which the contributions should be added

- Eclipse main menu: org.eclipse.ui.main.menu
- Eclipse main toolbar: org.eclipse.ui.main.toolbar
- “Any” context menu: org.eclipse.ui.popup.any

Convention: view pull-down menu id = view context menu id = view id
Menu Contribution – locationURI

menuContribution locationURI=“[scheme]:[id]?[argument-list]”

argument-list = allows fine-grained definition of the specific location within a given menu, popup or toolbar

placement
• “before”
• “after”

the identifier for some item in the menu, popup or tool or “additions”
Menu Contribution – Top Level

toolbar:org.eclipse.ui.main.toolbar?after=additions

menu:org.eclipse.ui.main.menu?after=additions

Add to “Window” menu
menu:window?after=additions

Add to “Help” menu
menu:help?after=additions

Menu item to appear in “all” context menus

popup:org.eclipse.ui.popup.any?after=additions
Menu Contribution – View

menu:org.eclipse.jdt.ui.MembersView?after=additions

toolbar:org.eclipse.jdt.ui.MembersView?after=additions

popup:org.eclipse.jdt.ui.MembersView?after=additions

Menu item to appear in “all” context menus

popup:org.eclipse.ui.popup.any?after=additions
Menu Contribution – visibility

visibleWhen = an expression controlling visibility of a menu or menu item

```xml
<menu ... >
  <visibleWhen>
    <with variable="selection">
      <iterate operator="or" ifEmpty="false">
        <instanceof value="org.eclipse.core.resources.IResource" />
      </iterate>
    </with>
  </visibleWhen>
</menu>
```

By default, menus and menu items are always visible.

To reduce clutter and improve usability of an application, include a visibleWhen expression
Menu Contribution – visibility

visibleWhen = an expression controlling visibility of a menu or menu item

```xml
<menu ... >
  <visibleWhen>
    <with variable="selection">
      <iterate operator="or" ifEmpty="false">
        <instanceof value="org.eclipse.core.resources.IResource" />
      </iterate>
    </with>
  </visibleWhen>
</menu>
```

Show this menu when the current selection contains an instance of IResource.
Handler

Menu Contribution

UI Representation

<table>
<thead>
<tr>
<th>Action</th>
<th>Keybinding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Ctrl+X</td>
</tr>
<tr>
<td>Copy</td>
<td>Ctrl+C</td>
</tr>
<tr>
<td>Copy Qualified Name</td>
<td>Ctrl+C</td>
</tr>
<tr>
<td>Paste</td>
<td>Ctrl+V</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete</td>
</tr>
<tr>
<td>Select All</td>
<td>Ctrl+A</td>
</tr>
</tbody>
</table>

Category

Command

Abstract Binding Point

"Paste"

Handler

Concrete Behavior

```java
public class PasteHandler extends AbstractHandler {
    public Object execute(ExecutionEvent event) throws ExecutionException {
        // Perform Paste operation
        return null;
    }
}
```
Handler

Zero or more handlers associated with a command

At most one **active** handler associated with a command

If two handlers have conditions indicating that they **could** be active, then the conditions are compared to determine the more specific handler
Handler – top level edit menu

- **Cut** = org.eclipse.ui.edit.cut
- **Copy** = org.eclipse.ui.edit.copy
- **Paste** = org.eclipse.ui.edit.paste
- **Delete** = org.eclipse.ui.edit.delete
Handler Declaration

global “cut” command

<handler
    commandId="org.eclipse.ui.edit.cut"
    class="com.qualityeclipse.favorites.handlers.CutFavoritesHandler">

<activeWhen>
    <with variable="activePartId">
        <equals value="… favorites view id …"/>
    </with>
</activeWhen>

<enabledWhen>
    <with variable="selection">
        <count value="+"/>
    </with>
</enabledWhen>

</handler>

```java
public class CutFavoritesHandler extends AbstractHandler {
    public Object execute(ExecutionEvent event) throws ExecutionException {
        copy.execute(event);
        remove.execute(event);
        return null;
    }
}
```
Handler Declaration

<handler
    commandId="org.eclipse.ui.edit.cut"
    class="com.qualityeclipse.favorites.handlers.CutFavoritesHandler">
    <activeWhen>
        <with variable="activePartId">
            <equals value="… favorites view id …"/>
        </with>
    </activeWhen>
    <enabledWhen>
        <with variable="selection">
            <count value="+"/>
        </with>
    </enabledWhen>
</handler>

Handler should only be called when the favorites view is active

Handler should only be enabled when one or more elements are selected
public class CopyFavoritesHandler extends AbstractHandler {
    public Object execute(ExecutionEvent event) throws ExecutionException {
        ISelection selection = HandlerUtil.getCurrentSelection(event);
        if (selection instanceof IStructuredSelection) {
            ... process ...
        }
        // HandlerUtil provides many useful utility methods:
        > getActiveContexts
        > getActiveEditor
        > getActivePart
        > getActiveShell
        > getActiveWorkbenchWindow
        > getCurrentSelection
        > getVariable
        // When handler is executed, application state may have changed so obtain application state information such as selection, modifiers, variables and the like from the event using HandlerUtil
Key Bindings
Key Bindings

```xml
<key
    commandId="com.qualityeclipse.favorites.commands.add"
    contextId="org.eclipse.ui.textEditorScope"
    schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"
    sequence="Ctrl+Shift+A">
  Key sequence consisting of one or more keystrokes
</key>
```

Special keys are represented by:

> ARROW_DOWN, ARROW_LEFT, ARROW_RIGHT, ARROW_UP
> BREAK, BS, CAPS_LOCK, CR, DEL, END, ESC
> F1, F2, F3, F4, F5, F6, F7, F8, F9, …
> NUMPAD_0, NUMPAD_1, NUMPAD_2, NUMPAD_3, NUMPAD_4, …
> …
Views
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 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 Commands

Recommendation:
Add separators (groups) to menus programmatically,
then add commands using the locationURI’s discussed earlier

Three places commands can appear:

Separators / Groups:
- local
- global
- additions
- other

context menu, pulldown menu
toolbar buttons,
View Context Menu – Dynamically Built

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) {
        menuMgr.add(new Separator("local"));
        menuMgr.add(new Separator("global"));
        menuMgr.add(new Separator(IWorkbenchActionConstants.MB_ADDITIONS));
        menuMgr.add(new Separator("other"));
    }
});
```
View Tool Bar

Recommendation:
Add separators (groups) to toolbars programmatically, then add commands using the locationURI’s discussed earlier

```
IToolBarManager toolBarMgr = 
    getViewSite().getActionBars().getToolBarManager();
toolBarMgr.add(new GroupMarker("local"));
toolBarMgr.add(new Separator(
    IWorkbenchActionConstants.MB_ADDITIONS));
```

Not built dynamically
View Pulldown Menu

Recommendation:
Add separators (groups) to pulldown menu programmatically, then add commands using the locationURI’s discussed earlier

Code similar to code for context menu

Not built dynamically
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
  public void stop(BundleContext context) {
    … save information …
  }
> ISaveParticipant for snapshots throughout Eclipse session
  public void saving(ISaveContext context) {
    … save information …
  }
View – Managing OS Resources

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

Motto: If you create it, then you dispose it

Some resources are provided for you
  • Workbench images, System fonts and colors
  • Eclipse ensures proper disposal – do not dispose yourself
Linking Views – Selection Provider

Provide selection to others

- `getSite().setSelectionProvider(viewer);`
- `viewer -or- other object implementing ISelectionProvider`

Provide adaptable objects

```java
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 – IAdapterManager

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

Unregister Adapter Factory when the plug-in shuts down
```
Platform.getAdapterManager().unregisterAdapters(...);
```
Adapter Overview

Favorites View

Selection
Favorites Object

getAdapter()

IResource

Package Explorer

Selection
IResource

getAdapter()

Favorites
Adapter
Factory

Favorites Object

IResource
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` ...
Editors
# Views versus Editors

Views and Editors differ in purpose and lifecycle

<table>
<thead>
<tr>
<th>Purpose</th>
<th>View</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display information</td>
<td>Modify information</td>
<td></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 Declaration

- Declaration
  - Name
  - Identifier

- Editor
  - Edit Controls
  - Context Menu

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

Add special group so others can contribute actions

```java
menuMgr.add(new Separator(
   IWorkbenchActionConstants.MB_ADDITIONS));
```

Separators / Groups:

- local
- global
- additions
- other
Editor Context Menu – Dynamically Built

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) {
        menuMgr.add(new Separator("local"));
        menuMgr.add(new Separator("global"));
        menuMgr.add(new Separator(
            IWorkbenchActionConstants.MB_ADDITIONS));
        menuMgr.add(new Separator("other"));
    }
});
```
**Editor Contributor**

Manages tool bar buttons, menus and menu items

> **init(…)**
  - Called when first created
  - Initialize contributor

> **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(…)

![Eclipse Editor](image)
Perspectives
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
  (or commands using the menu extension point)
Perspective Factory

> **Declaration**
  - Name
  - Identifier
  - Icon

> **Factory**
  - Views
  - Actions (or commands using the menu extension point)
And there’s more…

- Dialogs and Wizards
- Preference Pages and Property Dialogs
- Extensions and Extension-Points
- Builders, Markers, and Natures
- Logging and Tracing
- Help and Internationalization
- Features and Branding
- Build and Deploy
Questions?

Download risk-free trial versions:
www.instantiations.com/freetrials