Eclipse RCP Part VIII

Automotive Financial Services Insurance Life Science & Healthcare Public Sector Telecommunications & Media Travel & Logistics Utilities Automotive Financial Services Insurance Life Science & Healthcare Public Sector Telecommunications & Media Travel & Logistics Utilities Automotive Financial Services Insurance Life Science & Healthcare Public Sector Telecommunications & Media Travel & Logistics Utilities Automotive Financial Services Insurance Life Science Healthcare Public Sector Telecommunications & Media Travel & Logistics Utilities Automotive Financial Services Life Science & Healthcare Public Sector Telecommunications & Media Travel & Logistics Utilities Automotive Financial Services Insurance Life Science & Healthcare Public Sector Telecommunications & Media Travel & Logistics Utilities Automotive Financial Services Insurance Life Science & Healthcare Telecommunications & Media Travel & Logistics Utilities Automotive Financial Services Insurance Life Science & Healthcare Public Sector Telecommunications & Media Travel & Logistics Utilities Automotive Financial Services Insurance Life Science & Healthcare Public Sector Telecommunications & Media Travel & Logistics Utilities Automotive Financial Services Insurance Life Science & Healthcare Public Sector

.consulting .solutions .partnership

msg systems

Editors

- An editor is a tool that edits a particular application domain specific
- data in a way that is specialized for the data type
- » Plug-ins can contribute editors to the workbench
- » Typical editors can be:
- » text based (eg. Java editor)
- » form based (eg. Plug-in editor)
- » graphic intensive (eg. JBPM process editor)
- » Implementation of an editor is very specific to your application

Contributing editors

- The workbench provides a general infrastructure for building editors
- » To introduce an editor contribute to
- org.eclipse.ui.editors
- <editor id="com.yourcompany.yourapp.myEditor"
  name="%myEditorName"
  icon="icons/obj16/editor.png"
  class="com.yourcompany.yourapp.MyEditorPart">
- </editor>
- Be careful: Contributed editors need an icon, otherwise they will not be available.

Editor implementation

- An Editor must implement IEditorPart
- » extend EditorPart class instead!
- » Similar to views, editors create their UI in the
- createPartControl(Composite) method
- » For setting the editor's input the interface IEditorInput is used
- » **Lightweight** descriptor of an editor input
- » Can be compared to java.io.File
- » The implementation typically contains:
- » A description of an object to be edited (IEditorInput methods)
- » A Reference to an object to be edited

IEditorInput

- You have to implement the following methods:
- » getName() - to return a label for the UI
- » getImageDescriptor() - to return an ImageDescriptor for the UI
- » getToolTipText() - to return a tooltip
- » exists(...) - to find out if the underlying input still exists (used by the "recent editors" list)
- » equals(...) - to find out if an editor with this input is already open
- » An accessor method for the lightweight descriptor (i.e getPerson())

IEditorPart

- You have to implement the following methods:
- » From IWorkbenchPart interface:
- » createPartControl(...) - to create the editor's controls
- » setFocus() - to accept focus
- » From IEditorPart interface:
- » init(...) - to initialize editor when assigned to it's site
- » isDirty() - to decide whether a significant change has occurred
- » doSave() - to save the contents of editor
- » doSaveAs() - to "Save As..." the contents of editor
- » isSaveAsAllowed() - to control if "Save As..." is availiable

Dirty state

- An editor may inform the workbench that some of it's data has been changed
- » This can be done by firing a property change event for property PROP_DIRTY:
- » firePropertyChange(PROP_DIRTY)
- » The workbench will call isDirty() method to query the actual dirty state

Controlling editors

- The workbench page controls its editors:

```
IWorkbenchPage page = PlatformUI.getWorkbench()
        .getActiveWorkbenchWindow().getActivePage()
IEditorInput editorInput = new MyEditorInput();
String id = "com.mycompany.myEditor";

//--- to open an editor
IEditorPart editor = page.openEditor(editorInput, id);

//--- to close the editor
boolean askSaveOnClose = true;
page.closeEditor(editor, askSaveOnClose);
//--- to close all editors
page.closeAllEditors(askSaveOnClose);
```

Lab

- Extend the RCP application from part IX
- Implement PersonEditorInput
  - Remember that the EditorInput should be lightweight -> use an int index for identifying people
  - In PersonEditorInput.getName(), return „Person <index>“
  - Implement PersonEditorInput.equals to only consider the index
- Implement PersonEditor
  - Don‘t forget to properly set the input and site in PersonEditor.init()
  - Use PersonEditor.setPartName() to set the name of the editor instance
    - PersonEditor.setPartName(input.getName());
- Add a DoubleClickListener to your table viewer
  - Open the PersonEditor with the selected person when the listener is invoked

Vielen Dank für Ihre Aufmerksamkeit

.consulting .solutions .partnership

.msg systems