

# **XFinder Experts Study**

## **Content**

1. Ant editor
2. Feature manifest editor
3. HTML editor
4. Java class editor
5. Java source editor
6. JSP editor

## Section 2 – Eclipse Text Editor Implementations

In this section of the study, you will be presented with a short tutorial on how to create a generic Eclipse Text Editor. Then, you will be asked to associate each step in the tutorial with a program element of the editor you are reviewing: **Ant Editor**.

### Eclipse Text Editor Tutorial

To create an Eclipse Text Editor, a developer needs to follow these steps:

1. Create a plug-in project.
2. Make the plug-in project depend on **org.eclipse.ui.workbench.texteditor** plug-in.
3. Make the plug-in project depend on **org.eclipse.jface.text** plug-in.
4. Create an extension **org.eclipse.ui.editors**.
5. Create an element editor in the extension **org.eclipse.ui.editors**.
6. Create a class implementing **ITextEditor**.
7. Create a class extending **SourceViewerConfiguration**.
8. Call method **AbstractTextEditor.setSourceViewerConfiguration**.
9. Create a class extending **RuleBasedScanner**.
10. Create a class extending **RuleBasedPartitionScanner**.
11. Create a class implementing **ITextHover**.
12. Create a class implementing **IContentAssistProcessor**.
13. Create a class implementing **IContentOutlinePage**.

For each of these steps, you will be presented with a list of elements potentially fulfilling the step. Each element is of the right type for the step. For example, if a step asks to implement the interface **IHelloWorld**, only classes that implement the **IHelloWorld** interface (directly or indirectly) are listed.

Please select the element that best fulfills each step for this editor: **Ant Editor**. If more than one element is relevant, simply rank them from the most relevant (1) to the least relevant (10). If an element is not relevant at all, do not put anything beside it. If you are not sure, do not put anything beside it. If you think a relevant element is not in the list, add it beside the “Other” option. If no element is relevant for this step in the context of your editor, choose the “No relevant element” option by putting a 1 beside it.

The **Ant Editor** is represented by this editor element in the ant.ui plugin.xml file:

```
<extension
    point="org.eclipse.ui.editors">
    <editor
        name="%AntEditor.name"
        icon="$n$/icons/full/obj16/ant_buildfile.gif"
        class="org.eclipse.ant.internal.ui.editor.AntEditor"
        contributorClass="org.eclipse.ant.internal.ui.editor.AntEditorActionContributor"
        id="org.eclipse.ant.ui.internal.editor.AntEditor">
        <contentTypeBinding
            contentTypeId="org.eclipse.ant.core.antBuildFile"/>
    </editor>
    . . .
```

### Steps:

1. Create a class implementing **ITextEditor**.

The appropriate implementation(s) of **ITextEditor** for **Ant Editor** is(are):

- JSEditor
- StructuredTextEditor
- DebugTextEditor
- JavaSnippetEditor
- PropertiesFileEditor
- AbstractTextEditor (5)
- StatusTextEditor (4)
- AbstractDecoratedTextEditor (3)
- TextEditor (2)
- CompilationUnitEditor
- JavaEditor
- ClassFileEditor
- AntEditor (1)
- Other:
- No relevant element.

2. Create a class extending **SourceViewerConfiguration**.

The appropriate implementation(s) of **SourceViewerConfiguration** for **Ant Editor** is(are):

- JSEditorSourceViewerConfiguration
- JSSourceViewerConfiguration
- NewHTMLTempaltesWizardPage\$6
- TextSourceViewerConfiguration (3)
- StructuredTextViewerConfigurationHTML
- AntSourceViewerConfiguration (2)
- StructuredTextViewerConfigurationWSDL
- AntEditorSourceViewerCondiguration (1)
- JavaSourceViewerConfiguration

- AntTemplateViewerConfiguration
- Other:
- No relevant element.

3. Call method **AbstractTextEditor.setSourceViewerConfiguration**.

The method AbstractTextEditor.setSourceViewerConfiguration for **Ant Editor** is called in:

- PropertiesFileEditor() - PropertiesFileEditor
- initializeEditor() - JavaEditor
- setPreferenceStore(...) - JavaEditor
- JavaSnippedEditor() - JavaSnippedEditor
- TextEditor() – TextEditor
- createPartControl(Composite) – StructuredTextEditor
- setSourceViewerConfiguration() – JSEditor
- PDEProjectionSourcePage(PDEFormEditor, String, String) – PDEProjectionSourcePage
- AntEditor() – AntEditor (1)
- setSourceViewerConfiguration(SourceViewerConfiguration) – StructuredTextEditor
- updateSourceViewerConfiguration() - StructuredTextEditor
- Other:
- No relevant element.

4. Create a class extending **RuleBasedScanner**.

The appropriate implementation(s) of RuleBasedScanner for **Ant Editor** is(are):

- SingleTokenJavaScanner
- JavaCommentScanner
- JavaCodeScanner
- JavaPartitionScanner
- PropertyValueScanner
- PropertiesFilePartitionScanner
- BufferedRuleBasedScanner
- JavaDocScanner
- JavaCodeScanner
- JSPELCodeScanner
- RuleBasedPartitionScanner
- AbstractJavaScanner
- XMLPartitionScanner
- AbstractAntEditorScanner (1)
- AntEditorTagScanner (1)
- AntEditorProcInstrScanner (1)
- AntEditorPartitionScanner

- Other:
- No relevant element.

5. Create a class extending **RuleBasedPartitionScanner**.

The appropriate implementation(s) of RuleBasedPartitionScanner for **Ant Editor** is(are):

- JavaPartitionScanner
- PropertiesFilePartitionScanner
- AntEditorPartitionScanner (1)
- ManifestPartitionScanner
- XMLPartitionScanner
- Other:
- No relevant element.

6. Create a class implementing **ITextHover**.

The appropriate implementation(s) of ITextHover for **Ant Editor** is(are):

- JavaScriptTagInfoHoverProcessor
- BestMatchHover
- AnnotationHoverProcessor
- ProblemAnnotationHoverProcessor
- XMLTagInfoHoverProcessor
- AbstractHoverProcessor
- AntTemplateVariableTextHover
- ProblemHover
- BestMatchHover
- JavaEditorTextHoverProxy
- JavadocHover
- NLSStringHover
- JavaTypeHover
- JavaSourceHover
- AbstractJavaEditorTextHover
- JavaDebugHover
- AnnotationHover
- XMLTextHover (1)
- ManifestTextHover
- Other:
- No relevant element.

7. Create a class implementing **IContentAssistProcessor**.

The appropriate implementation(s) of IContentAssistProcessor for **Ant Editor**

is(are):

- JavaScriptContentAssistProcessor
- AbstractContentAssistProcessor
- JavaCompletionProcessor
- XMLContentAssistProcessor
- JavaCompletionProcessor
- TemplateCompletionProcessor (2)
- AntEditorCompletionProcessor (1)
- TypeContentProposalProvider
- VariableNamesProcessor
- JavaSourcePackageFragmentRootCompletionProcessor
- JavaDebugContentAssistProcessor
- FieldNameProcessor
- JavaPackageCompletionProcessor
- JavaTypeCompletionProcessor
- JavadocCompletionProcessor
- Other:
- No relevant element.

8. Create a class implementing **IContentOutlinePage**.

The appropriate implementation(s) of IContentOutlinePage for **Ant Editor** is(are):

- JSContentOutlinePage
- ConfigurableContentOutlinePage
- ASDContentOutlinePage
- ADTContentOutlinePage
- OthersContentOutlinePage
- ComponentsContentOutlinePage
- ContentOutlinePage (2)
- ManagedBeanContentOutlinePage
- JavaOutlinePage
- AntEditorContentOutlinePage (1)
- SourceOutlinePage
- Other:
- No relevant element.

This completes the mandatory part of the questionnaire. Thank you very much for taking the time to respond to it. If you have extra time, we would appreciate your completing the next two optional sections about the quality of the tutorial above and your preferences and habits when using and authoring framework documentation.

## Section 2 – Eclipse Text Editor Implementations

In this section of the study, you will be presented with a short tutorial on how to create a generic Eclipse Text Editor. Then, you will be asked to associate each step in the tutorial with a program element of the editor you are reviewing: **Feature Manifest Editor**.

### Eclipse Text Editor Tutorial

To create an Eclipse Text Editor, a developer needs to follow these steps:

1. Create a plug-in project.
2. Make the plug-in project depend on **org.eclipse.ui.workbench.texteditor** plug-in.
3. Make the plug-in project depend on **org.eclipse.jface.text** plug-in.
4. Create an extension **org.eclipse.ui.editors**.
5. Create an element editor in the extension **org.eclipse.ui.editors**.
6. Create a class implementing **ITextEditor**.
7. Create a class extending **SourceViewerConfiguration**.
8. Call method **AbstractTextEditor.setSourceViewerConfiguration**.
9. Create a class extending **RuleBasedScanner**.
10. Create a class extending **RuleBasedPartitionScanner**.
11. Create a class implementing **ITextHover**.
12. Create a class implementing **IContentAssistProcessor**.
13. Create a class implementing **IContentOutlinePage**.

For each of these steps, you will be presented with a list of elements potentially fulfilling the step. Each element is of the right type for the step. For example, if a step asks to implement the interface **IHelloWorld**, only classes that implement the **IHelloWorld** interface (directly or indirectly) are listed.

Please select the element that best fulfills each step for this editor: **Feature Manifest Editor**. If more than one element is relevant, simply rank them from the most relevant (1) to the least relevant (10). If an element is not relevant at all, do not put anything beside it. If you are not sure, do not put anything beside it. If you think a relevant element is not in the list, add it beside the “Other” option. If no element is relevant for this step in the context of your editor, choose the “No relevant element” option by putting a 1 beside it.

The **Feature Manifest Editor** is represented by this editor element in the `pde.ui.plugin.xml` file:

```
<extension
    point="org.eclipse.ui.editors">
...
    <editor
        default="true"
        name="%editors.featureManifest.name"
        icon="$nl$/icons/obj16/ftr_xml_obj.gif"
        class="org.eclipse.pde.internal.ui.editor.feature.FeatureEditor"

contributorClass="org.eclipse.pde.internal.ui.editor.feature.FeatureEditorContributor"

        matchingStrategy="org.eclipse.pde.internal.ui.editor.feature.FeatureEditorMatching
Strategy"
        id="org.eclipse.pde.ui.featureEditor">
        <contentTypeBinding contentTypeId="org.eclipse.pde.featureManifest"/>
    </editor>
...

```

### Steps:

1. Create a class implementing **ITextEditor**.

The appropriate implementation(s) of **ITextEditor** for **Feature Manifest Editor** is(are):

- JSEditor
- FeatureSourcePage - 1
- PDEProjectionSourcePage - 3
- KeyValueSourcePage
- BuildSourcePage
- BundleSourcePage
- XMLSourcePage - 2
- ManifestSourcePage
- SchemaSourcePage
- SiteSourcePage
- AbstractTextEditor
- StatusTextEditor
- AbstractDecoratedTextEditor
- TextEditor
- JavaEditor
- Other:
- No relevant element.

2. Create a class extending **SourceViewerConfiguration**.

The appropriate implementation(s) of **SourceViewerConfiguration** for **Feature Manifest Editor** is(are):

- JSEditorSourceViewerConfiguration
- JSSourceViewerConfiguration

- NewHTMLTempaltesWizardPage\$6
- ChangeAwareSourceViewerConfiguration - 2
- BuildSourceViewerConfiguration
- ManifestConfiguration
- XMLConfiguration - 1
- PluginXMLConfiguration
- StructuredTextViewerConfigurationXML
- JavaSourceViewerConfiguration
- PropertiesFileSourceViewerConfiguration
- TextSourceViewerConfiguration - 3
- Other:
- No relevant element.

3. Call method **AbstractTextEditor.setSourceViewerConfiguration**.

The method AbstractTextEditor.setSourceViewerConfiguration for **Feature Manifest Editor** is called in:

- PropertiesFileEditor() - PropertiesFileEditor
- initializeEditor() - JavaEditor
- setPreferenceStore(...) - JavaEditor
- JavaSnippetEditor() - JavaSnippetEditor
- TextEditor() – TextEditor
- createPartControl(Composite) – StructuredTextEditor
- setSourceViewerConfiguration() – JSEditor
- PDEProjectionSourcePage(PDEFormEditor, String, String) – PDEProjectionSourcePage - 1
- AntEditor() – AntEditor
- setSourceViewerConfiguration(SourceViewerConfiguration) – StructuredTextEditor
- updateSourceViewerConfiguration() - StructuredTextEditor
- Other:
- No relevant element.

4. Create a class extending **RuleBasedScanner**.

The appropriate implementation(s) of RuleBasedScanner for **Feature Manifest Editor** is(are):

- JavaCodeScanner
- PropertyValueScanner
- SingleTokenJavaScanner
- PropertiesFilePartitionScanner
- BufferedRuleBasedScanner
- JavaDocScanner
- BasePDEScanner - 4

- AbstractJavaScanner
- ManifestHeaderScanner
- ManifestValueScanner
- XMLScanner - 1
- XMLTagScanner - 2
- RuleBasedPartitionScanner
- AbstractJavaScanner
- XMLPartitionScanner - 3
- ManifestPartitionScanner
- Other:
- No relevant element.

5. Create a class extending **RuleBasedPartitionScanner**.

The appropriate implementation(s) of RuleBasedPartitionScanner for **Feature Manifest Editor** is(are):

- JavaPartitionScanner
- PropertiesFilePartitionScanner
- AntEditorPartitionScanner
- ManifestPartitionScanner
- XMLPartitionScanner -1
- Other:
- No relevant element.

6. Create a class implementing **ITextHover**.

The appropriate implementation(s) of ITextHover for **Feature Manifest Editor** is(are):

- PDETextHover
- ManifestTextHover
- AnnotationHoverProcessor
- ProblemAnnotationHoverProcessor
- XMLTagInfoHoverProcessor
- AbstractHoverProcessor
- IJavaEditorTextHover
- ProblemHover
- BestMatchHover
- JavaEditorTextHoverProxy
- JavadocHover
- NLSSStringHover
- JavaTypeHover
- JavaSourceHover
- AbstractJavaEditorTextHover
- JavaDebugHover
- AnnotationHover

- XMLTextHover -2
- PluginXMLTextHover
- ManifestTextHover
- Other:
- No relevant element.

7. Create a class implementing **IContentAssistProcessor**.

The appropriate implementation(s) of IContentAssistProcessor for **Feature Manifest Editor** is(are):

- JavaScriptContentAssistProcessor
- AbstractContentAssistProcessor
- JavaCompletionProcessor
- XMLContentAssistProcessor
- JavaCompletionProcessor
- TemplateCompletionProcessor
- TypePackageCompletionProcessor
- TypeContentProposalProvider
- VariableNamesProcessor
- JavaSourcePackageFragmentRootCompletionProcessor
- JavaDebugContentAssistProcessor
- FieldNameProcessor
- JavaPackageCompletionProcessor
- JavaTypeCompletionProcessor
- JavadocCompletionProcessor
- ManifestContentAssistProcessor
- Other:
- No relevant element. - 1

8. Create a class implementing **IContentOutlinePage**.

The appropriate implementation(s) of IContentOutlinePage for **Feature Manifest Editor** is(are):

- ISortableContentOutlinePage
- EmptyOutlinePage
- FormOutlinePage - 2
- ManifestOutlinePage
- SimpleCSFormOutlinePage
- SourceOutlinePage -4
- MultiPageEditorOutlinePage
- PDEMultiEditorPageContentOutline
- PDEOutlinePage -3
- OthersContentOutlinePage
- BuildOutlinePage

- JSContentOutlinePage
- FeatureOutlinePage - 1
- Other:
- No relevant element.

This completes the mandatory part of the questionnaire. Thank you very much for taking the time to respond to it. If you have extra time, we would appreciate your completing the next two optional sections about the quality of the tutorial above and your preferences and habits when using and authoring framework documentation.

## Section 2 – Eclipse Text Editor Implementations

In this section of the study, you will be presented with a short tutorial on how to create a generic Eclipse Text Editor. Then, you will be asked to associate each step in the tutorial with a program element of the editor you are reviewing: **HTML Source Page Editor**.

### Eclipse Text Editor Tutorial

To create an Eclipse Text Editor, a developer needs to follow these steps:

1. Create a plug-in project.
2. Make the plug-in project depend on **org.eclipse.ui.workbench.texteditor** plug-in.
3. Make the plug-in project depend on **org.eclipse.jface.text** plug-in.
4. Create an extension **org.eclipse.ui.editors**.
5. Create an element editor in the extension **org.eclipse.ui.editors**.
6. Create a class implementing **ITextEditor**.
7. Create a class extending **SourceViewerConfiguration**.
8. Call method **AbstractTextEditor.setSourceViewerConfiguration**.
9. Create a class extending **RuleBasedScanner**.
10. Create a class extending **RuleBasedPartitionScanner**.
11. Create a class implementing **ITextHover**.
12. Create a class implementing **IContentAssistProcessor**.
13. Create a class implementing **IContentOutlinePage**.

For each of these steps, you will be presented with a list of elements potentially fulfilling the step. Each element is of the right type for the step. For example, if a step asks to implement the interface **IHelloWorld**, only classes that implement the **IHelloWorld** interface (directly or indirectly) are listed.

Please select the element that best fulfills each step for this editor: **HTML Source Page Editor**. If more than one element is relevant, simply rank them from the most relevant (1) to the least relevant (10). If an element is not relevant at all, do not put anything beside it. If you are not sure, do not put anything beside it. If you think a relevant element is not in the list, add it beside the “Other” option. If no element is relevant for this step in the context of your editor, choose the “No relevant element” option by putting a 1 beside it.

The **HTML Source Page Editor** is represented by this editor element in the wst.html.ui plugin.xml file:

```
<extension
    point="org.eclipse.ui.editors">
    <editor
        name="%HTML_Source_Page_Editor.name"
        icon="$nl$/icons/full/obj16/sourceEditor.gif"

        contributorClass="org.eclipse.wst.html.ui.internal.edit.ui.ActionContributorHTML"
        class="org.eclipse.wst.sse.ui.StructuredTextEditor"
        symbolicFontName="org.eclipse.wst.sse.ui.textfont"
        id="org.eclipse.wst.html.core.htmlsource.source">
        <contentTypeBinding
            contentTypeId="org.eclipse.wst.html.core.htmlsource" />
    </editor>
...

```

### Steps:

1. Create a class implementing **ITextEditor**.

The appropriate implementation(s) of **ITextEditor** for **HTML Source Page Editor** is(are):

- JSEditor
- FeatureSourcePage
- PDEProjectionSourcePage
- KeyValueSourcePage
- BuildSourcePage
- BundleSourcePage
- StructuredTextEditor 1
- DesignerStructuredTextEditorJSP
- DTDEditor
- SiteSourcePage
- AbstractTextEditor
- StatusTextEditor
- AbstractDecoratedTextEditor
- TextEditor
- JavaEditor
- Other:
- No relevant element.

2. Create a class extending **SourceViewerConfiguration**.

The appropriate implementation(s) of **SourceViewerConfiguration** for **HTML Source Page Editor** is(are):

- JSEditorSourceViewerConfiguration
- JSSourceViewerConfiguration
- NewHTMLTempaltesWizardPage\$6
- ChangeAwareSourceViewerConfiguration

- BuildSourceViewerConfiguration
- ManifestConfiguration
- XMLConfiguration
- PluginXMLConfiguration
- StructuredTextViewerConfiguration
- StructuredTextViewerConfigurationJSP
- StructuredTextViewerConfigurationHTML 1
- StructuredTextViewerConfigurationCSS
- JavaSourceViewerConfiguration
- PropertiesFileSourceViewerConfiguration
- TextSourceViewerConfiguration
- Other:
- No relevant element.

3. Call method **AbstractTextEditor.setSourceViewerConfiguration**.

The method AbstractTextEditor.setSourceViewerConfiguration for **HTML Source Page Editor** is called in:

- PropertiesFileEditor() - PropertiesFileEditor
- initializeEditor() - JavaEditor
- setPreferenceStore(...) - JavaEditor
- JavaSnippetEditor() - JavaSnippetEditor
- TextEditor() – TextEditor
- createPartControl(Composite) – StructuredTextEditor 1
- setSourceViewerConfiguration() – JSEditor
- PDEProjectionSourcePage(PDEFormEditor, String, String) – PDEProjectionSourcePage
- AntEditor() – AntEditor
- setSourceViewerConfiguration(SourceViewerConfiguration) – StructuredTextEditor 2
- updateSourceViewerConfiguration() - StructuredTextEditor 3
- Other:
- No relevant element.

4. Create a class extending **RuleBasedScanner**.

The appropriate implementation(s) of RuleBasedScanner for **HTML Source Page Editor** is(are):

- JavaCodeScanner (in jdt)
- PropertyValueScanner
- SingleTokenJavaScanner
- PropertiesFilePartitionScanner
- BufferedRuleBasedScanner
- JavaDocScanner

- BasePDEScanner
- AbstractJavaScanner
- ManifestHeaderScanner
- ManifestValueScanner
- XMLScanner
- XMLTagScanner
- RuleBasedPartitionScanner
- JSPELCodeScanner
- JavaCodeScanner (in jst.jsp.ui.internal.style.java)
- Other:
- No relevant element. 1

5. Create a class extending **RuleBasedPartitionScanner**.

The appropriate implementation(s) of RuleBasedPartitionScanner for **HTML Source Page Editor** is(are):

- JavaPartitionScanner
- PropertiesFilePartitionScanner
- AntEditorPartitionScanner
- ManifestPartitionScanner
- XMLPartitionScanner
- Other:
- No relevant element. 1

6. Create a class implementing **ITextHover**.

The appropriate implementation(s) of ITextHover for **HTML Source Page Editor** is(are):

- AnnotationHoverProcessor
- ProblemAnnotationHoverProcessor
- XMLTagInfoHoverProcessor
- AbstractHoverProcessor
- DebugInfoHoverProcessor
- JavaScriptTagInfoHoverProcessor 2
- JSPJavaJavadocHoverProcessor
- XMLTagInfoHoverProcessor
- HTMLTagInfoHoverProcessor 1
- JSPTagInfoHoverProcessor
- IJavaEditorTextHover
- ProblemHover
- BestMatchHover
- JavaEditorTextHoverProxy
- JavadocHover
- NLSStringHover
- JavaTypeHover

- JavaSourceHover
- AbstractJavaEditorTextHover
- JavaDebugHover
- AnnotationHover
- XMLTextHover
- Other:
- No relevant element.

7. Create a class implementing **IContentAssistProcessor**.

The appropriate implementation(s) of IContentAssistProcessor for **HTML Source Page Editor** is(are):

- JavaScriptContentAssistProcessor 1
- AbstractContentAssistProcessor
- HTMLContentAssistProcessor 1
- JSPContentAssistProcessor
- JSPDummyContentAssistProcessor
- XMLContentAssistProcessor
- JavaCompletionProcessor
- JSPPropertyContentAssistProcessor
- JSPTaglibDirectiveContentAssistProcessor
- JSPUseBeanContentAssistProcessor
- JSPELCompletionProcessor
- VariableNamesProcessor
- JavaSourcePackageFragmentRootCompletionProcessor
- JavaDebugContentAssistProcessor
- FieldNameProcessor
- JSPCompletionProcessor
- JavaTypeCompletionProcessor
- JavadocCompletionProcessor
- JSFContentAssistProcessor
- JSPJavaContentAssistProcessor
- JSPELJavaContentAssistProcessor
- Other: CSSContentAssistProcessor 1,  
NoRegionContentAssistProcessorForHTML 1
- No relevant element.

8. Create a class implementing **IContentOutlinePage**.

The appropriate implementation(s) of IContentOutlinePage for **HTML Source Page Editor** is(are):

- ISortableContentOutlinePage
- EmptyOutlinePage
- FormOutlinePage

- ManifestOutlinePage
- SimpleCSFormOutlinePage
- SourceOutlinePage
- MultiPageEditorOutlinePage
- OthersContentOutlinePage
- PageflowEditorOutlinePage
- JSContentOutlinePage
- FeatureOutlinePage
- Other:
- No relevant element. 1

This completes the mandatory part of the questionnaire. Thank you very much for taking the time to respond to it. If you have extra time, we would appreciate your completing the next two optional sections about the quality of the tutorial above and your preferences and habits when using and authoring framework documentation.

## Section 2 – Eclipse Text Editor Implementations

In this section of the study, you will be presented with a short tutorial on how to create a generic Eclipse Text Editor. Then, you will be asked to associate each step in the tutorial with a program element of the editor you are reviewing: **Java Class Editor**.

### Eclipse Text Editor Tutorial

To create an Eclipse Text Editor, a developer needs to follow these steps:

14. Create a plug-in project.
15. Make the plug-in project depend on **org.eclipse.ui.workbench.texteditor** plug-in.
16. Make the plug-in project depend on **org.eclipse.jface.text** plug-in.
17. Create an extension **org.eclipse.ui.editors**.
18. Create an element editor in the extension **org.eclipse.ui.editors**.
19. Create a class implementing **ITextEditor**.
20. Create a class extending **SourceViewerConfiguration**.
21. Call method **AbstractTextEditor.setSourceViewerConfiguration**.
22. Create a class extending **RuleBasedScanner**.
23. Create a class extending **RuleBasedPartitionScanner**.
24. Create a class implementing **ITextHover**.
25. Create a class implementing **IContentAssistProcessor**.
26. Create a class implementing **IContentOutlinePage**.

For each of these steps, you will be presented with a list of elements potentially fulfilling the step. Each element is of the right type for the step. For example, if a step asks to implement the interface **IHelloWorld**, only classes that implement the **IHelloWorld** interface (directly or indirectly) are listed.

Please select the element that best fulfills each step for this editor: **Java Class Editor**. If more than one element is relevant, simply rank them from the most relevant (1) to the least relevant (10). If an element is not relevant at all, do not put anything beside it. If you are not sure, do not put anything beside it. If you think a relevant element is not in the list, add it beside the “Other” option. If no element is relevant for this step in the context of your editor, choose the “No relevant element” option by putting a 1 beside it.

The **Java Class Editor** is represented by this editor element in the jdt.ui plugin.xml file:

```
<extension
  id="javaeditor"
  point="org.eclipse.ui.editors">
...
  <editor
    name="%ClassFileViewerName"
    default="true"
    icon="$nl$/icons/full/obj16/classf_obj.gif"

contributorClass="org.eclipse.jdt.internal.ui.javaeditor.ClassFileEditorActionContributor
"
    class="org.eclipse.jdt.internal.ui.javaeditor.ClassFileEditor"
    symbolicFontName="org.eclipse.jdt.ui.editors.textfont"
    id="org.eclipse.jdt.ui.ClassFileEditor">
    <contentTypeBinding
      contentTypeId="org.eclipse.jdt.core.javaClass"
    />
  </editor>
...

```

### Steps:

1. Create a class implementing **ITextEditor**.

The appropriate implementation(s) of ITextEditor for **Java Class Editor** is(are):

- JSEditor
- StructuredTextEditor
- DebugTextEditor
- JavaSnippetEditor
- PropertiesFileEditor
- AbstractTextEditor
- StatusTextEditor
- AbstractDecoratedTextEditor
- TextEditor
- CompilationUnitEditor
- JavaEditor
- ClassFileEditor (1)
- DesignerStructuredTextEditorJSP
- Other:
- No relevant element.

2. Create a class extending **SourceViewerConfiguration**.

The appropriate implementation(s) of SourceViewerConfiguration for **Java Class Editor** is(are):

- JSEditorSourceViewerConfiguration
- JSSourceViewerConfiguration
- NewHTMLTempaltesWizardPage\$6
- StructuredTextViewerConfiguration
- JavaSnippetViewerConfiguration

- SimpleJavaSourceViewerConfiguration
- CodeTemplateSourceViewerConfiguration
- JavaSourceViewerConfiguration (1)
- PropertiesFileSourceViewerConfiguration
- TextSourceViewerConfiguration
- Other:
- No relevant element.

3. Call method **AbstractTextEditor.setSourceViewerConfiguration**.

The method AbstractTextEditor.setSourceViewerConfiguration for **Java Class Editor** is called in:

- PropertiesFileEditor() - PropertiesFileEditor
- initializeEditor() – JavaEditor (1)
- setPreferenceStore(...) - JavaEditor
- JavaSnippetEditor() - JavaSnippetEditor
- TextEditor() – TextEditor (2)
- createPartControl(Composite) – StructuredTextEditor
- setSourceViewerConfiguration() – JSEditor
- PDEProjectionSourcePage(PDEFormEditor, String, String) – PDEProjectionSourcePage
- AntEditor() – AntEditor
- setSourceViewerConfiguration(SourceViewerConfiguration) – StructuredTextEditor
- updateSourceViewerConfiguration() - StructuredTextEditor
- Other:
- No relevant element.

4. Create a class extending **RuleBasedScanner**.

The appropriate implementation(s) of RuleBasedScanner for **Java Class Editor** is(are):

- SingleTokenJavaScanner (1)
- JavaCommentScanner (1)
- JavaCodeScanner (1)
- JavaPartitionScanner
- PropertyValueScanner
- PropertiesFilePartitionScanner
- BufferedRuleBasedScanner
- JavaDocScanner (1)
- JSPELCodeScanner
- RuleBasedPartitionScanner
- AbstractJavaScanner
- XMLPartitionScanner

- AntEditorTagScanner
- Other:
- No relevant element.

5. Create a class extending **RuleBasedPartitionScanner**.

The appropriate implementation(s) of RuleBasedPartitionScanner for **Java Class Editor** is(are):

- JavaPartitionScanner (1)
- PropertiesFilePartitionScanner
- AntEditorPartitionScanner
- ManifestPartitionScanner
- XMLPartitionScanner
- Other:
- No relevant element.

6. Create a class implementing **ITextHover**.

The appropriate implementation(s) of ITextHover for **Java Class Editor** is(are):

- JavaScriptTagInfoHoverProcessor
- BestMatchHover (1)
- AnnotationHoverProcessor
- ProblemAnnotationHoverProcessor
- XMLTagInfoHoverProcessor
- AbstractHoverProcessor
- IJavaEditorTextHover
- ProblemHover (1)
- BestMatchHover (1)
- JavaEditorTextHoverProxy
- JavadocHover (1)
- NLSStringHover (1)
- JavaTypeHover
- JavaSourceHover (1)
- AbstractJavaEditorTextHover
- JavaDebugHover
- AnnotationHover
- XMLTextHover
- ManifestTextHover
- Other:

- No relevant element.

7. Create a class implementing **IContentAssistProcessor**.

The appropriate implementation(s) of IContentAssistProcessor for **Java Class Editor** is(are):

- JavaScriptContentAssistProcessor
- AbstractContentAssistProcessor
- JavaCompletionProcessor (1)
- XMLContentAssistProcessor
- JavaCompletionProcessor (1)
- TemplateCompletionProcessor
- TypePackageCompletionProcessor
- TypeContentProposalProvider
- VariableNamesProcessor
- JavaSourcePackageFragmentRootCompletionProcessor
- JavaDebugContentAssistProcessor
- FieldNameProcessor
- JavaPackageCompletionProcessor
- JavaTypeCompletionProcessor
- JavadocCompletionProcessor
- Other:
- No relevant element.

8. Create a class implementing **IContentOutlinePage**.

The appropriate implementation(s) of IContentOutlinePage for **Java Class Editor** is(are):

- JSContentOutlinePage
- ConfigurableContentOutlinePage
- ASDContentOutlinePage
- ADTContentOutlinePage
- OthersContentOutlinePage
- ComponentsContentOutlinePage
- ContentOutlinePage
- PageflowEditorOutlinePage
- ManagedBeanContentOutlinePage
- JavaOutlinePage (1)
- MultiPageEditorOutlinePage
- SourceOutlinePage
- Other:
- No relevant element.

This completes the mandatory part of the questionnaire. Thank you very much for taking

the time to respond to it. If you have extra time, we would appreciate your completing the next two optional sections about the quality of the tutorial above and your preferences and habits when using and authoring framework documentation.

## Section 2 – Eclipse Text Editor Implementations

In this section of the study, you will be presented with a short tutorial on how to create a generic Eclipse Text Editor. Then, you will be asked to associate each step in the tutorial with a program element of the editor you are reviewing: **Java Source Editor**.

### Eclipse Text Editor Tutorial

To create an Eclipse Text Editor, a developer needs to follow these steps:

27. Create a plug-in project.
28. Make the plug-in project depend on **org.eclipse.ui.workbench.texteditor** plug-in.
29. Make the plug-in project depend on **org.eclipse.jface.text** plug-in.
30. Create an extension **org.eclipse.ui.editors**.
31. Create an element editor in the extension **org.eclipse.ui.editors**.
32. Create a class implementing **ITextEditor**.
33. Create a class extending **SourceViewerConfiguration**.
34. Call method **AbstractTextEditor.setSourceViewerConfiguration**.
35. Create a class extending **RuleBasedScanner**.
36. Create a class extending **RuleBasedPartitionScanner**.
37. Create a class implementing **ITextHover**.
38. Create a class implementing **IContentAssistProcessor**.
39. Create a class implementing **IContentOutlinePage**.

For each of these steps, you will be presented with a list of elements potentially fulfilling the step. Each element is of the right type for the step. For example, if a step asks to implement the interface **IHelloWorld**, only classes that implement the **IHelloWorld** interface (directly or indirectly) are listed.

Please select the element that best fulfills each step for this editor: **Java Source Editor**. If more than one element is relevant, simply rank them from the most relevant (1) to the least relevant (10). If an element is not relevant at all, do not put anything beside it. If you are not sure, do not put anything beside it. If you think a relevant element is not in the list, add it beside the “Other” option. If no element is relevant for this step in the context of your editor, choose the “No relevant element” option by putting a 1 beside it.

The **Java Source Editor** is represented by this editor element in the `jdk.ui.plugin.xml` file:

```
<extension
  id="javaeditor"
  point="org.eclipse.ui.editors">
  <editor
    name="%CompilationUnitEditorName"
    default="true"
    icon="$nl$/icons/full/obj16/jcu_obj.gif"
    contributorClass=
"org.eclipse.jdt.internal.ui.javaeditor.CompilationUnitEditorActionContributor"
    class="org.eclipse.jdt.internal.ui.javaeditor.CompilationUnitEditor"
    symbolicFontName="org.eclipse.jdt.ui.editors.textfont"
    id="org.eclipse.jdt.ui.CompilationUnitEditor">
    <contentTypeBinding
      contentTypeId="org.eclipse.jdt.core.javaSource"
    />
  </editor>
  ...
```

**NOTE: I only considered non-abstract stuff as being a relevant implementation.**

### Steps:

1. Create a class implementing **ITextEditor**.

The appropriate implementation(s) of **ITextEditor** for **Java Source Editor** is(are):

- JSEditor
- StructuredTextEditor
- DebugTextEditor
- JavaSnippedEditor
- PropertiesFileEditor
- AbstractTextEditor
- StatusTextEditor
- AbstractDecoratedTextEditor
- TextEditor
- CompilationUnitEditor**
- JavaEditor
- ClassFileEditor
- DesignerStructuredTextEditorJSP
- Other:
- No relevant element.

2. Create a class extending **SourceViewerConfiguration**.

The appropriate implementation(s) of **SourceViewerConfiguration** for **Java Source Editor** is(are):

- JSEditorSourceViewerConfiguration
- JSSourceViewerConfiguration

- NewHTMLTempaltesWizardPage\$6
- StructuredTextViewerConfiguration
- StructuredTextViewerConfigurationHTML
- StructuredTextViewerConfigurationWSDL
- StructuredTextViewerConfigurationWSDL
- StructuredTextViewerConfigurationXML
- JavaSourceViewerConfiguration**
- PropertiesFileSourceViewerConfiguration
- Other:
- No relevant element

3. Call method **AbstractTextEditor.setSourceViewerConfiguration**.

The method AbstractTextEditor.setSourceViewerConfiguration for **Java Source Editor** is called in:

- PropertiesFileEditor() - PropertiesFileEditor
- initializeEditor() - JavaEditor ← **used to be called but is no longer as of now**
- setPreferenceStore(...) - JavaEditor**
- JavaSnippedEditor() - JavaSnippedEditor
- TextEditor() – TextEditor
- createPartControl(Composite) – StructuredTextEditor
- setSourceViewerConfiguration() – JSEditor
- PDEProjectionSourcePage(PDEFormEditor, String, String) – PDEProjectionSourcePage
- AntEditor() – AntEditor
- setSourceViewerConfiguration(SourceViewerConfiguration) – StructuredTextEditor
- updateSourceViewerConfiguration() - StructuredTextEditor
- Other:
- No relevant element.

4. Create a class extending **RuleBasedScanner**.

The appropriate implementation(s) of RuleBasedScanner for **Java Source Editor** is(are):

- 1 SingleTokenJavaScanner**
- 1 JavaCommentScanner**
- 1 JavaCodeScanner**
- JavaPartitionScanner
- PropertyValueScanner
- PropertiesFilePartitionScanner
- BufferedRuleBasedScanner
- 1 JavaDocScanner**

- JSPELCodeScanner
- RuleBasedPartitionScanner
- AbstractJavaScanner
- XMLPartitionScanner
- AntEditorTagScanner
- Other:
- No relevant element.

5. Create a class extending **RuleBasedPartitionScanner**.

The appropriate implementation(s) of RuleBasedPartitionScanner for **Java Source Editor** is(are):

- JavaPartitionScanner**
- PropertiesFilePartitionScanner
- AntEditorPartitionScanner
- ManifestPartitionScanner
- XMLPartitionScanner
- Other:
- No relevant element.

6. Create a class implementing **ITextHover**.

The appropriate implementation(s) of ITextHover for **Java Source Editor** is(are):

- JavaScriptTagInfoHoverProcessor
- BestMatchHover
- AnnotationHoverProcessor
- ProblemAnnotationHoverProcessor
- XMLTagInfoHoverProcessor
- AbstractHoverProcessor
- IJavaEditorTextHover
- 1 ProblemHover**
- 1 BestMatchHover**
- 1 JavaEditorTextHoverProxy**
- 1 JavadocHover**
- 1 NLSStringHover**
- 1 JavaTypeHover**
- 1 JavaSourceHover**
- AbstractJavaEditorTextHover
- 2 JavaDebugHover** ← **contributed by Debug plug-in**
- 1 AnnotationHover**
- XMLTextHover
- ManifestTextHover
- Other:
- No relevant element.

7. Create a class implementing **IContentAssistProcessor**.

The appropriate implementation(s) of IContentAssistProcessor for **Java Source Editor** is(are):

- JavaScriptContentAssistProcessor
- AbstractContentAssistProcessor
- 1 JavaCompletionProcessor**
- XMLContentAssistProcessor
- 2 TemplateCompletionProcessor**
- TypePackageCompletionProcessor
- TypeContentProposalProvider
- VariableNamesProcessor
- JavaSourcePackageFragmentRootCompletionProcessor
- JavaDebugContentAssistProcessor
- FieldNameProcessor
- JavaPackageCompletionProcessor
- JavaTypeCompletionProcessor
- 1 JavadocCompletionProcessor**
- Other: **2 HippeCompletionProcessor**
- No relevant element.

8. Create a class implementing **IContentOutlinePage**.

The appropriate implementation(s) of IContentOutlinePage for **Java Source Editor** is(are):

- JSContentOutlinePage
- ConfigurableContentOutlinePage
- ASDContentOutlinePage
- ADTContentOutlinePage
- OthersContentOutlinePage
- ComponentsContentOutlinePage
- ContentOutlinePage
- PageflowEditorOutlinePage
- ManagedBeanContentOutlinePage
- JavaOutlinePage**
- MultiPageEditorOutlinePage
- SourceOutlinePage
- Other:
- No relevant element.

This completes the mandatory part of the questionnaire. Thank you very much for taking the time to respond to it. If you have extra time, we would appreciate your completing the next two optional sections about the quality of the tutorial above and your preferences

and habits when using and authoring framework documentation.

## Section 2 – Eclipse Text Editor Implementations

In this section of the study, you will be presented with a short tutorial on how to create a generic Eclipse Text Editor. Then, you will be asked to associate each step in the tutorial with a program element of the editor you are reviewing: **JSP Source Page Editor**.

### Eclipse Text Editor Tutorial

To create an Eclipse Text Editor, a developer needs to follow these steps:

40. Create a plug-in project.
41. Make the plug-in project depend on **org.eclipse.ui.workbench.texteditor** plug-in.
42. Make the plug-in project depend on **org.eclipse.jface.text** plug-in.
43. Create an extension **org.eclipse.ui.editors**.
44. Create an element editor in the extension **org.eclipse.ui.editors**.
45. Create a class implementing **ITextEditor**.
46. Create a class extending **SourceViewerConfiguration**.
47. Call method **AbstractTextEditor.setSourceViewerConfiguration**.
48. Create a class extending **RuleBasedScanner**.
49. Create a class extending **RuleBasedPartitionScanner**.
50. Create a class implementing **ITextHover**.
51. Create a class implementing **IContentAssistProcessor**.
52. Create a class implementing **IContentOutlinePage**.

For each of these steps, you will be presented with a list of elements potentially fulfilling the step. Each element is of the right type for the step. For example, if a step asks to implement the interface **IHelloWorld**, only classes that implement the **IHelloWorld** interface (directly or indirectly) are listed.

Please select the element that best fulfills each step for this editor: **JSP Source Page Editor**. If more than one element is relevant, simply rank them from the most relevant (1) to the least relevant (10). If an element is not relevant at all, do not put anything beside it. If you are not sure, do not put anything beside it. If you think a relevant element is not in the list, add it beside the “Other” option. If no element is relevant for this step in the context of your editor, choose the “No relevant element” option by putting a 1 beside it.

The **JSP Source Page Editor** is represented by this editor element in the `jst.jsp.ui.plugin.xml` file:

```
<extension
  point="org.eclipse.ui.editors">
  <editor
    name="%JSP_Source_Page_Editor.name"
    icon="$nl$/icons//full/obj16/sourceEditor.gif"
    extensions="jsp, jsf, jspf, jsp, tag, tagx, tagf"

    contributorClass="org.eclipse.jst.jsp.ui.internal.editor.ActionContributorJSP"
    class="org.eclipse.wst.sse.ui.StructuredTextEditor"
    symbolicFontName="org.eclipse.wst.sse.ui.textfont"
    id="org.eclipse.jst.jsp.core.jspsource.source">
    <contentTypeBinding
      contentTypeId="org.eclipse.jst.jsp.core.jspsource" />
  </editor>
  ...
```

### Steps:

1. Create a class implementing **ITextEditor**.

The appropriate implementation(s) of **ITextEditor** for **JSP Source Page Editor** is(are):

- JSEditor
- FeatureSourcePage
- PDEProjectionSourcePage
- KeyValueSourcePage
- BuildSourcePage
- BundleSourcePage
- StructuredTextEditor
- DesignerStructuredTextEditorJSP
- DTDEditor
- SiteSourcePage
- AbstractTextEditor
- StatusTextEditor
- AbstractDecoratedTextEditor
- TextEditor
- JavaEditor
- Other:
- No relevant element.

2. Create a class extending **SourceViewerConfiguration**.

The appropriate implementation(s) of **SourceViewerConfiguration** for **JSP Source Page Editor** is(are):

- JSEditorSourceViewerConfiguration
- JSSourceViewerConfiguration
- NewHTMLTempaltesWizardPage\$6
- ChangeAwareSourceViewerConfiguration

- BuildSourceViewerConfiguration
- ManifestConfiguration
- XMLConfiguration
- PluginXMLConfiguration
- StructuredTextViewerConfiguration
- StructuredTextViewerConfigurationJSP
- StructuredTextViewerConfigurationHTML
- StructuredTextViewerConfigurationCSS
- JavaSourceViewerConfiguration
- PropertiesFileSourceViewerConfiguration
- TextSourceViewerConfiguration
- Other:
- No relevant element.

3. Call method **AbstractTextEditor.setSourceViewerConfiguration**.

The method AbstractTextEditor.setSourceViewerConfiguration for **JSP Source Page Editor** is called in:

- PropertiesFileEditor() - PropertiesFileEditor
- initializeEditor() - JavaEditor
- setPreferenceStore(...) - JavaEditor
- JavaSnippetEditor() - JavaSnippetEditor
- TextEditor() – TextEditor
- createPartControl(Composite) – StructuredTextEditor
- setSourceViewerConfiguration() – JSEditor
- PDEProjectionSourcePage(PDEFormEditor, String, String) – PDEProjectionSourcePage
- AntEditor() – AntEditor
- setSourceViewerConfiguration(SourceViewerConfiguration) – StructuredTextEditor
- updateSourceViewerConfiguration() - StructuredTextEditor
- Other:
- No relevant element.

4. Create a class extending **RuleBasedScanner**.

The appropriate implementation(s) of RuleBasedScanner for **JSP Source Page Editor** is(are):

- JavaCodeScanner (in jdt)
- PropertyValueScanner
- SingleTokenJavaScanner
- PropertiesFilePartitionScanner
- BufferedRuleBasedScanner
- JavaDocScanner

- BasePDEScanner
- AbstractJavaScanner
- ManifestHeaderScanner
- ManifestValueScanner
- XMLScanner
- XMLTagScanner
- RuleBasedPartitionScanner
- JSPELCodeScanner
- JavaCodeScanner (in jst.jsp.ui.internal.style.java)
- Other:
- No relevant element.

5. Create a class extending **RuleBasedPartitionScanner**.

The appropriate implementation(s) of RuleBasedPartitionScanner for **JSP Source Page Editor** is(are):

- JavaPartitionScanner
- PropertiesFilePartitionScanner
- AntEditorPartitionScanner
- ManifestPartitionScanner
- XMLPartitionScanner
- Other:
- No relevant element.

6. Create a class implementing **ITextHover**.

The appropriate implementation(s) of ITextHover for **JSP Source Page Editor** is(are):

- AnnotationHoverProcessor
- ProblemAnnotationHoverProcessor
- XMLTagInfoHoverProcessor
- AbstractHoverProcessor
- DebugInfoHoverProcessor
- JavaScriptTagInfoHoverProcessor
- JSPJavaJavadocHoverProcessor
- XMLTagInfoHoverProcessor
- HTMLTagInfoHoverProcessor
- JSPTagInfoHoverProcessor
- IJavaEditorTextHover
- ProblemHover
- BestMatchHover
- JavaEditorTextHoverProxy
- JavadocHover
- NLSStringHover
- JavaTypeHover

- JavaSourceHover
- AbstractJavaEditorTextHover
- JavaDebugHover
- AnnotationHover
- XMLTextHover
- Other:
- No relevant element.

7. Create a class implementing **IContentAssistProcessor**.

The appropriate implementation(s) of IContentAssistProcessor for **JSP Source Page Editor** is(are):

- JavaScriptContentAssistProcessor
- AbstractContentAssistProcessor
- HTMLContentAssistProcessor
- JSPContentAssistProcessor
- JSPDummyContentAssistProcessor
- XMLContentAssistProcessor
- JavaCompletionProcessor
- JSPPropertyContentAssistProcessor
- JSPTaglibDirectiveContentAssistProcessor
- JSPUseBeanContentAssistProcessor
- JSPELCompletionProcessor
- VariableNamesProcessor
- JavaSourcePackageFragmentRootCompletionProcessor
- JavaDebugContentAssistProcessor
- FieldNameProcessor
- JSPCompletionProcessor
- JavaTypeCompletionProcessor
- JavadocCompletionProcessor
- JSFContentAssistProcessor
- JSPJavaContentAssistProcessor
- JSPELJavaContentAssistProcessor
- Other:
- No relevant element.

8. Create a class implementing **IContentOutlinePage**.

The appropriate implementation(s) of IContentOutlinePage for **JSP Source Page**

**Editor** is(are):

- ISortableContentOutlinePage
- EmptyOutlinePage
- FormOutlinePage
- ManifestOutlinePage
- SimpleCSFormOutlinePage
- SourceOutlinePage
- MultiPageEditorOutlinePage
- JSContentOutlinePage
- OthersContentOutlinePage
- PageflowEditorOutlinePage
- JSContentOutlinePage
- FeatureOutlinePage
- Other: ConfigurableContentOutlinePage
- No relevant element.

This completes the mandatory part of the questionnaire. Thank you very much for taking the time to respond to it. If you have extra time, we would appreciate your completing the next two optional sections about the quality of the tutorial above and your preferences and habits when using and authoring framework documentation.