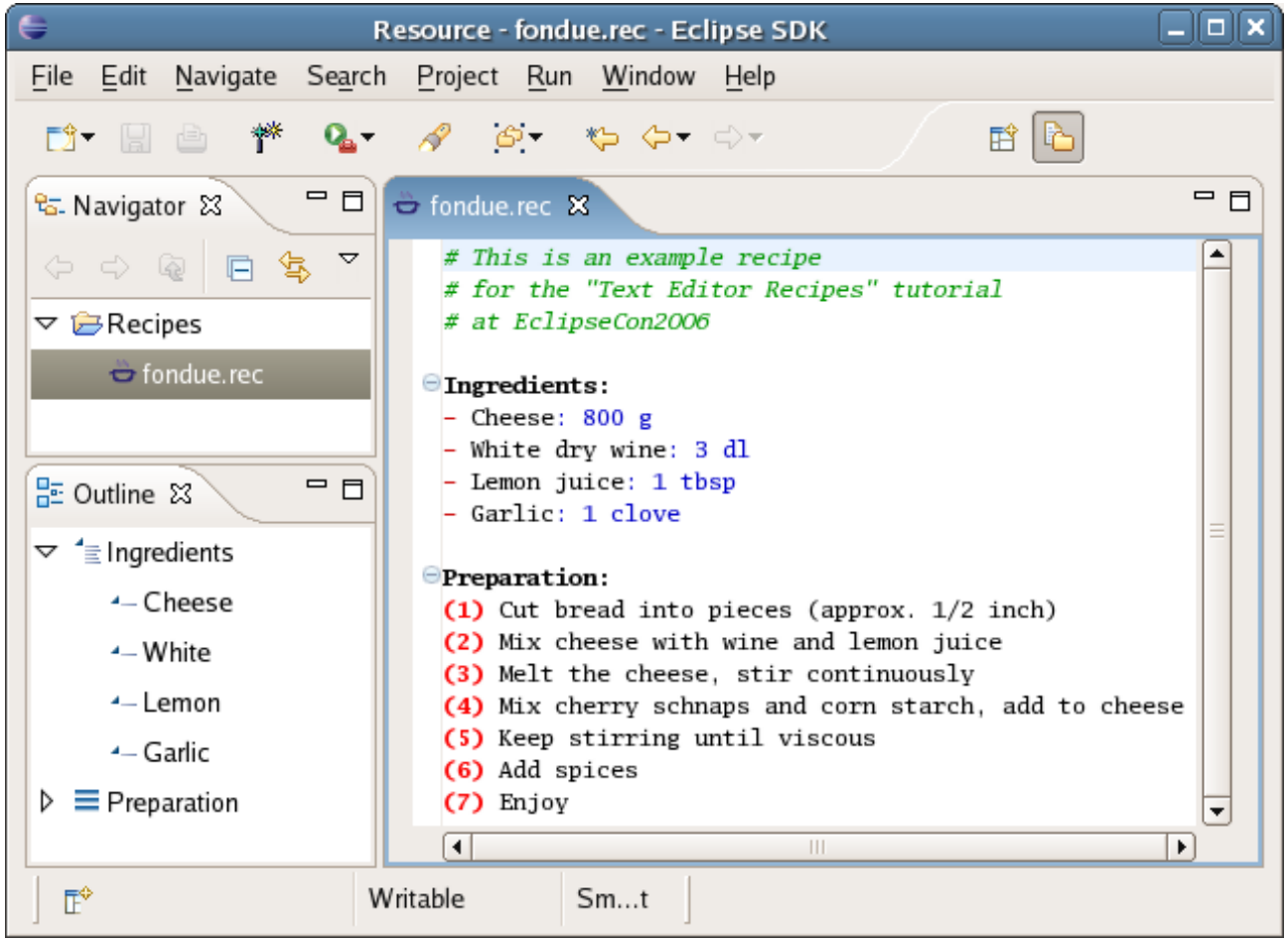
Text Editor Recipes

Season's recipes for your text editor

Tom Eicher
IBM Eclipse Team

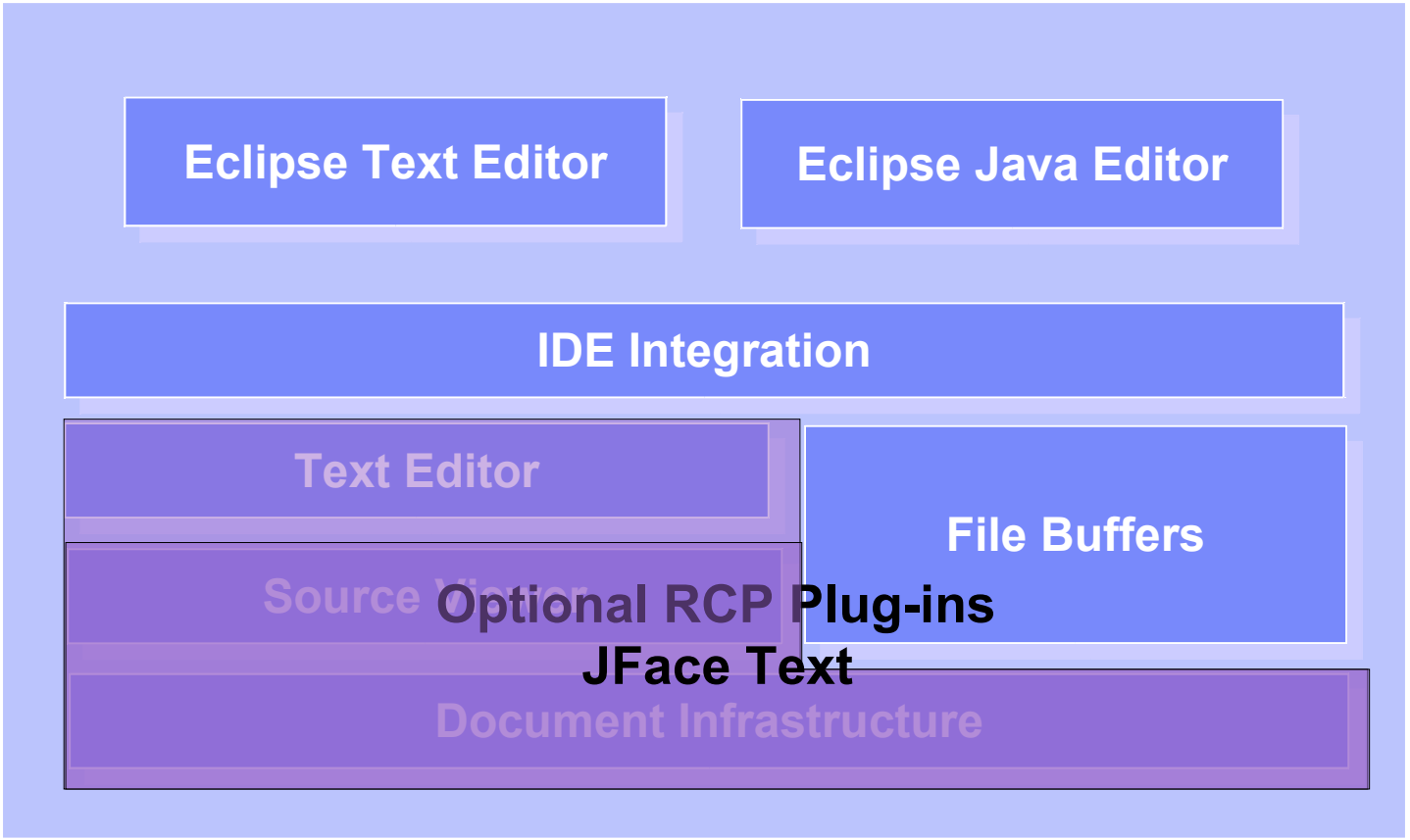
Goal: Rich Source Code Editor



Outline

- Architecture overview
  - Where does RCP come in?
- Create a Text Editor
  - No Java™ code needed initially
  - Check out all the stuff we got for free!
  - Add features:
    - Syntax highlighting
    - Partitioning
- Q & A

Overview of the Architectural Layers (1/4)



Overview of the Architectural Layers (2/4)

- Document Infrastructure
  - Text manipulation through text edits
  - Positions
  - Find/replace
  - Line tracking
- Source Viewer Framework
  - **`TextViewer`**, **`SourceViewer`** and **`SourceViewerConfiguration`**
  - Concept of annotations, hovers
  - Content assist
  - Syntax highlighting
  - Reconciler

Overview of the Architectural Layers (3/4)

- Text Editor Framework
  - Leverages source viewer framework inside editor part and workbench
  - **`AbstractTextEditor`**
  - No concrete editor is provided
- IDE Integration
  - **`AbstractDecoratedTextEditor`**
  - Uses file buffers and resource model
  - Introduces shared text editor preferences
- Default Text Editor
  - Default text editor for Eclipse SDK
  - Simple editor for editing and annotating text documents

Overview of the Architectural Layers (4/4)

- File Buffers
  - A file buffer represents a file that is being modified over time
  - Clients can share a file buffer
  - File buffers provide access to
    - a content model: **`IDocument`**
    - a marker model: **`IAnnotationModel`**
  - Provided for workspace files (**`IFile`**) and external files (**`java.io.File`**)

Contribute a Text Editor

- Contribute an editor by re-using existing implementations
  - Needs some searching around
  - Get the dependencies right
  - Existing editor class: **`TextEditor`**
  - Need to provide an icon
- Connect the menu and tool bar
  - Existing implementation: **`TextEditorActionContributor`**

Create Our Own Editor Class

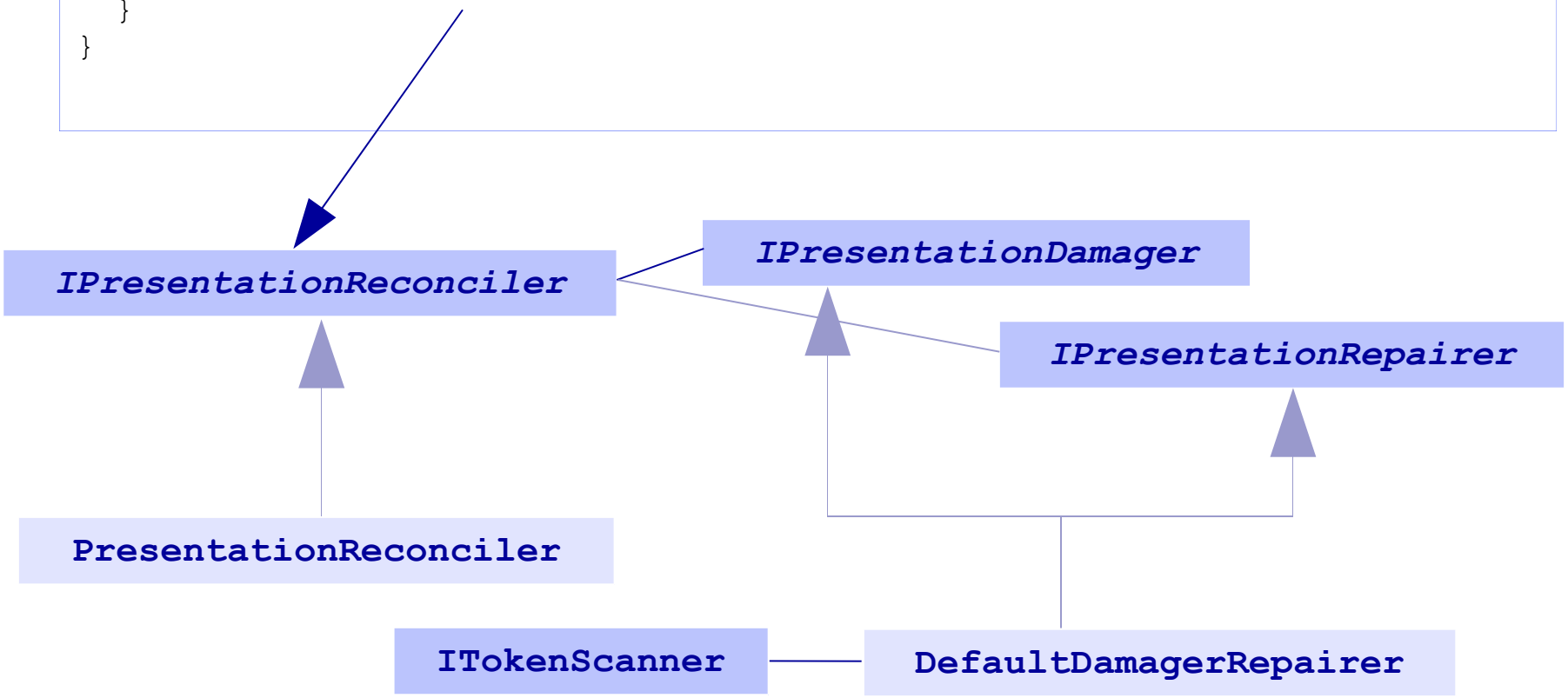
- Text editor hierarchy
  - **`TextEditor`**
    - the default Eclipse text editor
  - **`AbstractDecoratedTextEditor`**
    - Rulers, line numbers, quick diff, …
    - Support for General > Editors > Text Editors preferences
  - **`AbstractTextEditor`**
    - Find/Replace, URL hyperlink navigation, …
    - RCP

Source Viewer Configuration

- Bundles the configuration space of a source viewer
  - Presentation reconciler (syntax coloring)
  - Content assist
  - Hovers
  - Formatter
  - ...
- Many features can be provided separately for each ***partition type***

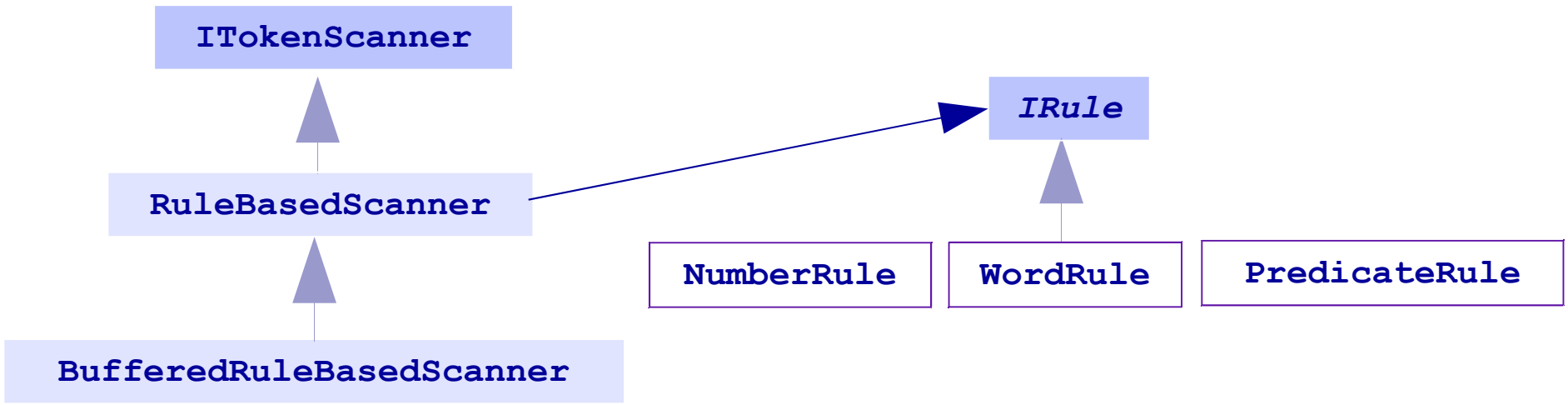
Adding Syntax Coloring: Viewer Configuration

```
public class ExampleSourceViewerConfiguration {
…
   IPresentationReconciler getPresentationReconciler(ISourceViewer viewer) {
      return IPresentationReconciler;
   }
}
```



Adding Syntax Coloring: Presentation Reconciler

```
public class ExampleSourceViewerConfiguration {
   …
   IPresentationReconciler getPresentationReconciler(ISourceViewer viewer) {
     PresentationReconciler reconciler= new PresentationReconciler();
     DefaultDamagerRepairer dflt= new DefaultDamagerRepairer(createScanner());
     reconciler.setDamager(dflt, IDocument.DEFAULT_CONTENT_TYPE);
     reconciler.setRepairer(dflt, IDocument.DEFAULT_CONTENT_TYPE);
     return reconciler;
   }
}
```



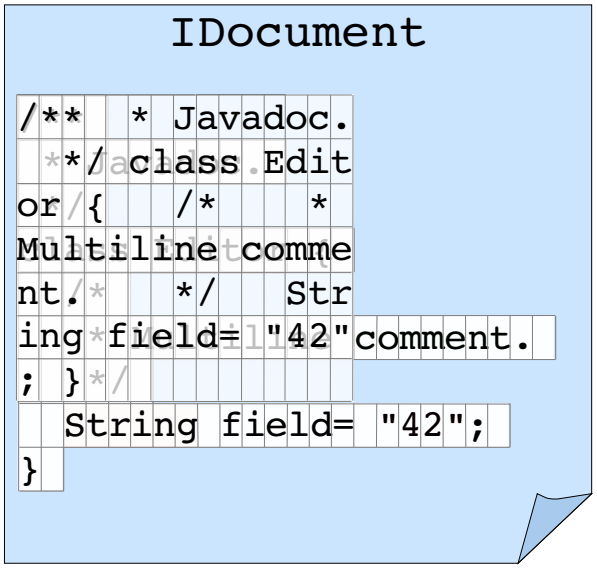
Adding Syntax Coloring: Token Scanner

```
public class ExampleSourceViewerConfiguration {
  …
  private ITokenScanner createScanner() {
    RuleBasedScanner scanner= new RuleBasedScanner();
    scanner.setRules(createRules());
    return scanner;
  }

  private IRule[] createRules() {
    IToken tokenA= new Token(new TextAttribute(getBlueColor());
    IToken tokenB= new Token(new TextAttribute(getGrayColor());

    return new IRule[] {
      new PatternRule(">", "<", tokenA, '\\', false),
      new EndOfLineRule("-- ", tokenB)
    };
  }
}
```

The `IDocument` Text Model



- Sequence of characters
  - Supports random access and replace
  - Event notifications via **`IDocumentListener`**
- Sequence of lines
  - Query by offset and line number
- Positions
  - Ranges that are adjusted to modifications
  - **`IPositionUpdater`** strategies handles overlapping changes
- Partitions
  - Slice the document into segments of the same **content type**
  - Domain model dependent

Document Partitioning

```
/**
 * Returns the name.
 */
String getName() {
    /*
     * return fName
     */
    return fName; // null?
}
```

- Partitioning is a semantic view onto the document
  - each partition has a content type
  - each character of a document belongs to a partition
  - documents support multiple partitionings
  - partitioning is always up-to-date

- Often source viewer configuration is based on content types
  - syntax coloring
  - model reconcilers
  - hovers
  - double click strategy

Document Partitioning

- Document provider should ensure that needed partitionings are installed
- Document setup can also be managed by the file buffer manager
  - ➔ participate in the document setup process of the file buffer manager
- File buffer document setup should only be used if the partitioning is considered of interest for non-UI clients. It should not contribute the default partitioning

Where Do Documents Come From? (1/2)

```
interface ITextEditor extends IEditorPart {
…
     IDocumentProvider getDocumentProvider();
}
```

- Each editor is connected to a document provider which is normally shared between editors
- Document Provider
  - maps editor inputs to documents and annotation models
  - tracks and communicates changes to the editor inputs into editor understandable events (**`IElementChangeListener`**)
  - manages save, dirty state, modification stamps, encoding
  - provides uniform access to editor inputs and their underlying elements

Where Do Documents Come From? (2/2)

```
ITextFileBufferManager mgr= FileBuffers.getTextFileBufferManager();
mgr.connect(IPath, getProgressMonitor());
ITextFileBuffer fileBuffer= mgr.getTextFileBuffer(location);
IDocument document= fileBuffer.getDocument();
IDocument document= fileBuffer.getAnnotationModel();
```

- File buffers
  - provide access to file document and annotation model
  - can be used headless i.e. no editor or document provider needed
  - important step is to call **`connect(…)`** and **`disconnect(…)`**

Configuration Areas

- Editor superclass
  - RCP vs. IDE
- Editor action bar contributor
  - use existing class as is
- Source viewer configuration
  - syntax coloring, hovers, content assist...
- Text model
  - partitioning
  - domain model

Text Editors in a RCP

- Text editors are considered optional RCP plug-ins
  - ➔ not part of the RCP download/distribution
  - **`org.eclipse.text`**
  - **`org.eclipse.jface.text`**
  - **`org.eclipse.ui.workbench.texteditor`**
- Why is **`org.eclipse.ui.editors`** not part of it?
  - ➔ depends on **`org.eclipse.ui.ide`** which drags in UI layout and elements

Tips & Tricks

- Don't forget the **action contributor**
- Set **key binding context**
- Source viewer configuration and document must use the same **partitioning**

Eclipse 3.3 Outlook

- Push down more features from JDT Text to Platform Text
  - extended template support (typed variables)
  - spell checking (finally)
- Complete text editing feature set
  - triple-click, 1.5-click
- More extensibility
  - contributable rulers
  - hyperlink extension point

Thank You – Questions

Legal Notices

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.