JFace – Content Assist – One Trace
Use Concept Trace Slicing

Supporting Applications : [JSP Editor]

```
import org.eclipse.jface.text.contentassist.IContentAssistProcessor;
import org.eclipse.jface.text.reconciler.DirtyRegion;
import org.eclipse.jface.util.SafeRunnable;
import org.eclipse.jface.text.source.IOverviewRuler;
import org.eclipse.jface.viewers.ISelectionProvider;
import org.eclipse.jface.text.IInformationControlCreator;
import org.eclipse.jface.window.IShellProvider;
import org.eclipse.jface.text.source.ICharacterPairMatcher;
import org.eclipse.jface.text.reconciler.IReconcilableModel;
import org.eclipse.jface.text.source.IVerticalRuler;
import org.eclipse.jface.text.source.SourceViewerConfiguration;
import org.eclipse.jface.text.IDocumentRewriteSessionListener;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.jface.resource.JFaceResources;
import org.eclipse.swt.graphics.Image;
import org.eclipse.jface.text.ITextHoverExtension;
import org.eclipse.jface.text.source.Annotation;
import org.eclipse.jface.text.ITypedRegion;
import org.eclipse.jface.text.source.AnnotationModel;
import java.util.List;
import org.eclipse.jface.text.ISelectionValidator;
import org.eclipse.jface.text.contentassist.ICompletionProposalExtension;
import org.eclipse.jface.text.source.IAnnotationHover;
import org.eclipse.swt.graphics.RGB;
import org.eclipse.jface.preference.IPreferenceStore;
import org.eclipse.jface.viewers.IPostSelectionProvider;
import java.util.Iterator;
import org.eclipse.jface.text.ITextInputListener;
import org.eclipse.jface.text.IDocumentListener;
import org.eclipse.jface.viewers.SelectionChangedEvent;
import org.eclipse.jface.text.contentassist.ICompletionProposal;
import org.eclipse.jface.text.TextAttribute;
import org.eclipse.jface.text.ITextHover;
import org.eclipse.jface.text.reconciler.IReconcileResult;
import org.eclipse.jface.text.TextUtilities;
import java.util.Map;
import org.eclipse.jface.util.ListenerList;
import org.eclipse.jface.text.reconciler.IReconcilingStrategyExtension;
import org.eclipse.jface.text.reconciler.AbstractReconcileStep;
import org.eclipse.jface.util.IPropertyChangeListener;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.text.IUndoManager;
import org.eclipse.jface.text.source.SourceViewer;
import org.eclipse.swt.graphics.Color;
import org.eclipse.jface.text.hyperlink.IHyperlinkPresenter;
import org.eclipse.jface.text.hyperlink.IHyperlinkDetector;
import org.eclipse.jface.text.contentassist.CompletionProposal;
import org.eclipse.jface.text.DocumentEvent;
import org.eclipse.jface.text.reconciler.IReconcilerExtension;
import org.eclipse.swt.custom.StyledText;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.swt.graphics.Device;
import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.text.contentassist.ContentAssistant;
import org.eclipse.swt.widgets.Control;
import org.eclipse.jface.text.source.IAnnotationModel;
import org.eclipse.jface.text.IRegion;
import org.eclipse.jface.text.reconciler.IReconciler;
import org.eclipse.jface.text.contentassist.ICompletionProposalExtension2;
import org.eclipse.jface.text.information.IInformationPresenter;
import org.eclipse.jface.viewers.ISelectionChangedListener;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.jface.text.information.InformationPresenter;
import org.eclipse.jface.text.source.IVerticalRulerInfo;
import org.eclipse.jface.text.contentassist.IContentAssistant;
import org.eclipse.jface.text.reconciler.IReconcilingStrategy;
import org.eclipse.jface.text.IAutoEditStrategy;
import org.eclipse.jface.action.Action;

public class AppCompletionProposal
implements ICompletionProposal, ICompletionProposalExtension2, ICompletionProposalExtension  {

      public  AppCompletionProposal() {
            CompletionProposal completionProposal = new CompletionProposal(image);
      }

      public String getAdditionalProposalInfo() {
            String string8 = completionProposal.getAdditionalProposalInfo();
      }

      public void selected(sourceViewer) {
      }

      public String getDisplayString() {
            String string = appCompletionProposal.getDisplayString();   // REPEATED!
      }

      public Image getImage() {
            Image image2 = completionProposal.getImage();   // REPEATED!
      }

      public void unselected(sourceViewer) {
      }
}

public class AppCharacterPairMatcher
implements ICharacterPairMatcher  {

      public IRegion match(idocument) {
            char app_char = idocument.getChar(int);   // REPEATED!
      }
}

public class AppSelectionChangedListener
implements ISelectionChangedListener  {

      public void selectionChanged(SelectionChangedEvent) {
            ISelectionChangedListener.selectionChanged(SelectionChangedEvent);      // REPEATED!
            AppSafeRunnable appSafeRunnable = new AppSafeRunnable(appPostSelectionProvider && appSelectionValidator);
            Object[] objectArray = listenerList.getListeners();   // REPEATED!
      }
}

public class AppAbstractReconcileStep
extends AbstractReconcileStep {
```

```
	public IReconcileResult[] reconcileModel(dirtyRegion) {
		/*                      Cyclic Statements                   */
		IReconcilableModel ireconcilableModel = appAbstractReconcileStep.getModel();  // REPEATED!
		IProgressMonitor iprogressMonitor = appAbstractReconcileStep.getProgressMonitor();  // REPEATED!
	}

	public void someMethod() {
		IReconcilableModel ireconcilableModel1 = appAbstractReconcileStep.getInputModel();
	}
}

public class AppContentAssistProcessor
implements IContentAssistProcessor  {

	public  AppContentAssistProcessor() {
		/*                      Cyclic Statements                   */
		String string3 = IPreferenceStore.getString(String);
		boolean app_boolean2 = IPreferenceStore.getBoolean(String);
		IPreferenceStore.addPropertyChangeListener(appPropertyChangeListener);
	}

	public ICompletionProposal[] computeCompletionProposals(sourceViewer) {
		ImageDescriptor.createImage(Device):(Image)||(boolean):(Image)||():(Image)||(boolean,Device):(Image); // REPEATED!
		String string2 = appReconciler.getDocumentPartitioning();
		JFaceResources.getImageRegistry();  // REPEATED!
		ImageDescriptor.getMissingImageDescriptor();    // REPEATED!
		/*                      Cyclic Statements                   */
		String string = appCompletionProposal.getDisplayString();   // REPEATED!
		idocument.get(int,int):(String)||():(String);
		appReconcilingStrategy.reconcile(dirtyRegion);
		DirtyRegion dirtyRegion = new DirtyRegion();
		IDocument idocument = sourceViewer.getDocument();     // REPEATED!
		char app_char = idocument.getChar(int);
		TextUtilities.computePartitioning(string2 && idocument && string6);
		Iterator iterator = iannotationModel.getAnnotationIterator();
		String string4 = ITypedRegion.getType();
		int app_int2 = dirtyRegion.getOffset();
		int app_int4 = dirtyRegion.getLength();
		IAnnotationModel iannotationModel = sourceViewer.getAnnotationModel();  // REPEATED!
		int app_int1 = idocument.getLength();     // REPEATED!
		AppCompletionProposal appCompletionProposal = new AppCompletionProposal(image);
	}

	public String getErrorMessage() {
	}

	public char[] getCompletionProposalAutoActivationCharacters() {
	}

	public char[] getContextInformationAutoActivationCharacters() {
	}
}

public class AppDocumentListener
implements IDocumentListener  {

	public  AppDocumentListener() {
		JFaceResources.getColorRegistry();
		AppPropertyChangeListener appPropertyChangeListener = new AppPropertyChangeListener(appReconcilingStrategyExtension && appDocumentListener);
		/*                      Cyclic Statements                   */
		String string3 = IPreferenceStore.getString(String);
		idocument.get(int,int):(String)||():(String);
		IPreferenceStore.addPropertyChangeListener(appPropertyChangeListener);
		TextAttribute textAttribute = new TextAttribute(color2);
		IDocument.addDocumentListener(appDocumentListener);
	}

	public void documentAboutToBeChanged(DocumentEvent) {
		String string4 = ITypedRegion.getType();
		String string2 = appReconciler.getDocumentPartitioning();
		TextUtilities.computePartitioning(string2 && idocument && string6);
		int app_int1 = idocument.getLength();
		/*                      Cyclic Statements                   */
		int app_int7 = DocumentEvent.getOffset();
		int app_int5 = DocumentEvent.getLength();
	}

	public void documentChanged(DocumentEvent) {
		String string2 = appReconciler.getDocumentPartitioning();
		TextUtilities.computePartitioning(string2 && idocument && string6);
		StyledText styledText = sourceViewer.getTextWidget(); // REPEATED!
		String string4 = ITypedRegion.getType();
		/*                      Cyclic Statements                   */
		boolean app_boolean = textAttribute.equals(textAttribute);  // REPEATED!
		idocument.get(int,int):(String)||():(String);   // REPEATED!
		int app_int5 = DocumentEvent.getLength(); // REPEATED!
		int app_int = textAttribute.getStyle();   // REPEATED!
		DirtyRegion dirtyRegion = new DirtyRegion();
		String string5 = DocumentEvent.getText(); // REPEATED!
		Color color1 = textAttribute.getForeground();   // REPEATED!
		Color color = textAttribute.getBackground();    // REPEATED!
		int app_int7 = DocumentEvent.getOffset(); // REPEATED!
		int app_int8 = idocument.getLineOffset(int);
		int app_int6 = idocument.getLineOfOffset(int);
		int app_int1 = idocument.getLength();     // REPEATED!
	}
}

public class AppAnnotationModel
extends AnnotationModel {

	public void connected() {
	}

	public void removeAnnotations(List,boolean,boolean) {
	}
}

public class AppSourceViewerConfiguration
extends SourceViewerConfiguration {

	public int getHyperlinkStateMask(sourceViewer) {
	}

	public int[] getConfiguredTextHoverStateMasks(sourceViewer) {
	}

	public IInformationPresenter getInformationPresenter(sourceViewer) {
		String string6 = appSourceViewerConfiguration.getConfiguredDocumentPartitioning(sourceViewer);
		AppInformationControlCreator appInformationControlCreator = new AppInformationControlCreator(appSourceViewerConfiguration);
		InformationPresenter informationPresenter = new InformationPresenter(appInformationControlCreator);
		informationPresenter.setSizeConstraints(int,int,boolean,boolean);
		informationPresenter.setDocumentPartitioning(string6);
```

```
	}

	public String getConfiguredDocumentPartitioning(ISourceViewer) {
	}

	public IInformationControlCreator getInformationControlCreator(sourceViewer) {
		AppInformationControlCreator appInformationControlCreator = new AppInformationControlCreator(appSourceViewerConfiguration);
	}

	public IHyperlinkDetector[] getHyperlinkDetectors(sourceViewer) {
	}

	public IContentAssistant getContentAssistant(sourceViewer) {
		ContentAssistant contentAssistant = new ContentAssistant();
		AppContentAssistProcessor appContentAssistProcessor = new AppContentAssistProcessor(appReconcilingStrategyExtension && appDocumentListener);
		/*                      Cyclic Statements                      */
		contentAssistant.setInformationControlCreator(appInformationControlCreator && iinformationControlCreator);
		contentAssistant.setDocumentPartitioning(string8);
		contentAssistant.setContextInformationPopupOrientation(int);
		String string8 = appSourceViewerConfiguration.getConfiguredDocumentPartitioning(sourceViewer);
		contentAssistant.setAutoActivationDelay(int);
		contentAssistant.setProposalPopupOrientation(int);
		IInformationControlCreator iinformationControlCreator = appSourceViewerConfiguration.getInformationControlCreator(sourceViewer);
		contentAssistant.setContentAssistProcessor(appContentAssistProcessor);
		contentAssistant.enableAutoActivation(boolean);
	}

	public IAnnotationHover getAnnotationHover(sourceViewer) {
		AppAnnotationHover appAnnotationHover = new AppAnnotationHover();
	}

	public IHyperlinkPresenter getHyperlinkPresenter(sourceViewer) {
	}

	public IAnnotationHover getOverviewRulerAnnotationHover(sourceViewer) {
	}

	public String[] getIndentPrefixes(sourceViewer) {
		int app_int10 = IPreferenceStore.getInt(String);
		String string3 = IPreferenceStore.getString(String);
	}

	public int getTabWidth(sourceViewer) {
	}

	public IUndoManager getUndoManager(sourceViewer) {
	}

	public IReconciler getReconciler(sourceViewer) {
		AppReconciler appReconciler = new AppReconciler(appSourceViewerConfiguration);
	}

	public IAutoEditStrategy[] getAutoEditStrategies(sourceViewer) {
		AppAutoEditStrategy appAutoEditStrategy = new AppAutoEditStrategy();
	}

	public void getTextHover(sourceViewer) {
		AppTextHoverExtension appTextHoverExtension = new AppTextHoverExtension();
		AppTextHover appTextHover = new AppTextHover();
	}
}

public class AppAutoEditStrategy
implements IAutoEditStrategy  {

	public void customizeDocumentCommand(idocument) {
	}
}

public class AppReconciler
implements IReconciler, IReconcilerExtension  {

	public  AppReconciler() {
		AppDocumentRewriteSessionListener appDocumentRewriteSessionListener = new AppDocumentRewriteSessionListener(appReconciler && ireconciler);
		AppDocumentListener appDocumentListener = new AppDocumentListener(appReconciler && ireconciler);
	}

	public void install(sourceViewer) {
		AppTextInputListener appTextInputListener = new AppTextInputListener(appReconciler && ireconciler);
		sourceViewer.addTextInputListener(appTextInputListener);
	}
}

public class AppTextInputListener
implements ITextInputListener  {

	public void inputDocumentChanged(IDocument,IDocument) {
		AppReconcilingStrategyExtension appReconcilingStrategyExtension = new AppReconcilingStrategyExtension(sourceViewer);
		IDocument idocument = sourceViewer.getDocument();
		AppReconcilingStrategy appReconcilingStrategy = new AppReconcilingStrategy(sourceViewer);
		appReconcilingStrategy.setDocument(idocument);  // REPEATED!
		/*                      Cyclic Statements                      */
		idocument.get(int,int):(String)||():(String);
		idocument.removeDocumentRewriteSessionListener(appDocumentRewriteSessionListener);
		DirtyRegion dirtyRegion = new DirtyRegion();
		IDocument.addDocumentListener(appDocumentListener);
		int app_int1 = idocument.getLength();     // REPEATED!
		idocument.addDocumentRewriteSessionListener(appDocumentRewriteSessionListener);     // REPEATED!
	}

	public void inputDocumentAboutToBeChanged(IDocument,IDocument) {
	}
}

public class AppSafeRunnable
extends SafeRunnable {
}

public class AppAction
extends Action {

	public  AppAction() {
		appAction.setEnabled(boolean);
		appAction.setText(String);
		Control control = IVerticalRulerInfo.getControl();
	}
}

public class AppInformationControlCreator
implements IInformationControlCreator  {
}

public class AppReconcilingStrategy
```

```
implements IReconcilingStrategy  {

      public  AppReconcilingStrategy() {
            AppAbstractReconcileStep appAbstractReconcileStep = new AppAbstractReconcileStep(appReconcilingStrategyExtension);
            AppPropertyChangeListener appPropertyChangeListener = new AppPropertyChangeListener(appReconcilingStrategyExtension && appDocumentListener);
      }

      public void someMethod() {
            AppReconcilableModel appReconcilableModel = new AppReconcilableModel(idocument);
            appAbstractReconcileStep.setInputModel(appReconcilableModel);
            appAbstractReconcileStep.reconcile(dirtyRegion);
            IPreferenceStore.addPropertyChangeListener(appPropertyChangeListener);
            IAnnotationModel iannotationModel = sourceViewer.getAnnotationModel();  // REPEATED!
            /*                    Cyclic Statements                 */
            Iterator iterator = iannotationModel.getAnnotationIterator();
            iannotationModel.replaceAnnotations(Annotation[],Map);
      }
}

public class AppReconcilingStrategyExtension
implements IReconcilingStrategyExtension  {

      public  AppReconcilingStrategyExtension() {
            AppAbstractReconcileStep appAbstractReconcileStep = new AppAbstractReconcileStep(appReconcilingStrategyExtension);
            AppPropertyChangeListener appPropertyChangeListener = new AppPropertyChangeListener(appReconcilingStrategyExtension && appDocumentListener);
      }
}

public class AppReconcilableModel
implements IReconcilableModel  {
}

public class AppShellProvider
implements IShellProvider  {
}

public class AppDocumentRewriteSessionListener
implements IDocumentRewriteSessionListener  {
}

public class AppAnnotationHover
implements IAnnotationHover  {
}

public class AppTextHover
implements ITextHover  {
}

public class AppTextHoverExtension
implements ITextHoverExtension  {
}

public class AppPostSelectionProvider
implements IPostSelectionProvider  {

      public  AppPostSelectionProvider() {
            ListenerList listenerList = new ListenerList(int)||();
            AppSelectionChangedListener appSelectionChangedListener = new AppSelectionChangedListener(appPostSelectionProvider && appSelectionValidator);
            ISelectionProvider.addSelectionChangedListener(appSelectionChangedListener);
            IPostSelectionProvider.addPostSelectionChangedListener(appSelectionChangedListener);
      }
}

public class AppSelectionValidator
implements ISelectionValidator  {

      public  AppSelectionValidator() {
            ListenerList listenerList = new ListenerList(int)||();
            AppSelectionChangedListener appSelectionChangedListener = new AppSelectionChangedListener(appPostSelectionProvider && appSelectionValidator);
            ISelectionProvider.addSelectionChangedListener(appSelectionChangedListener);
            IPostSelectionProvider.addPostSelectionChangedListener(appSelectionChangedListener);
      }
}

public class SomeClass {

      public void someMethod() {
            SourceViewer sourceViewer = new SourceViewer(Composite,IVerticalRuler,int)||(Composite,IVerticalRuler,IOverviewRuler,boolean,int);
            AppReconcilingStrategyExtension appReconcilingStrategyExtension = new AppReconcilingStrategyExtension(sourceViewer);
            AppReconcilingStrategy appReconcilingStrategy = new AppReconcilingStrategy(sourceViewer);
            appReconcilingStrategy.setDocument(idocument);
            ISelection iselection = ISelectionProvider.getSelection();  // REPEATED!
            int app_int3 = iselection.getOffset();    // REPEATED!
            AppPostSelectionProvider appPostSelectionProvider = new AppPostSelectionProvider();
            AppCharacterPairMatcher appCharacterPairMatcher = new AppCharacterPairMatcher();
            AppSourceViewerConfiguration appSourceViewerConfiguration = new AppSourceViewerConfiguration();
            AppShellProvider appShellProvider = new AppShellProvider();
            AppDocumentListener appDocumentListener = new AppDocumentListener(appReconciler && ireconciler);
            AppAction appAction = new AppAction();
            boolean app_boolean1 = appAction.isEnabled();   // REPEATED!
            AppSelectionValidator appSelectionValidator = new AppSelectionValidator();
            listenerList.add(Object);     // REPEATED!
            AppAnnotationModel appAnnotationModel = new AppAnnotationModel();
            IPreferenceStore.setDefault(String,String):()||(String,boolean):()||(String,float):()||(String,double):()||(String,long):()||(String,int):();   // REPEATED!
            /*                    Cyclic Statements                 */
            boolean app_boolean = textAttribute.equals(textAttribute);  // REPEATED!
            int app_int = textAttribute.getStyle();   // REPEATED!
            TextUtilities.computePartitioning(string2 && idocument && string6);     // REPEATED!
            Color color = textAttribute.getBackground();    // REPEATED!
            IReconcilingStrategy ireconcilingStrategy = appReconciler.getReconcilingStrategy(string4);
            Iterator iterator = iannotationModel.getAnnotationIterator();
            String string4 = ITypedRegion.getType();  // REPEATED!
            int app_int2 = dirtyRegion.getOffset();   // REPEATED!
            int app_int4 = dirtyRegion.getLength();   // REPEATED!
            idocument.get(int,int):(String)||():(String);   // REPEATED!
            appReconcilingStrategy.reconcile(dirtyRegion);
            DirtyRegion dirtyRegion = new DirtyRegion();
            String string2 = appReconciler.getDocumentPartitioning();   // REPEATED!
            StyledText styledText = sourceViewer.getTextWidget(); // REPEATED!
            Color color1 = textAttribute.getForeground();   // REPEATED!
            IAnnotationModel iannotationModel = sourceViewer.getAnnotationModel();  // REPEATED!
            int app_int6 = idocument.getLineOfOffset(int);  // REPEATED!
            int app_int1 = idocument.getLength();     // REPEATED!
      }
}
```

Description of False Negatives:

We have one false negative because of not calling the following instruction since it is in org.eclipse.ui, not in JFace:
TextEditor.setSourceViewerConfiguration(appSourceViewerConfiguration);