Eclipse – Navigation – One Trace
Use Full Trace (No Slicing)

Supporting Applications : [SVN]

```
import org.eclipse.ui.progress.DeferredTreeContentManager;
import org.eclipse.jface.action.Separator;
import org.eclipse.core.runtime.NullProgressMonitor;
import org.eclipse.jface.action.IMenuListener;
import org.eclipse.ui.IActionBars;
import org.eclipse.swt.graphics.Font;
import org.eclipse.team.core.variants.IResourceVariantComparator;
import org.osgi.service.prefs.Preferences;
import org.eclipse.ui.IViewActionDelegate;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.jface.action.IStatusLineManager;
import org.eclipse.core.resources.IResource;
import org.eclipse.jface.resource.FontRegistry;
import org.eclipse.team.core.variants.SessionResourceVariantByteStore;
import org.eclipse.ui.IObjectActionDelegate;
import org.eclipse.ui.model.WorkbenchContentProvider;
import org.eclipse.ui.model.WorkbenchLabelProvider;
import org.eclipse.core.runtime.Preferences;
import org.eclipse.team.ui.TeamImages;
import org.eclipse.ui.actions.SelectionListenerAction;
import org.eclipse.team.core.RepositoryProvider;
import org.eclipse.ui.help.IWorkbenchHelpSystem;
import org.eclipse.ui.IViewPart;
import org.eclipse.team.ui.TeamOperation;
import org.eclipse.ui.IWorkbenchPartSite;
import org.eclipse.jface.preference.PreferenceConverter;
import org.eclipse.swt.graphics.RGB;
import org.eclipse.core.runtime.Plugin;
import org.eclipse.jface.preference.IPreferenceStore;
import org.eclipse.ui.themes.ITheme;
import java.util.Iterator;
import org.eclipse.core.runtime.IAdapterFactory;
import org.eclipse.jface.viewers.IDoubleClickListener;
import org.eclipse.ui.themes.IThemeManager;
import org.eclipse.jface.viewers.SelectionChangedEvent;
import java.io.File;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.jface.action.IToolBarManager;
import org.eclipse.core.runtime.IPath;
import org.eclipse.core.resources.ISynchronizer;
import org.eclipse.core.runtime.Preferences.IPropertyChangeListener;
import org.eclipse.jface.viewers.TreeViewer;
import org.eclipse.team.ui.history.HistoryPageSource;
import org.eclipse.core.runtime.jobs.ISchedulingRule;
import org.eclipse.jface.viewers.ILightweightLabelDecorator;
import java.text.Collator;
import org.eclipse.ui.ISelectionListener;
import org.eclipse.swt.widgets.Display;
import org.eclipse.core.runtime.Platform;
import org.eclipse.ui.actions.ActionFactory;
import org.osgi.framework.Bundle;
import org.eclipse.team.core.subscribers.Subscriber;
import org.eclipse.jface.util.IPropertyChangeListener;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.window.SameShellProvider;
import org.eclipse.core.resources.IProject;
import org.eclipse.ui.dialogs.PropertyDialogAction;
import org.eclipse.ui.part.PluginTransfer;
import org.eclipse.ui.model.IWorkbenchAdapter;
import org.eclipse.core.runtime.SubProgressMonitor;
import org.eclipse.core.resources.IResourceChangeListener;
import org.eclipse.swt.events.KeyAdapter;
import org.eclipse.swt.graphics.Color;
import org.eclipse.ui.ISelectionService;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.part.DrillDownAdapter;
import org.eclipse.ui.progress.IDeferredWorkbenchAdapter;
import org.eclipse.team.internal.core.subscribers.SubscriberChangeSetManager;
import org.eclipse.swt.widgets.Tree;
import org.eclipse.ui.PlatformUI;
import org.eclipse.core.resources.ISaveParticipant;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.core.resources.ISavedState;
import org.eclipse.team.core.variants.IResourceVariant;
import org.eclipse.team.core.variants.CachedResourceVariant;
import org.eclipse.swt.widgets.Control;
import org.eclipse.ui.actions.ActionDelegate;
import org.eclipse.swt.dnd.DragSourceListener;
import org.eclipse.swt.dnd.Transfer;
import org.eclipse.core.runtime.IAdaptable;
```

```
import org.eclipse.swt.dnd.ByteArrayTransfer;
import org.eclipse.ui.IWorkbenchPart;
import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.eclipse.jface.resource.ColorRegistry;
import org.eclipse.core.runtime.QualifiedName;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.jface.viewers.ViewerSorter;
import org.eclipse.jface.viewers.ISelectionChangedListener;
import org.eclipse.ui.IViewSite;
import org.eclipse.core.runtime.preferences.InstanceScope;
import org.eclipse.core.runtime.preferences.IEclipsePreferences;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.ui.console.IOConsole;
import org.eclipse.jface.action.Action;

public class AppWorkbenchAdapter
implements IWorkbenchAdapter, IAdaptable  {

      public  AppWorkbenchAdapter() {
            AppWorkbenchAdapter appWorkbenchAdapter = new AppWorkbenchAdapter();
            AppCachedResourceVariant appCachedResourceVariant = new AppCachedResourceVariant(appWorkbenchAdapter);
            AppResourceVariant appResourceVariant = new AppResourceVariant(appWorkbenchAdapter);
      }

      public Object[] getChildren(appWorkbenchAdapter && object) {
            String[] stringArray = Preferences.childrenNames();   // REPEATED!
            AppWorkbenchAdapter appWorkbenchAdapter = new AppWorkbenchAdapter();
            NullProgressMonitor nullProgressMonitor = new NullProgressMonitor();    // REPEATED!
            nullProgressMonitor.beginTask(String,int);      // REPEATED!
            nullProgressMonitor.done();   // REPEATED!
            InstanceScope instanceScope = new InstanceScope();    // REPEATED!
            IEclipsePreferences ieclipsePreferences = instanceScope.getNode(String);      // REPEATED!
            /*                    Cyclic Statements                   */
            IPath ipath = Plugin.getStateLocation();  // REPEATED!
            ipath.append(String):(IPath)||(IPath):(IPath);  // REPEATED!
            boolean app_boolean4 = Plugin.isDebugging();
            Bundle bundle = Plugin.getBundle(); // REPEATED!
      }

      public Object getAdapter(Class) {
            AdapterManager adapterManager = Platform.getAdapterManager();     // REPEATED!
            adapterManager.getAdapter(appWorkbenchAdapter && object1);  // REPEATED!
      }
}

public class AppLightweightLabelDecorator
implements ILightweightLabelDecorator  {

      public void decorate(iproject) {
            /*                    Cyclic Statements                   */
            RepositoryProvider.getProvider(iproject); // REPEATED!
            IProject iproject = IResource.getProject();     // REPEATED!
            int app_int = iproject.getType();   // REPEATED!
      }
}

public class AppWorkbenchContentProvider
extends WorkbenchContentProvider {

      public Object[] getChildren(appWorkbenchAdapter && object && object1) {
            Object[] objectArray1 = deferredTreeContentManager.getChildren(appWorkbenchAdapter && object && object1);   // REPEATED!
      }

      public boolean hasChildren(appWorkbenchAdapter && object1) {
            boolean app_boolean8 = deferredTreeContentManager.isDeferredAdapter(Object);
      }

      public void inputChanged(appWorkbenchAdapter && treeViewer && object && object1) {
            DeferredTreeContentManager deferredTreeContentManager = new DeferredTreeContentManager(appWorkbenchContentProvider
                                                                                                    && treeViewer);   // REPEATED!
      }
}

public class AppWorkbenchPart
implements IWorkbenchPart, ISelectionListener  {

      public void setFocus() {
            Control control = treeViewer.getControl();
            boolean app_boolean9 = control.setFocus();
      }

      public void createPartControl(Composite) {
            AppByteArrayTransfer appByteArrayTransfer = new AppByteArrayTransfer();
            AppViewActionDelegate appViewActionDelegate = new AppViewActionDelegate();
            AppActionDelegate appActionDelegate = new AppActionDelegate();
            AppragSourceListener appragSourceListener = new AppragSourceListener();
            MenuManager menuManager = new MenuManager(String)||(String,String)||();
            AppKeyAdapter appKeyAdapter = new AppKeyAdapter();
            AppObjectActionDelegate appObjectActionDelegate = new AppObjectActionDelegate();
            AppWorkbenchAdapter appWorkbenchAdapter = new AppWorkbenchAdapter();
```

```
            PlatformUI.getWorkbench();    // REPEATED!
            AppWorkbenchContentProvider appWorkbenchContentProvider = new AppWorkbenchContentProvider();
            ViewerSorter viewerSorter = new ViewerSorter(Collator)||();
            AppDoubleClickListener appDoubleClickListener = new AppDoubleClickListener();
            PluginTransfer.getInstance();
            WorkbenchLabelProvider workbenchLabelProvider = new WorkbenchLabelProvider();
            TreeViewer treeViewer = new TreeViewer(Composite,int)||(Tree)||(Composite);
            treeViewer.setContentProvider(appWorkbenchContentProvider);
            treeViewer.setLabelProvider(workbenchLabelProvider);
            treeViewer.setInput(appWorkbenchAdapter);
            treeViewer.setSorter(viewerSorter);
            DrillDownAdapter drillDownAdapter = new DrillDownAdapter(treeViewer);
            String string1 = ActionFactory.getId();   // REPEATED!
            Separator separator = new Separator(String)||();
            AppMenuListener appMenuListener = new AppMenuListener();
            Transfer.registerType();
            AppSelectionChangedListener appSelectionChangedListener = new AppSelectionChangedListener();
            /*                      Cyclic Statements                   */
            ISelection iselection1 = treeViewer.getSelection();   // REPEATED!
            itoolBarManager.update(boolean);
            iactionBars.setGlobalActionHandler(string1 && appAction && propertyDialogAction && appSelectionListenerAction);   // REPEATED!
            iworkbenchHelpSystem.setHelp(appAction && control && tree && appSelectionListenerAction); // REPEATED!
            control.setMenu(menu);
            int app_int1 = iselection1.size();
            treeViewer.addDragSupport(appragSourceListener);
            AppSelectionListenerAction appSelectionListenerAction = new AppSelectionListenerAction(shell);
            propertyDialogAction.setEnabled(boolean); // REPEATED!
            appAction.setDisabledImageDescriptor();   // REPEATED!
            IToolBarManager itoolBarManager = iactionBars.getToolBarManager();
            IWorkbenchWindow iworkbenchWindow = iworkbenchPartSite.getWorkbenchWindow();
            Shell shell = control.getShell();   // REPEATED!
            IViewSite iviewSite = IViewPart.getViewSite();  // REPEATED!
            menuManager.addMenuListener(appMenuListener);
            iworkbenchPartSite.setSelectionProvider(treeViewer);
            treeViewer.addSelectionChangedListener(appSelectionChangedListener && appSelectionListenerAction);    // REPEATED!
            appAction.setHoverImageDescriptor();      // REPEATED!
            Menu menu = menuManager.createContextMenu(control && tree);
            iactionBars.updateActionBars();
            appSelectionListenerAction.selectionChanged(SelectionChangedEvent);
            treeViewer.addDoubleClickListener(appDoubleClickListener);
            menuManager.setRemoveAllWhenShown(boolean);
            appAction.setToolTipText(String);   // REPEATED!
            IActionBars iactionBars = iworkbenchPartSite.getActionBars();     // REPEATED!
            itoolBarManager.add(separator && appAction);    // REPEATED!
            IWorkbenchPartSite iworkbenchPartSite = IWorkbenchPart.getSite(); // REPEATED!
            AppAction appAction = new AppAction(shell);
            ISelectionService iselectionService = iworkbenchWindow.getSelectionService();
            control.addKeyListener(appKeyAdapter);
            drillDownAdapter.addNavigationActions(itoolBarManager);
            iselectionService.addPostSelectionListener(ISelectionListener):()||(String,ISelectionListener):();
            iworkbenchPartSite.registerContextMenu(treeViewer && menuManager);
            Tree tree = treeViewer.getTree();   // REPEATED!
            Control control = treeViewer.getControl();      // REPEATED!
            PropertyDialogAction propertyDialogAction = new PropertyDialogAction(treeViewer && sameShellProvider);
            SameShellProvider sameShellProvider = new SameShellProvider(shell);
      }

      public void selectionChanged(iselection2) {
            IViewSite iviewSite = IViewPart.getViewSite();  // REPEATED!
            IActionBars iactionBars = iviewSite.getActionBars();  // REPEATED!
            IStatusLineManager istatusLineManager = iactionBars.getStatusLineManager();   // REPEATED!
            istatusLineManager.setMessage(string);    // REPEATED!
            /*                      Cyclic Statements                   */
            Object object2 = object1.getAdapter(Class);
            int app_int1 = iselection1.size();
            Object object1 = iselection.getFirstElement();
            boolean app_boolean2 = iselection.isEmpty();    // REPEATED!
      }
}

public class AppDeferredWorkbenchAdapter
implements IDeferredWorkbenchAdapter  {

      public ImageDescriptor getImageDescriptor(appWorkbenchAdapter) {
      }

      public ISchedulingRule getRule(appWorkbenchAdapter && object1) {
            AppSchedulingRule appSchedulingRule = new AppSchedulingRule(appWorkbenchAdapter && object1);
      }

      public String getLabel(appWorkbenchAdapter && object1) {
      }
}

public class AppViewActionDelegate
implements IViewActionDelegate  {

      public void init(IViewPart) {
            IWorkbenchPartSite iworkbenchPartSite = IWorkbenchPart.getSite();
            Shell shell1 = iworkbenchPartSite.getShell();
```

```
	}

	public void selectionChanged(iselection2) {
	}
}

public class AppSubscriber
extends Subscriber {

	public  AppSubscriber() {
		AppResourceVariantComparator appResourceVariantComparator = new AppResourceVariantComparator();
		SessionResourceVariantByteStore sessionResourceVariantByteStore = new SessionResourceVariantByteStore();
	}

	public String getName() {
	}
}

public class AppSelectionListenerAction
extends SelectionListenerAction {

	public boolean updateSelection(iselection) {
		AppSubProgressMonitor appSubProgressMonitor = new AppSubProgressMonitor();
		boolean app_boolean2 = iselection.isEmpty();
		Iterator iterator = iselection.iterator();
	}
}

public class AppSelectionChangedListener
implements ISelectionChangedListener  {

	public void selectionChanged(SelectionChangedEvent) {
		AppSubProgressMonitor appSubProgressMonitor = new AppSubProgressMonitor();
		appSubProgressMonitor.beginTask(String,int);
		ISelection iselection = SelectionChangedEvent.getSelection();     // REPEATED!
		int app_int1 = iselection.size();
		Object object1 = iselection.getFirstElement();
		propertyDialogAction.setEnabled(boolean); // REPEATED!
	}
}

public class AppSelectionChangedEvent
extends SelectionChangedEvent {

	public ISelection getSelection() {
		AppTeamOperation appTeamOperation = new AppTeamOperation(appWorkbenchAdapter && appCachedResourceVariant && appResourceVariant);
	}
}

public class AppSubProgressMonitor
extends SubProgressMonitor {

	public  AppSubProgressMonitor() {
		ISelection iselection = SelectionChangedEvent.getSelection();
		boolean app_boolean2 = iselection.isEmpty();
	}

	public void setTaskName(String) {
	}
}

public class AppProgressMonitor
implements IProgressMonitor  {

	public void beginTask(String,int) {
		propertyDialogAction.setEnabled(boolean); // REPEATED!
	}
}

public class AppImageDescriptor
extends ImageDescriptor {
}

public class AppAction
extends Action {

	public  AppAction() {
		Preferences preferences = Plugin.getPluginPreferences();
		boolean app_boolean3 = preferences.getBoolean(String);
		boolean app_boolean4 = Plugin.isDebugging();
		appAction.setText(String);
		appAction.setChecked(boolean);
	}
}

public class AppResourceChangeListener
implements IResourceChangeListener  {

	public  AppResourceChangeListener() {
		QualifiedName qualifiedName = new QualifiedName(String,String);
```

```
            Preferences preferences = Plugin.getPluginPreferences();
            boolean app_boolean3 = preferences.getBoolean(String);
            ResourcesPlugin.getWorkspace();
            ISynchronizer isynchronizer = IResourceChangeListener.getSynchronizer();
            isynchronizer.add(qualifiedName);
      }
}

public class AppPropertyChangeListener
implements IPropertyChangeListener  {

      public  AppPropertyChangeListener() {
            ImageDescriptor.createFromURL();
            IPreferenceStore ipreferenceStore = AbstractUIPlugin.getPreferenceStore();
            /*                    Cyclic Statements                    */
            int app_int2 = ipreferenceStore.getInt(String);
            boolean app_boolean = ipreferenceStore.getBoolean(String);
            ipreferenceStore.addPropertyChangeListener(ioConsole && appPropertyChangeListener);
      }
}

public class AppAdapterFactory
implements IAdapterFactory  {

      public  AppAdapterFactory() {
            AppDeferredWorkbenchAdapter appDeferredWorkbenchAdapter = new AppDeferredWorkbenchAdapter();
            AppWorkbenchAdapter appWorkbenchAdapter = new AppWorkbenchAdapter();
            AppHistoryPageSource appHistoryPageSource = new AppHistoryPageSource();
      }
}

public class AppSaveParticipant
implements ISaveParticipant  {
}

public class AppResourceVariant
implements IResourceVariant  {
}

public class AppTeamOperation
extends TeamOperation {
}

public class AppHistoryPageSource
extends HistoryPageSource {
}

public class AppMenuListener
implements IMenuListener  {
}

public class AppActionDelegate
extends ActionDelegate {
}

public class AppObjectActionDelegate
implements IObjectActionDelegate  {
}

public class AppKeyAdapter
extends KeyAdapter {
}

public class AppDoubleClickListener
implements IDoubleClickListener  {
}

public class AppragSourceListener
implements DragSourceListener  {
}

public class AppByteArrayTransfer
extends ByteArrayTransfer {
}

public class AppResourceVariantComparator
implements IResourceVariantComparator  {
}

public class AppCachedResourceVariant
extends CachedResourceVariant {
}

public class AppSchedulingRule
implements ISchedulingRule  {
}

public class AppPropertyChangeListener
implements IPropertyChangeListener  {
```

```
	public AppPropertyChangeListener() {
		QualifiedName qualifiedName = new QualifiedName(String,String);
		Preferences preferences = Plugin.getPluginPreferences();
		boolean app_boolean3 = preferences.getBoolean(String);
		ResourcesPlugin.getWorkspace();
		isynchronizer.add(qualifiedName);
	}
}

public class SomeClass {

	public void someMethod() {
		IOConsole ioConsole = new
IOConsole(String,String,ImageDescriptor,String,boolean)||(String,ImageDescriptor)||(String,String,ImageDescriptor,boolean);
		AppResourceChangeListener appResourceChangeListener = new AppResourceChangeListener();
		ResourcesPlugin.getWorkspace();     // REPEATED!
		TeamImages.getImageDescriptor();    // REPEATED!
		AppImageDescriptor appImageDescriptor = new AppImageDescriptor();
		appSelectionListenerAction.selectionChanged(SelectionChangedEvent);     // REPEATED!
		boolean app_boolean10 = appSelectionListenerAction.updateSelection(iselection2);
		Preferences preferences = Plugin.getPluginPreferences();

	preferences.setDefault(String,int):()||(String,boolean):()||(String,double):()||(String,String):()||(String,float):()||(String,long):();
		int app_int1 = iselection1.size();
		String string4 = appWorkbenchAdapter.getLabel(appWorkbenchAdapter && object1);      // REPEATED!
		AppAction appAction = new AppAction(shell);
		boolean app_boolean5 = appAction.isChecked();
		nullProgressMonitor.beginTask(String,int);      // REPEATED!
		AppSubscriber appSubscriber = new AppSubscriber();
		SubscriberChangeSetManager subscriberChangeSetManager = new SubscriberChangeSetManager(appSubscriber);
		boolean app_boolean11 = appWorkbenchContentProvider.hasChildren(Object);
		AppPropertyChangeListener appPropertyChangeListener = new AppPropertyChangeListener();
		preferences.addPropertyChangeListener(appResourceChangeListener && appPropertyChangeListener);
		preferences.setValue(String,int):()||(String,String):()||(String,double):()||(String,boolean):()||(String,float):()||(String,long):();
		Platform.getAdapterManager(); // REPEATED!
		PlatformUI.getWorkbench();
		FontRegistry fontRegistry = itheme.getFontRegistry();
		itheme.addPropertyChangeListener(IPropertyChangeListener);
		Font font = fontRegistry.get(String);
		Color color = colorRegistry.get(String);
		IPreferenceStore ipreferenceStore = AbstractUIPlugin.getPreferenceStore();    // REPEATED!
		AppPropertyChangeListener appPropertyChangeListener = new AppPropertyChangeListener(ipreferenceStore);
		AppSaveParticipant appSaveParticipant = new AppSaveParticipant();
		RGB rGB = new RGB(float,float,float)||(int,int,int);  // REPEATED!
		PreferenceConverter.setDefault(rGB && ipreferenceStore);    // REPEATED!
		AppAdapterFactory appAdapterFactory = new AppAdapterFactory();
		Display.getCurrent();
		boolean app_boolean7 = appSchedulingRule.isConflicting(ischedulingRule && appSchedulingRule);
		Display.getDefault();
		/*                    Cyclic Statements                  */
		String string = ipreferenceStore.getString(String);   // REPEATED!
		ipreferenceStore.addPropertyChangeListener(ioConsole && appPropertyChangeListener);
		Platform.getAuthorizationInfo(string);    // REPEATED!
		ipreferenceStore.setDefault(String,String):()||(String,boolean):()||(String,float):()||(String,double):()||
                                                                (String,long):()||(String,int):();  // REPEATED!
		boolean app_boolean = ipreferenceStore.getBoolean(String);  // REPEATED!
	}
}
```