extensions.org.eclipse.ui.actions.TextActionHandler.java - src.org.eclipse.ui.internal.navigator.TextActionHandler.java

```java
/*******************************************************************************
 * Copyright (c) 2000, 2008 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *     IBM Corporation - initial API and implementation
 *******************************************************************************/
package org.eclipse.ui.actions;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.util.IPropertyChangeListener;
import org.eclipse.jface.util.PropertyChangeEvent;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyAdapter;
import org.eclipse.swt.events.KeyEvent;
import org.eclipse.swt.events.MouseAdapter;
import org.eclipse.swt.events.MouseEvent;
import org.eclipse.swt.graphics.Point;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IActionBars;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.internal.ide.IDEWorkbenchMessages;
import org.eclipse.ui.internal.ide.IIDEHelpContextIds;

/**
 * Handles the redirection of the global Cut, Copy, Paste, and
 * Select All actions to either the current inline text control
 * or the part's supplied action handler.
 * <p>
 * This class may be instantiated; it is not intended to be subclassed.
 * </p><p>
 * Example usage:
 * <pre>
 * textActionHandler = new TextActionHandler(this.getViewSite().getActionBars());
 * textActionHandler.addText((Text)textCellEditor1.getControl());
 * textActionHandler.addText((Text)textCellEditor2.getControl());
 * textActionHandler.setSelectAllAction(selectAllAction);
 * </pre>
 * </p>
 * @noextend This class is not intended to be subclassed by clients.
 */
public class TextActionHandler {
    private DeleteActionHandler textDeleteAction = new DeleteActionHandler();

    private CutActionHandler textCutAction = new CutActionHandler();

    private CopyActionHandler textCopyAction = new CopyActionHandler();

    private PasteActionHandler textPasteAction = new PasteActionHandler();

```

```java
/*******************************************************************************
 * Copyright (c) 2005, 2006 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *     IBM Corporation - initial API and implementation
 *******************************************************************************/
package org.eclipse.ui.internal.navigator;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.util.IPropertyChangeListener;
import org.eclipse.jface.util.PropertyChangeEvent;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyAdapter;
import org.eclipse.swt.events.KeyEvent;
import org.eclipse.swt.events.MouseAdapter;
import org.eclipse.swt.events.MouseEvent;

import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IActionBars;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.actions.ActionFactory;


/**
 * Handles the redirection of the global Cut, Copy, Paste, and
 * Select All actions to either the current inline text control
 * or the part's supplied action handler.
 * <p>
 * This class may be instantiated; it is not intended to be subclassed.
 * </p><p>
 * Example usage:
 * <pre>
 * textActionHandler = new TextActionHandler(this.getViewSite().getActionBars());
 * textActionHandler.addText((Text)textCellEditor1.getControl());
 * textActionHandler.addText((Text)textCellEditor2.getControl());
 * textActionHandler.setSelectAllAction(selectAllAction);
 * </pre>
 * </p>

 */
public class TextActionHandler {
    private DeleteActionHandler textDeleteAction = new DeleteActionHandler();

    private CutActionHandler textCutAction = new CutActionHandler();

    private CopyActionHandler textCopyAction = new CopyActionHandler();

    private PasteActionHandler textPasteAction = new PasteActionHandler();

```

extensions.org.eclipse.ui.actions.TextActionHandler.java - src.org.eclipse.ui.internal.navigator.TextActionHandler.java

```
57    private SelectAllActionHandler textSelectAllAction = new
  SelectAllActionHandler();
58
59    private IAction deleteAction;
60
61    private IAction cutAction;
62
63    private IAction copyAction;
64
65    private IAction pasteAction;
66
67    private IAction selectAllAction;
68
69    private IPropertyChangeListener deleteActionListener = new
  PropertyChangeListener(
70            textDeleteAction);
71
72    private IPropertyChangeListener cutActionListener = new PropertyChangeListener(
73            textCutAction);
74
75    private IPropertyChangeListener copyActionListener = new
  PropertyChangeListener(
76            textCopyAction);
77
78    private IPropertyChangeListener pasteActionListener = new
  PropertyChangeListener(
79            textPasteAction);
80
81    private IPropertyChangeListener selectAllActionListener = new
  PropertyChangeListener(
82            textSelectAllAction);
83
84    private Listener textControlListener = new TextControlListener();
85
86    private Text activeTextControl;
87
88    private MouseAdapter mouseAdapter = new MouseAdapter() {
89        public void mouseUp(MouseEvent e) {
90            updateActionsEnableState();
91        }
92    };
93
94    private KeyAdapter keyAdapter = new KeyAdapter() {
95        public void keyReleased(KeyEvent e) {
96            updateActionsEnableState();
97        }
98    };
99
100   private class TextControlListener implements Listener {
101       public void handleEvent(Event event) {
102           switch (event.type) {
103           case SWT.Activate:
104               activeTextControl = (Text) event.widget;
105               updateActionsEnableState();
106               break;
107           case SWT.Deactivate:
```

```
54    private SelectAllActionHandler textSelectAllAction = new
  SelectAllActionHandler();
55
56    private IAction deleteAction;
57
58    private IAction cutAction;
59
60    private IAction copyAction;
61
62    private IAction pasteAction;
63
64    private IAction selectAllAction;
65
66    private IPropertyChangeListener deleteActionListener = new
  PropertyChangeListener(
67            textDeleteAction);
68
69    private IPropertyChangeListener cutActionListener = new PropertyChangeListener(
70            textCutAction);
71
72    private IPropertyChangeListener copyActionListener = new
  PropertyChangeListener(
73            textCopyAction);
74
75    private IPropertyChangeListener pasteActionListener = new
  PropertyChangeListener(
76            textPasteAction);
77
78    private IPropertyChangeListener selectAllActionListener = new
  PropertyChangeListener(
79            textSelectAllAction);
80
81    private Listener textControlListener = new TextControlListener();
82
83    private Text activeTextControl;
84
85    private MouseAdapter mouseAdapter = new MouseAdapter() {
86        public void mouseUp(MouseEvent e) {
87            updateActionsEnableState();
88        }
89    };
90
91    private KeyAdapter keyAdapter = new KeyAdapter() {
92        public void keyReleased(KeyEvent e) {
93            updateActionsEnableState();
94        }
95    };
96
97    private class TextControlListener implements Listener {
98        public void handleEvent(Event event) {
99            switch (event.type) {
100           case SWT.Activate:
101               activeTextControl = (Text) event.widget;
102               updateActionsEnableState();
103               break;
104           case SWT.Deactivate:
```

extensions.org.eclipse.ui.actions.TextActionHandler.java - src.org.eclipse.ui.internal.navigator.TextActionHandler.java

```
                activeTextControl = null;
                updateActionsEnableState();
                break;
            default:
                break;
            }
        }
    }

    private class PropertyChangeListener implements IPropertyChangeListener {
        private IAction actionHandler;

        protected PropertyChangeListener(IAction actionHandler) {
            super();
            this.actionHandler = actionHandler;
        }

        public void propertyChange(PropertyChangeEvent event) {
            if (activeTextControl != null) {
                return;
            }
            if (event.getProperty().equals(IAction.ENABLED)) {
                Boolean bool = (Boolean) event.getNewValue();
                actionHandler.setEnabled(bool.booleanValue());
            }
        }
    }

    private class DeleteActionHandler extends Action {
        protected DeleteActionHandler() {
            super(IDEWorkbenchMessages.Delete);
            setId("TextDeleteActionHandler");//$NON-NLS-1$
            setEnabled(false);
            PlatformUI.getWorkbench().getHelpSystem().setHelp(this,
                    IIDEHelpContextIds.TEXT_DELETE_ACTION);
        }

        public void runWithEvent(Event event) {
            if (activeTextControl != null && !activeTextControl.isDisposed()) {
                String text = activeTextControl.getText();
                Point selection = activeTextControl.getSelection();
                if (selection.y == selection.x) {
                    ++selection.y;
                }
                if (selection.y > text.length()) {
                    return;
                }
                StringBuffer buf = new StringBuffer(text.substring(0,
                        selection.x));
                buf.append(text.substring(selection.y));
                activeTextControl.setText(buf.toString());
                activeTextControl.setSelection(selection.x, selection.x);
                updateActionsEnableState();
                return;
            }
            if (deleteAction != null) {
```

```
                activeTextControl = null;
                updateActionsEnableState();
                break;
            default:
                break;
            }
        }
    }

    private class PropertyChangeListener implements IPropertyChangeListener {
        private IAction actionHandler;

        protected PropertyChangeListener(IAction actionHandler) {
            super();
            this.actionHandler = actionHandler;
        }

        public void propertyChange(PropertyChangeEvent event) {
            if (activeTextControl != null) {
                return;
            }
            if (event.getProperty().equals(IAction.ENABLED)) {
                Boolean bool = (Boolean) event.getNewValue();
                actionHandler.setEnabled(bool.booleanValue());
            }
        }
    }

    private class DeleteActionHandler extends Action {
        protected DeleteActionHandler() {
            super(CommonNavigatorMessages.Delete);
            setId("TextDeleteActionHandler");//$NON-NLS-1$
            setEnabled(false);
            PlatformUI.getWorkbench().getHelpSystem().setHelp(this,
                    INavigatorHelpContextIds.TEXT_DELETE_ACTION);
        }

        public void runWithEvent(Event event) {
            if (activeTextControl != null && !activeTextControl.isDisposed()) {
                activeTextControl.clearSelection();
                return;
            }

            if (deleteAction != null) {
```

extensions.org.eclipse.ui.actions.TextActionHandler.java - src.org.eclipse.ui.internal.navigator.TextActionHandler.java

```
                deleteAction.runWithEvent(event);
                return;
            }
        }

        /**
         * Update state.
         */
        public void updateEnabledState() {
            if (activeTextControl != null && !activeTextControl.isDisposed()) {
                setEnabled(activeTextControl.getSelectionCount() > 0
                        || activeTextControl.getCaretPosition() < activeTextControl
                                .getCharCount());
                return;
            }
            if (deleteAction != null) {
                setEnabled(deleteAction.isEnabled());
                return;
            }
            setEnabled(false);
        }
    }

    private class CutActionHandler extends Action {
        protected CutActionHandler() {
            super(IDEWorkbenchMessages.Cut);
            setId("TextCutActionHandler");//$NON-NLS-1$
            setEnabled(false);
            PlatformUI.getWorkbench().getHelpSystem().setHelp(this,
                    IIDEHelpContextIds.TEXT_CUT_ACTION);
        }

        public void runWithEvent(Event event) {
            if (activeTextControl != null && !activeTextControl.isDisposed()) {
                activeTextControl.cut();
                updateActionsEnableState();
                return;
            }
            if (cutAction != null) {
                cutAction.runWithEvent(event);
                return;
            }
        }

        /**
         * Update state.
         */
        public void updateEnabledState() {
            if (activeTextControl != null && !activeTextControl.isDisposed()) {
                setEnabled(activeTextControl.getSelectionCount() > 0);
                return;
            }
            if (cutAction != null) {
                setEnabled(cutAction.isEnabled());
                return;
            }
```

```
                deleteAction.runWithEvent(event);
                return;
            }
        }

        /**
         * Update state.
         */
        public void updateEnabledState() {
            if (activeTextControl != null && !activeTextControl.isDisposed()) {
                setEnabled(activeTextControl.getSelectionCount() > 0
                        || activeTextControl.getCaretPosition() < activeTextControl
                                .getCharCount());
                return;
            }
            if (deleteAction != null) {
                setEnabled(deleteAction.isEnabled());
                return;
            }
            setEnabled(false);
        }
    }

    private class CutActionHandler extends Action {
        protected CutActionHandler() {
            super(CommonNavigatorMessages.Cut);
            setId("TextCutActionHandler");//$NON-NLS-1$
            setEnabled(false);
            PlatformUI.getWorkbench().getHelpSystem().setHelp(this,
                    INavigatorHelpContextIds.TEXT_CUT_ACTION);
        }

        public void runWithEvent(Event event) {
            if (activeTextControl != null && !activeTextControl.isDisposed()) {
                activeTextControl.cut();

                return;
            }
            if (cutAction != null) {
                cutAction.runWithEvent(event);
                return;
            }
        }

        /**
         * Update state.
         */
        public void updateEnabledState() {
            if (activeTextControl != null && !activeTextControl.isDisposed()) {
                setEnabled(activeTextControl.getSelectionCount() > 0);
                return;
            }
            if (cutAction != null) {
                setEnabled(cutAction.isEnabled());
                return;
            }
```

extensions.org.eclipse.ui.actions.TextActionHandler.java - src.org.eclipse.ui.internal.navigator.TextActionHandler.java

```
220             setEnabled(false);
221         }
222     }
223
224     private class CopyActionHandler extends Action {
225         protected CopyActionHandler() {
226             super(IDEWorkbenchMessages.Copy);
227             setId("TextCopyActionHandler");//$NON-NLS-1$
228             setEnabled(false);
229             PlatformUI.getWorkbench().getHelpSystem().setHelp(this,
230                     IIDEHelpContextIds.TEXT_COPY_ACTION);
231         }
232
233         public void runWithEvent(Event event) {
234             if (activeTextControl != null && !activeTextControl.isDisposed()) {
235                 activeTextControl.copy();
236                 updateActionsEnableState();
237                 return;
238             }
239             if (copyAction != null) {
240                 copyAction.runWithEvent(event);
241                 return;
242             }
243         }
244
245         /**
246          * Update the state.
247          */
248         public void updateEnabledState() {
249             if (activeTextControl != null && !activeTextControl.isDisposed()) {
250                 setEnabled(activeTextControl.getSelectionCount() > 0);
251                 return;
252             }
253             if (copyAction != null) {
254                 setEnabled(copyAction.isEnabled());
255                 return;
256             }
257             setEnabled(false);
258         }
259     }
260
261     private class PasteActionHandler extends Action {
262         protected PasteActionHandler() {
263             super(IDEWorkbenchMessages.Paste);
264             setId("TextPasteActionHandler");//$NON-NLS-1$
265             setEnabled(false);
266             PlatformUI.getWorkbench().getHelpSystem().setHelp(this,
267                     IIDEHelpContextIds.TEXT_PASTE_ACTION);
268         }
269
270         public void runWithEvent(Event event) {
271             if (activeTextControl != null && !activeTextControl.isDisposed()) {
272                 activeTextControl.paste();
273                 updateActionsEnableState();
274                 return;
275             }
```

```
203             setEnabled(false);
204         }
205     }
206
207     private class CopyActionHandler extends Action {
208         protected CopyActionHandler() {
209             super(CommonNavigatorMessages.Copy);
210             setId("TextCopyActionHandler");//$NON-NLS-1$
211             setEnabled(false);
212             PlatformUI.getWorkbench().getHelpSystem().setHelp(this,
213                     INavigatorHelpContextIds.TEXT_COPY_ACTION);
214         }
215
216         public void runWithEvent(Event event) {
217             if (activeTextControl != null && !activeTextControl.isDisposed()) {
218                 activeTextControl.copy();
219                 return;
220             }
221             if (copyAction != null) {
222                 copyAction.runWithEvent(event);
223                 return;
224             }
225         }
226
227         /**
228          * Update the state.
229          */
230         public void updateEnabledState() {
231             if (activeTextControl != null && !activeTextControl.isDisposed()) {
232                 setEnabled(activeTextControl.getSelectionCount() > 0);
233                 return;
234             }
235             if (copyAction != null) {
236                 setEnabled(copyAction.isEnabled());
237                 return;
238             }
239             setEnabled(false);
240         }
241     }
242
243     private class PasteActionHandler extends Action {
244         protected PasteActionHandler() {
245             super(CommonNavigatorMessages.Paste);
246             setId("TextPasteActionHandler");//$NON-NLS-1$
247             setEnabled(false);
248             PlatformUI.getWorkbench().getHelpSystem().setHelp(this,
249                     INavigatorHelpContextIds.TEXT_PASTE_ACTION);
250         }
251
252         public void runWithEvent(Event event) {
253             if (activeTextControl != null && !activeTextControl.isDisposed()) {
254                 activeTextControl.paste();
255                 return;
256             }
```

extensions.org.eclipse.ui.actions.TextActionHandler.java - src.org.eclipse.ui.internal.navigator.TextActionHandler.java

```
276             if (pasteAction != null) {
277                 pasteAction.runWithEvent(event);
278                 return;
279             }
280         }
281 
282         /**
283          * Update the state
284          */
285         public void updateEnabledState() {
286             if (activeTextControl != null && !activeTextControl.isDisposed()) {
287                 setEnabled(true);
288                 return;
289             }
290             if (pasteAction != null) {
291                 setEnabled(pasteAction.isEnabled());
292                 return;
293             }
294             setEnabled(false);
295         }
296     }
297 
298     private class SelectAllActionHandler extends Action {
299         protected SelectAllActionHandler() {
300             super(IDEWorkbenchMessages.TextAction_selectAll);
301             setId("TextSelectAllActionHandler");//$NON-NLS-1$
302             setEnabled(false);
303             PlatformUI.getWorkbench().getHelpSystem().setHelp(this,
304                     IIDEHelpContextIds.TEXT_SELECT_ALL_ACTION);
305         }
306 
307         public void runWithEvent(Event event) {
308             if (activeTextControl != null && !activeTextControl.isDisposed()) {
309                 activeTextControl.selectAll();
310                 updateActionsEnableState();
311                 return;
312             }
313             if (selectAllAction != null) {
314                 selectAllAction.runWithEvent(event);
315                 return;
316             }
317         }
318 
319         /**
320          * Update the state.
321          */
322         public void updateEnabledState() {
323             if (activeTextControl != null && !activeTextControl.isDisposed()) {
324                 setEnabled(true);
325                 return;
326             }
327             if (selectAllAction != null) {
328                 setEnabled(selectAllAction.isEnabled());
329                 return;
330             }
331             setEnabled(false);
```

```
257             if (pasteAction != null) {
258                 pasteAction.runWithEvent(event);
259                 return;
260             }
261         }
262 
263         /**
264          * Update the state
265          */
266         public void updateEnabledState() {
267             if (activeTextControl != null && !activeTextControl.isDisposed()) {
268                 setEnabled(true);
269                 return;
270             }
271             if (pasteAction != null) {
272                 setEnabled(pasteAction.isEnabled());
273                 return;
274             }
275             setEnabled(false);
276         }
277     }
278 
279     private class SelectAllActionHandler extends Action {
280         protected SelectAllActionHandler() {
281             super(CommonNavigatorMessages.TextAction_selectAll);
282             setId("TextSelectAllActionHandler");//$NON-NLS-1$
283             setEnabled(false);
284             PlatformUI.getWorkbench().getHelpSystem().setHelp(this,
285                     INavigatorHelpContextIds.TEXT_SELECT_ALL_ACTION);
286         }
287 
288         public void runWithEvent(Event event) {
289             if (activeTextControl != null && !activeTextControl.isDisposed()) {
290                 activeTextControl.selectAll();
291                 return;
292             }
293             if (selectAllAction != null) {
294                 selectAllAction.runWithEvent(event);
295                 return;
296             }
297         }
298 
299         /**
300          * Update the state.
301          */
302         public void updateEnabledState() {
303             if (activeTextControl != null && !activeTextControl.isDisposed()) {
304                 setEnabled(true);
305                 return;
306             }
307             if (selectAllAction != null) {
308                 setEnabled(selectAllAction.isEnabled());
309                 return;
310             }
311             setEnabled(false);
```

extensions.org.eclipse.ui.actions.TextActionHandler.java - src.org.eclipse.ui.internal.navigator.TextActionHandler.java

```
332         }
333     }
334 
335     /**
336      * Creates a <code>Text</code> control action handler
337      * for the global Cut, Copy, Paste, Delete, and Select All
338      * of the action bar.
339      *
340      * @param actionBar the action bar to register global
341      *    action handlers for Cut, Copy, Paste, Delete,
342      *    and Select All
343      */
344     public TextActionHandler(IActionBars actionBar) {
345         super();
346         actionBar.setGlobalActionHandler(ActionFactory.CUT.getId(),
347                 textCutAction);
348         actionBar.setGlobalActionHandler(ActionFactory.COPY.getId(),
349                 textCopyAction);
350         actionBar.setGlobalActionHandler(ActionFactory.PASTE.getId(),
351                 textPasteAction);
352         actionBar.setGlobalActionHandler(ActionFactory.SELECT_ALL.getId(),
353                 textSelectAllAction);
354         actionBar.setGlobalActionHandler(ActionFactory.DELETE.getId(),
355                 textDeleteAction);
356     }
357 
358     /**
359      * Add a <code>Text</code> control to the handler
360      * so that the Cut, Copy, Paste, Delete, and Select All
361      * actions are redirected to it when active.
362      *
363      * @param textControl the inline <code>Text</code> control
364      */
365     public void addText(Text textControl) {
366         if (textControl == null) {
367             return;
368         }
369 
370         activeTextControl = textControl;
371         textControl.addListener(SWT.Activate, textControlListener);
372         textControl.addListener(SWT.Deactivate, textControlListener);
373 
374         // We really want a selection listener but it is not supported so we
375         // use a key listener and a mouse listener to know when selection changes
376         // may have occured
377         textControl.addKeyListener(keyAdapter);
378         textControl.addMouseListener(mouseAdapter);
379 
380     }
381 
382     /**
383      * Dispose of this action handler
384      */
385     public void dispose() {
386         setCutAction(null);
387         setCopyAction(null);
```

```
312         }
313     }
314 
315     /**
316      * Creates a <code>Text</code> control action handler
317      * for the global Cut, Copy, Paste, Delete, and Select All
318      * of the action bar.
319      *
320      * @param actionBar the action bar to register global
321      *    action handlers for Cut, Copy, Paste, Delete,
322      *    and Select All
323      */
324     public TextActionHandler(IActionBars actionBar) {
325         super();
326         actionBar.setGlobalActionHandler(ActionFactory.CUT.getId(),
327                 textCutAction);
328         actionBar.setGlobalActionHandler(ActionFactory.COPY.getId(),
329                 textCopyAction);
330         actionBar.setGlobalActionHandler(ActionFactory.PASTE.getId(),
331                 textPasteAction);
332         actionBar.setGlobalActionHandler(ActionFactory.SELECT_ALL.getId(),
333                 textSelectAllAction);
334         actionBar.setGlobalActionHandler(ActionFactory.DELETE.getId(),
335                 textDeleteAction);
336     }
337 
338     /**
339      * Add a <code>Text</code> control to the handler
340      * so that the Cut, Copy, Paste, Delete, and Select All
341      * actions are redirected to it when active.
342      *
343      * @param textControl the inline <code>Text</code> control
344      */
345     public void addText(Text textControl) {
346         if (textControl == null) {
347             return;
348         }
349 
350         activeTextControl = textControl;
351         textControl.addListener(SWT.Activate, textControlListener);
352         textControl.addListener(SWT.Deactivate, textControlListener);
353 
354         // We really want a selection listener but it is not supported so we
355         // use a key listener and a mouse listener to know when selection changes
356         // may have occured
357         textControl.addKeyListener(keyAdapter);
358         textControl.addMouseListener(mouseAdapter);
359 
360     }
361 
362     /**
363      * Dispose of this action handler
364      */
365     public void dispose() {
366         setCutAction(null);
367         setCopyAction(null);
```

```
388         setPasteAction(null);
389         setSelectAllAction(null);
390         setDeleteAction(null);
391     }
392
393     /**
394      * Removes a <code>Text</code> control from the handler
395      * so that the Cut, Copy, Paste, Delete, and Select All
396      * actions are no longer redirected to it when active.
397      *
398      * @param textControl the inline <code>Text</code> control
399      */
400     public void removeText(Text textControl) {
401         if (textControl == null) {
402             return;
403         }
404
405         textControl.removeListener(SWT.Activate, textControlListener);
406         textControl.removeListener(SWT.Deactivate, textControlListener);
407
408         textControl.removeMouseListener(mouseAdapter);
409         textControl.removeKeyListener(keyAdapter);
410
411         activeTextControl = null;
412         updateActionsEnableState();
413     }
414
415     /**
416      * Set the default <code>IAction</code> handler for the Copy
417      * action. This <code>IAction</code> is run only if no active
418      * inline text control.
419      *
420      * @param action the <code>IAction</code> to run for the
421      *    Copy action, or <code>null</code> if not interested.
422      */
423     public void setCopyAction(IAction action) {
424         if (copyAction == action) {
425             return;
426         }
427
428         if (copyAction != null) {
429             copyAction.removePropertyChangeListener(copyActionListener);
430         }
431
432         copyAction = action;
433
434         if (copyAction != null) {
435             copyAction.addPropertyChangeListener(copyActionListener);
436         }
437
438         textCopyAction.updateEnabledState();
439     }
440
441     /**
442      * Set the default <code>IAction</code> handler for the Cut
443      * action. This <code>IAction</code> is run only if no active
```

```
368         setPasteAction(null);
369         setSelectAllAction(null);
370         setDeleteAction(null);
371     }
372
373     /**
374      * Removes a <code>Text</code> control from the handler
375      * so that the Cut, Copy, Paste, Delete, and Select All
376      * actions are no longer redirected to it when active.
377      *
378      * @param textControl the inline <code>Text</code> control
379      */
380     public void removeText(Text textControl) {
381         if (textControl == null) {
382             return;
383         }
384
385         textControl.removeListener(SWT.Activate, textControlListener);
386         textControl.removeListener(SWT.Deactivate, textControlListener);
387
388         textControl.removeMouseListener(mouseAdapter);
389         textControl.removeKeyListener(keyAdapter);
390
391         activeTextControl = null;
392         updateActionsEnableState();
393     }
394
395     /**
396      * Set the default <code>IAction</code> handler for the Copy
397      * action. This <code>IAction</code> is run only if no active
398      * inline text control.
399      *
400      * @param action the <code>IAction</code> to run for the
401      *    Copy action, or <code>null</code> if not interested.
402      */
403     public void setCopyAction(IAction action) {
404         if (copyAction == action) {
405             return;
406         }
407
408         if (copyAction != null) {
409             copyAction.removePropertyChangeListener(copyActionListener);
410         }
411
412         copyAction = action;
413
414         if (copyAction != null) {
415             copyAction.addPropertyChangeListener(copyActionListener);
416         }
417
418         textCopyAction.updateEnabledState();
419     }
420
421     /**
422      * Set the default <code>IAction</code> handler for the Cut
423      * action. This <code>IAction</code> is run only if no active
```

extensions.org.eclipse.ui.actions.TextActionHandler.java - src.org.eclipse.ui.internal.navigator.TextActionHandler.java

```
444     * inline text control.
445     *
446     * @param action the <code>IAction</code> to run for the
447     *    Cut action, or <code>null</code> if not interested.
448     */
449    public void setCutAction(IAction action) {
450        if (cutAction == action) {
451            return;
452        }
453
454        if (cutAction != null) {
455            cutAction.removePropertyChangeListener(cutActionListener);
456        }
457
458        cutAction = action;
459
460        if (cutAction != null) {
461            cutAction.addPropertyChangeListener(cutActionListener);
462        }
463
464        textCutAction.updateEnabledState();
465    }
466
467    /**
468     * Set the default <code>IAction</code> handler for the Paste
469     * action. This <code>IAction</code> is run only if no active
470     * inline text control.
471     *
472     * @param action the <code>IAction</code> to run for the
473     *    Paste action, or <code>null</code> if not interested.
474     */
475    public void setPasteAction(IAction action) {
476        if (pasteAction == action) {
477            return;
478        }
479
480        if (pasteAction != null) {
481            pasteAction.removePropertyChangeListener(pasteActionListener);
482        }
483
484        pasteAction = action;
485
486        if (pasteAction != null) {
487            pasteAction.addPropertyChangeListener(pasteActionListener);
488        }
489
490        textPasteAction.updateEnabledState();
491    }
492
493    /**
494     * Set the default <code>IAction</code> handler for the Select All
495     * action. This <code>IAction</code> is run only if no active
496     * inline text control.
497     *
498     * @param action the <code>IAction</code> to run for the
499     *    Select All action, or <code>null</code> if not interested.
```

```
424     * inline text control.
425     *
426     * @param action the <code>IAction</code> to run for the
427     *    Cut action, or <code>null</code> if not interested.
428     */
429    public void setCutAction(IAction action) {
430        if (cutAction == action) {
431            return;
432        }
433
434        if (cutAction != null) {
435            cutAction.removePropertyChangeListener(cutActionListener);
436        }
437
438        cutAction = action;
439
440        if (cutAction != null) {
441            cutAction.addPropertyChangeListener(cutActionListener);
442        }
443
444        textCutAction.updateEnabledState();
445    }
446
447    /**
448     * Set the default <code>IAction</code> handler for the Paste
449     * action. This <code>IAction</code> is run only if no active
450     * inline text control.
451     *
452     * @param action the <code>IAction</code> to run for the
453     *    Paste action, or <code>null</code> if not interested.
454     */
455    public void setPasteAction(IAction action) {
456        if (pasteAction == action) {
457            return;
458        }
459
460        if (pasteAction != null) {
461            pasteAction.removePropertyChangeListener(pasteActionListener);
462        }
463
464        pasteAction = action;
465
466        if (pasteAction != null) {
467            pasteAction.addPropertyChangeListener(pasteActionListener);
468        }
469
470        textPasteAction.updateEnabledState();
471    }
472
473    /**
474     * Set the default <code>IAction</code> handler for the Select All
475     * action. This <code>IAction</code> is run only if no active
476     * inline text control.
477     *
478     * @param action the <code>IAction</code> to run for the
479     *    Select All action, or <code>null</code> if not interested.
```

extensions.org.eclipse.ui.actions.TextActionHandler.java - src.org.eclipse.ui.internal.navigator.TextActionHandler.java

```java
     */
    public void setSelectAllAction(IAction action) {
        if (selectAllAction == action) {
            return;
        }

        if (selectAllAction != null) {
            selectAllAction
                    .removePropertyChangeListener(selectAllActionListener);
        }

        selectAllAction = action;

        if (selectAllAction != null) {
            selectAllAction.addPropertyChangeListener(selectAllActionListener);
        }

        textSelectAllAction.updateEnabledState();
    }

    /**
     * Set the default <code>IAction</code> handler for the Delete
     * action. This <code>IAction</code> is run only if no active
     * inline text control.
     *
     * @param action the <code>IAction</code> to run for the
     *    Delete action, or <code>null</code> if not interested.
     */
    public void setDeleteAction(IAction action) {
        if (deleteAction == action) {
            return;
        }

        if (deleteAction != null) {
            deleteAction.removePropertyChangeListener(deleteActionListener);
        }

        deleteAction = action;

        if (deleteAction != null) {
            deleteAction.addPropertyChangeListener(deleteActionListener);
        }

        textDeleteAction.updateEnabledState();
    }

    /**
     * Update the enable state of the Cut, Copy,
     * Paste, Delete, and Select All action handlers
     */
    private void updateActionsEnableState() {
        textCutAction.updateEnabledState();
        textCopyAction.updateEnabledState();
        textPasteAction.updateEnabledState();
        textSelectAllAction.updateEnabledState();
        textDeleteAction.updateEnabledState();
```

```java
     */
    public void setSelectAllAction(IAction action) {
        if (selectAllAction == action) {
            return;
        }

        if (selectAllAction != null) {
            selectAllAction
                    .removePropertyChangeListener(selectAllActionListener);
        }

        selectAllAction = action;

        if (selectAllAction != null) {
            selectAllAction.addPropertyChangeListener(selectAllActionListener);
        }

        textSelectAllAction.updateEnabledState();
    }

    /**
     * Set the default <code>IAction</code> handler for the Delete
     * action. This <code>IAction</code> is run only if no active
     * inline text control.
     *
     * @param action the <code>IAction</code> to run for the
     *    Delete action, or <code>null</code> if not interested.
     */
    public void setDeleteAction(IAction action) {
        if (deleteAction == action) {
            return;
        }

        if (deleteAction != null) {
            deleteAction.removePropertyChangeListener(deleteActionListener);
        }

        deleteAction = action;

        if (deleteAction != null) {
            deleteAction.addPropertyChangeListener(deleteActionListener);
        }

        textDeleteAction.updateEnabledState();
    }

    /**
     * Update the enable state of the Cut, Copy,
     * Paste, Delete, and Select All action handlers
     */
    private void updateActionsEnableState() {
        textCutAction.updateEnabledState();
        textCopyAction.updateEnabledState();
        textPasteAction.updateEnabledState();
        textSelectAllAction.updateEnabledState();
        textDeleteAction.updateEnabledState();
```

extensions.org.eclipse.ui.actions.TextActionHandler.java - src.org.eclipse.ui.internal.navigator.TextActionHandler.java

```
556     }                                                    536     }
557 }                                                        537 }
                                                             538
558                                                          539
```