MCT

Version: 8ad49b18d000dc81103a931c868d34e7f4eee91a

Parents: e394b7a59806cfa56f29911f0abd675228ad6e8e
c2ffdb072537537c3067385e38512d7feb18adb5

mct/fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/settings/PlotSettings.java

Chunk 1: (version 1/ annotation, method declaration)

```
<<<<<<< HEAD
        public <T> T getExtension(String key, Class<T> extensionClass) {
                return super.get(key, extensionClass);
        }

        @Override
        public <T> void setExtension(String key, T value) {
                super.set(key, value);
=======
        public void setFeedTypeSetting(String feedType) {
                this.set(PlotConstants.FEED_TYPE_SETTING, feedType);
        }

        @Override
        public String getFeedTypeSetting() {
                return this.get(PlotConstants.FEED_TYPE_SETTING, String.class);
>>>>>>> c2ffdb072537537c3067385e38512d7feb18adb5
        }
```

```
        @Override
        public <T> T getExtension(String key, Class<T> extensionClass) {
                return super.get(key, extensionClass);
        }

        @Override
        public <T> void setExtension(String key, T value) {
                super.set(key, value);
        }
```

mct/fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/view/PlotDataAssigner.java

Chunk 2: (concatenation/ import declaration)

```
<<<<<<< HEAD
import gov.nasa.arc.mct.components.FeedFilterProvider;
=======
import gov.nasa.arc.mct.components.FeedInfoProvider;
import gov.nasa.arc.mct.components.FeedInfoProvider.FeedInfo;
>>>>>>> c2ffdb072537537c3067385e38512d7feb18adb5
import gov.nasa.arc.mct.components.FeedProvider;
import gov.nasa.arc.mct.components.AbstractComponent;
import gov.nasa.arc.mct.components.FeedFilterProvider;
import gov.nasa.arc.mct.components.FeedInfoProvider;
import gov.nasa.arc.mct.components.FeedInfoProvider.FeedInfo;
import gov.nasa.arc.mct.components.FeedProvider;
```

mct/fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/view/PlotViewManifestation.java

Chunk 3: (concatenation/ import declaration)

```
<<<<<<< HEAD
import gov.nasa.arc.mct.components.FeedFilterProvider;
import gov.nasa.arc.mct.components.FeedFilterProvider.FeedFilter;
=======
import gov.nasa.arc.mct.components.FeedInfoProvider;
import gov.nasa.arc.mct.components.FeedInfoProvider.FeedInfo;
>>>>>>> c2ffdb072537537c3067385e38512d7feb18adb5
import gov.nasa.arc.mct.components.FeedProvider;
```

```
import gov.nasa.arc.mct.components.AbstractComponent;
import gov.nasa.arc.mct.components.FeedFilterProvider;
import gov.nasa.arc.mct.components.FeedFilterProvider.FeedFilter;
import gov.nasa.arc.mct.components.FeedInfoProvider;
import gov.nasa.arc.mct.components.FeedInfoProvider.FeedInfo;
import gov.nasa.arc.mct.components.FeedProvider;
```

mct/fastPlotViews/src/test/java/gov/nasa/arc/mct/fastplot/bridge/ShellPlotPackageImplementation.java

Chunk 4: (version 1/annotation, method declaration)

```
        @Override
<<<<<<< HEAD
        public <T> T getExtension(String key, Class<T> extensionClass) {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public <T> void setExtension(String key, T value) {
                // TODO Auto-generated method stub

=======
        public void setFeedTypeSetting(String feedType) {
                // TODO Auto-generated method stub

        }

        @Override
        public String getFeedTypeSetting() {
                // TODO Auto-generated method stub
                return null;
>>>>>>> c2ffdb072537537c3067385e38512d7feb18adb5
        }

        @Override
        public <T> T getExtension(String key, Class<T> extensionClass) {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public <T> void setExtension(String key, T value) {
                // TODO Auto-generated method stub
        }
```

Version: b6276614e9dc25e10c04b2ba589b12c1f7cc9496

Parents: 38ebddc99d9a3514c3114089ee1dc5887af014d4
0e810501d8cdcdf7847f72dfd8b3a915438f6291

mct/fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/scatter/ScatterPlot.java

Chunk 5: (concatenation/variable)

```
<<<<<<< HEAD
        private double initialNonTimeMin;
        private double initialNonTimeMax;
=======
        private        PlotLocalControlsManagerImpl        localControls        =        new
PlotLocalControlsManagerImpl();
        private PlotViewActionListener actionListener;

        private       Map<AxisVisibleOrientation,       Collection<AbstractAxisBoundManager>>
boundManagers =
                new HashMap<AxisVisibleOrientation, Collection<AbstractAxisBoundManager>>();
>>>>>>> 0e810501d8cdcdf7847f72dfd8b3a915438f6291
```

```
        private double initialNonTimeMin;
        private double initialNonTimeMax;

        private        PlotLocalControlsManagerImpl        localControls        =        new
PlotLocalControlsManagerImpl();
        private PlotViewActionListener actionListener;

        private       Map<AxisVisibleOrientation,       Collection<AbstractAxisBoundManager>>
boundManagers =
                new HashMap<AxisVisibleOrientation, Collection<AbstractAxisBoundManager>>();
```

Chunk 6: (version 1/ variable)

```
                timeAxis.setEnd(delegate.getMaxTime());
<<<<<<< HEAD
                initialNonTimeMin = delegate.getMinNonTime();
                initialNonTimeMax = delegate.getMaxNonTime();
=======
>>>>>>> 0e810501d8cdcdf7847f72dfd8b3a915438f6291
        }
```

```
                timeAxis.setEnd(delegate.getMaxTime());

                initialNonTimeMin = delegate.getMinNonTime();
                initialNonTimeMax = delegate.getMaxNonTime();
        }
```

Version: 754e20105acdfec3d0e73887ab83bb01b69bd24a

Parents: d0bc5a836a9d249932b6767addb2df55f187243d
87330eb67d3caa42a6d8669a067739d17cbeebfc

fastPlotViews/src/test/java/gov/nasa/arc/mct/fastplot/bridge/TestPlotView.java

Chunk 7: (version 2/ commentary, method invocation, variable)

```
        @Test
        public void testPlotMatchSettings(){

                PlotConfiguration plotSettings = new PlotSettings();
<<<<<<< HEAD
                PlotSettings      other        = new PlotSettings();
                // Copy time values to avoid intermittent failure
                other.setMinTime(plotSettings.getMinTime());
                other.setMaxTime(plotSettings.getMaxTime());
                PlotView                   basePlot                   =                   new
PlotView.Builder(PlotterPlot.class).plotSettings(other).build();
=======
                // Create a second set of settings with defaults...
                PlotConfiguration otherPlotSettings = new PlotSettings();
                // ...but explicitly make sure min/max times match
                //  (these  are  defined  relative  to  "now",  resulting  in  intermittent  test
failures otherwise)
                otherPlotSettings.setMinTime(plotSettings.getMinTime());
                otherPlotSettings.setMaxTime(plotSettings.getMaxTime());
                PlotView                   basePlot                   =                   new
PlotView.Builder(PlotterPlot.class).plotSettings(otherPlotSettings).build();
>>>>>>> 87330eb67d3caa42a6d8669a067739d17cbeebfc
```

```
        public void testPlotMatchSettings(){

                PlotConfiguration plotSettings = new PlotSettings();

                // Create a second set of settings with defaults...
                PlotConfiguration otherPlotSettings = new PlotSettings();
                // ...but explicitly make sure min/max times match
                //  (these  are  defined  relative  to  "now",  resulting  in  intermittent  test
failures otherwise)
                otherPlotSettings.setMinTime(plotSettings.getMinTime());
                otherPlotSettings.setMaxTime(plotSettings.getMaxTime());
                PlotView                   basePlot                   =                   new
PlotView.Builder(PlotterPlot.class).plotSettings(otherPlotSettings).build();


                Assert.assertTrue(basePlot.plotMatchesSetting(plotSettings));
```

fastPlotViews/src/test/java/gov/nasa/arc/mct/fastplot/view/TestPlotViewRole.java

Chunk 8: (new code/ annotation, method declaration, method invocation, method signature, variable)

```
         @Test (dataProvider="ingoresPredictiveTimeServiceTestCases")
         public void testIgnoresPredictiveTimeService(boolean p1, boolean p2, boolean p3, int t) {
        MockitoAnnotations.initMocks(this);
<<<<<<< HEAD
        SwingUtilities.invokeLater(new Runnable() {
         public void run() {
                 Mockito.when(feed1Component.getCapability(FeedProvider.class)).thenReturn(feed1);
                 Mockito.when(feed2Component.getCapability(FeedProvider.class)).thenReturn(feed2);
                 Mockito.when(feed3Component.getCapability(FeedProvider.class)).thenReturn(feed3);
                 Mockito.when(feed1Component.isLeaf()).thenReturn(true);
                 Mockito.when(feed2Component.isLeaf()).thenReturn(true);
                 Mockito.when(feed3Component.isLeaf()).thenReturn(true);

                 Mockito.when(feed1.getTimeService()).thenReturn(makeStaticTimeService(1));
                 Mockito.when(feed2.getTimeService()).thenReturn(makeStaticTimeService(2));
                 Mockito.when(feed3.getTimeService()).thenReturn(makeStaticTimeService(3));
                 Mockito.when(feed1.getSubscriptionId()).thenReturn("feed1");
                 Mockito.when(feed2.getSubscriptionId()).thenReturn("feed2");
                 Mockito.when(feed3.getSubscriptionId()).thenReturn("feed3");

                 TestersComponent component = new TestersComponent("x") {
                         @Override
                         public synchronized List<AbstractComponent> getComponents() {
                                 return Arrays.asList(feed1Component, feed2Component, feed3Component);
                         }
                 };

                 PlotViewManifestation plot;

                 Mockito.when(feed1.isPrediction()).thenReturn(false);
                 Mockito.when(feed2.isPrediction()).thenReturn(false);
                 Mockito.when(feed3.isPrediction()).thenReturn(false);
                 plot = new PlotViewManifestation(component, new
ViewInfo(PlotViewManifestation.class,"",ViewType.OBJECT));
                 Assert.assertEquals(plot.getCurrentMCTTime(), 1); // First non-predictive;

                 Mockito.when(feed1.isPrediction()).thenReturn(true);
                 Mockito.when(feed2.isPrediction()).thenReturn(false);
                 Mockito.when(feed3.isPrediction()).thenReturn(false);
                 plot = new PlotViewManifestation(component, new
ViewInfo(PlotViewManifestation.class,"",ViewType.OBJECT));
                 Assert.assertEquals(plot.getCurrentMCTTime(), 2); // First non-predictive;

                 Mockito.when(feed1.isPrediction()).thenReturn(true);
                 Mockito.when(feed2.isPrediction()).thenReturn(true);
                 Mockito.when(feed3.isPrediction()).thenReturn(true);
                 plot = new PlotViewManifestation(component, new
ViewInfo(PlotViewManifestation.class,"",ViewType.OBJECT));
                 Assert.assertEquals(plot.getCurrentMCTTime(), 1); // First non-predictive;
         }
        });
=======

        Mockito.when(feed1Component.getCapability(FeedProvider.class)).thenReturn(feed1);
        Mockito.when(feed2Component.getCapability(FeedProvider.class)).thenReturn(feed2);
        Mockito.when(feed3Component.getCapability(FeedProvider.class)).thenReturn(feed3);
        Mockito.when(feed1Component.isLeaf()).thenReturn(true);
        Mockito.when(feed2Component.isLeaf()).thenReturn(true);
        Mockito.when(feed3Component.isLeaf()).thenReturn(true);

        Mockito.when(feed1.getTimeService()).thenReturn(this.makeStaticTimeService(1));
        Mockito.when(feed2.getTimeService()).thenReturn(this.makeStaticTimeService(2));
        Mockito.when(feed3.getTimeService()).thenReturn(this.makeStaticTimeService(3));
        Mockito.when(feed1.getSubscriptionId()).thenReturn("feed1");
        Mockito.when(feed2.getSubscriptionId()).thenReturn("feed2");
        Mockito.when(feed3.getSubscriptionId()).thenReturn("feed3");

        TestersComponent component = new TestersComponent("x") {
                        @Override
                        public synchronized List<AbstractComponent> getComponents() {
                                return Arrays.asList(feed1Component, feed2Component, feed3Component);
                        }
                };

                PlotViewManifestation plot;

        Mockito.when(feed1.isPrediction()).thenReturn(p1);
        Mockito.when(feed2.isPrediction()).thenReturn(p2);
        Mockito.when(feed3.isPrediction()).thenReturn(p3);
                plot           =          new          PlotViewManifestation(component,           new
ViewInfo(PlotViewManifestation.class,"",ViewType.OBJECT));
                Assert.assertEquals(plot.getCurrentMCTTime(), t); // First non-predictive;

        }

        @DataProvider
        public Object[][] ingoresPredictiveTimeServiceTestCases() {
                return new Object[][]{
                                {true,true,true,1},
                                {true,false,false,2},
                                {false,false,false,1}
                };
>>>>>>> 87330eb67d3caa42a6d8669a067739d17cbeebfc
        }
```

```
        SwingUtilities.invokeLater(new Runnable() {
                public void run() {

                        TestersComponent component = new TestersComponent("x") {
                                @Override
                                public  synchronized  List<AbstractComponent>  getComponents()  {

                                        return   Arrays.asList(feed1Component,   feed2Component,
feed3Component);
                                }
                        };

                        PlotViewManifestation plot;

                        plot       =       new       PlotViewManifestation(component,       new
ViewInfo(PlotViewManifestation.class,"",ViewType.OBJECT));
                        Assert.assertEquals(plot.getCurrentMCTTime(),   t);   //   First   non-
predictive;
                }
        });

        }

        @DataProvider
        public Object[][] ingoresPredictiveTimeServiceTestCases() {
                return new Object[][]{
                                {true,true,true,1},
                                {true,false,false,2},
                                {false,false,false,1}
                };
        }

        private TimeService makeStaticTimeService(final long time) {
                return new TimeService() {
                        @Override
                        public long getCurrentTime() {
                                return time;
                        }
                };
        }
```

Version: 211691bd88352fb13781b9beaf1d35f502b0b17b

Parents: 0103b604fcd5926615fc3e25a47df4e7ee85eead
e9d24e022c514c8af14a3711975750a196568ba3
Merge base:
6be27daac9da64d0bd242db30fbf2d76d0fc4317

limits/src/main/java/gov/nasa/arc/mct/limits/LimitLineComponentProvider.java

Chunk 9: (concatenation/import declaration)

```
import gov.nasa.arc.mct.services.component.ComponentTypeInfo;
<<<<<<< HEAD
import gov.nasa.arc.mct.services.component.CreateWizardUI;
import gov.nasa.arc.mct.services.component.TypeInfo;
=======
>>>>>>> e9d24e022c514c8af14a3711975750a196568ba3
import gov.nasa.arc.mct.services.component.ViewInfo;
```

```
import gov.nasa.arc.mct.services.component.ComponentTypeInfo;
import gov.nasa.arc.mct.services.component.CreateWizardUI;
import gov.nasa.arc.mct.services.component.TypeInfo;
import gov.nasa.arc.mct.services.component.ViewInfo;
```

platform/src/main/java/gov/nasa/arc/mct/gui/housing/Inspector.java

Chunk 10: (concatenation/import declaration)

```
import gov.nasa.arc.mct.defaults.view.SwitcherView;
<<<<<<< HEAD
=======
import gov.nasa.arc.mct.gui.ActionContext;
import gov.nasa.arc.mct.gui.ContextAwareButton;
>>>>>>> e9d24e022c514c8af14a3711975750a196568ba3
import gov.nasa.arc.mct.gui.OptionBox;
```

```
import gov.nasa.arc.mct.defaults.view.SwitcherView;
import gov.nasa.arc.mct.gui.ActionContext;
import gov.nasa.arc.mct.gui.ContextAwareButton;
import gov.nasa.arc.mct.gui.OptionBox;
```

Version: 48388f4b64837ce9c7b4f634a50a41eeb87e5176

Parents:

e6938dc7cd78c86dbd688618ae8dff8293f23c90
79deb0c56b8ea72548c2b8b086b961edd7a08b7f

Merge base:

8964af3b15dd0f75985eed1aaf902b2bf058f714

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/view/PlotViewManifestation.java

Chunk 11: (combination/variable)

```
<<<<<<< HEAD
        private    SwingWorker<Map<String,    List<Map<String,    String>>>,    Map<String,
List<Map<String, String>>>> currentDataRequest;
        private    SwingWorker<Map<String,    List<Map<String,    String>>>,    Map<String,
List<Map<String, String>>>> currentPredictionRequest;

=======
        SwingWorker<Map<String,  List<Map<String,  String>>>,  Map<String,  List<Map<String,
String>>>> currentDataRequest;
        SwingWorker<Map<String,  List<Map<String,  String>>>,  Map<String,  List<Map<String,
String>>>> currentPredictionRequest;

        private List<Runnable> feedCallbacks = new ArrayList<Runnable>();
>>>>>>> 79deb0c56b8ea72548c2b8b086b961edd7a08b7f
```

```
       private    SwingWorker<Map<String,    List<Map<String,    String>>>,    Map<String,
List<Map<String, String>>>> currentDataRequest;
        private    SwingWorker<Map<String,    List<Map<String,    String>>>,    Map<String,
List<Map<String, String>>>> currentPredictionRequest;

        private List<Runnable> feedCallbacks = new ArrayList<Runnable>();
```

Chunk 12: (combination/ for statement, method invocation)

```
        @Override
        public void updateFromFeed(Map<String, List<Map<String, String>>> data) {
<<<<<<< HEAD
                plotDataFeedUpdateHandler.updateFromFeed(data, false);
=======
                plotDataFedUpdateHandler.updateFromFeed(data, false);
                for (Runnable r : feedCallbacks) {
                        SwingUtilities.invokeLater(r);
                }
>>>>>>> 79deb0c56b8ea72548c2b8b086b961edd7a08b7f
        }
```

```
@Override
        public void updateFromFeed(Map<String, List<Map<String, String>>> data) {
                plotDataFeedUpdateHandler.updateFromFeed(data, false);
                for (Runnable r : feedCallbacks) {
                        SwingUtilities.invokeLater(r);
                }
        }
```

Version: 8b8fc11cf51665896d3919d334b16e79c77bdbfb

Parents:

b13d06e3b49f5a7ae9f7b458cc8b0282131009b9
8ac8746572042287f030987cfd209354e5a62b1a

Merge base:

6a7ed82accca6fcb85518c77c9ed20f47643e4aa

multiColumnTables/src/main/java/org/acme/example/view/MultiColView.java

Chunk 13: (new code/commentary, method invocation, variable)

```
               super(ac,vi);
               JPanel viewPanel = new JPanel(new BorderLayout());

<<<<<<< HEAD
               JPanel view = new JPanel();
               view.setLayout(new BoxLayout(view, BoxLayout.Y_AXIS));

               // Add the content for this view manifestation.
               AbstractComponent component = getManifestedComponent();
=======
>>>>>>> 8ac8746572042287f030987cfd209354e5a62b1a
               settings = new ViewSettings();

               AbstractComponent component = getManifestedComponent();
               List<AbstractComponent> childrenList = component.getComponents();
               //If no children, we display the selectedComponent.
```

```
               super(ac,vi);
               JPanel viewPanel = new JPanel(new BorderLayout());

               ViewSettings settings = new ViewSettings();

               AbstractComponent component = getManifestedComponent();
               List<AbstractComponent> childrenList = component.getComponents();
               //If no children, we display the selectedComponent.
```

Version: e6001b8b12cc0b8e47a736dc38071f65ee759127

Parents: e17efe25ca4fa278ccdcdf1791fa37b7a606af52
069552e9838f93b3b6ab3c9686e5719b93d0809a

Merge base:
8f23fb01223ac9108fb09397238a654b3f2d35f6

multiColumnTables/src/main/java/org/acme/example/view/MultiColTable.java

Chunk 14: (new code/method invocation)

```
        public    MultiColTable(AbstractComponent    component,    ViewSettings    settings,
MultiColView multiColView) {
                super(new GridLayout(1,0));
                this.multiColView = multiColView;
                model = new MultiColTableModel(component, this, settings);
                table = new JTable(model);
                table.setAutoCreateRowSorter(true);
                table.setPreferredScrollableViewportSize(new Dimension(400,750)); //+++ TODO
                table.setFillsViewportHeight(true);
                DynamicValueCellRender dynamicValueCellRender = new DynamicValueCellRender();
<<<<<<< HEAD

        table.getColumnModel().getColumn(ColumnType.VALUE.ordinal()).setCellRenderer(dynamic
ValueCellRender);

        table.getColumnModel().getColumn(ColumnType.RAW.ordinal()).setCellRenderer(dynamicVa
lueCellRender);
                TimeCellRender timeCellRender = new TimeCellRender();

        table.getColumnModel().getColumn(ColumnType.ERT.ordinal()).setCellRenderer(timeCellR
ender);

        table.getColumnModel().getColumn(ColumnType.SCLK.ordinal()).setCellRenderer(timeCell
Render);

        table.getColumnModel().getColumn(ColumnType.SCET.ordinal()).setCellRenderer(timeCell
Render);
=======

        table.getColumnModel().getColumn(settings.getIndexForColumn(ColumnType.VALUE)).setCe
llRenderer(dynamicValueCellRender);

        table.getColumnModel().getColumn(settings.getIndexForColumn(ColumnType.RAW)).setCell
Renderer(dynamicValueCellRender);
>>>>>>> 069552e9838f93b3b6ab3c9686e5719b93d0809a
                //attempt to hide column header borders:
                for(int colIndex=0; colIndex<model.getColumnCount(); colIndex++) {
                        setColumnHeaderBorderState(colIndex, new BorderState("NONE"));
                        setColumnHeaderBorderColor(colIndex, Color.black);
                }
                scroll = new JScrollPane(table);
                add(scroll);
        }
```

```
public   MultiColTable(AbstractComponent   component,   ViewSettings   settings,   MultiColView
multiColView) {
                super(new GridLayout(1,0));
                this.multiColView = multiColView;
                model = new MultiColTableModel(component, this, settings);
                table = new JTable(model);
                table.setAutoCreateRowSorter(true);
                table.setPreferredScrollableViewportSize(new Dimension(400,750)); //+++ TODO
                table.setFillsViewportHeight(true);
                DynamicValueCellRender dynamicValueCellRender = new DynamicValueCellRender();

        table.getColumnModel().getColumn(settings.getIndexForColumn(ColumnType.VALUE)).setCe
llRenderer(dynamicValueCellRender);

        table.getColumnModel().getColumn(settings.getIndexForColumn(ColumnType.RAW)).setCell
Renderer(dynamicValueCellRender);
                TimeCellRender timeCellRender = new TimeCellRender();

        table.getColumnModel().getColumn(settings.getIndexForColumn(ColumnType.ERT)).setCell
Renderer(timeCellRender);

        table.getColumnModel().getColumn(settings.getIndexForColumn(ColumnType.SCLK)).setCel
lRenderer(timeCellRender);

        table.getColumnModel().getColumn(settings.getIndexForColumn(ColumnType.SCET)).setCel
lRenderer(timeCellRender);
                //attempt to hide column header borders:
                for(int colIndex=0; colIndex<model.getColumnCount(); colIndex++) {
                        setColumnHeaderBorderState(colIndex, new BorderState("NONE"));
                        setColumnHeaderBorderColor(colIndex, Color.black);
                }
                scroll = new JScrollPane(table);
                add(scroll);
        }
```

multiColumnTables/src/main/java/org/acme/example/view/MultiColTableModel.java

Chunk 15: (new code/method invocation)

```
                        for (Integer row : locations) {
<<<<<<< HEAD
                                fireTableCellUpdated(row, ColumnType.VALUE.ordinal());
                                fireTableCellUpdated(row, ColumnType.RAW.ordinal());
                                fireTableCellUpdated(row, ColumnType.ERT.ordinal());
                                fireTableCellUpdated(row, ColumnType.SCET.ordinal());
                                fireTableCellUpdated(row, ColumnType.SCLK.ordinal());
=======
                                fireTableCellUpdated(row,
settings.getIndexForColumn(ColumnType.VALUE));
                                fireTableCellUpdated(row,
settings.getIndexForColumn(ColumnType.RAW));
>>>>>>> 069552e9838f93b3b6ab3c9686e5719b93d0809a
                        }
                }
        }
```

```
                        for (Integer row : locations) {
                                fireTableCellUpdated(row,
settings.getIndexForColumn(ColumnType.VALUE));
                                fireTableCellUpdated(row,
settings.getIndexForColumn(ColumnType.RAW));
                                fireTableCellUpdated(row,
settings.getIndexForColumn(ColumnType.ERT));
                                fireTableCellUpdated(row,
settings.getIndexForColumn(ColumnType.SCET));
                                fireTableCellUpdated(row,
settings.getIndexForColumn(ColumnType.SCLK));
                        }
                }
        }
```

Version: 4420f541e530ad199aee3912e74c3ee2a282e138

Parents: 49b5db09a80c286abf7341202cd7ceccae0bc004
        42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
Merge base:
        e13034bf846b8aeb9a0f5a2107a5f51a048b25c3

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/bridge/PanAndZoomManager.java

Chunk 16: (new code/import declaration, class declaration)

```
<<<<<<< HEAD
package gov.nasa.arc.mct.fastplot.bridge;

import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.AxisOrientationSetting;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.PanDirection;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.PlotDisplayState;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.ZoomDirection;
import gov.nasa.arc.mct.fastplot.view.Axis;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import plotter.xy.XYAxis;

public class PanAndZoomManager {

        private         final         static         Logger         logger         =
LoggerFactory.getLogger(PanAndZoomManager.class);

        private PlotterPlot plot;

        private boolean inZoomMode;

        private boolean inPanMode;

        public PanAndZoomManager(PlotterPlot quinnCurtisPlot) {
                plot = quinnCurtisPlot;
        }

        public void enteredPanMode() {
                logger.debug("Entering pan mode");
                inPanMode = true;
                // turn off the limit manager.
                plot.limitManager.setEnabled(false);
                plot.setPlotDisplayState(PlotDisplayState.USER_INTERACTION);
        }

        public void exitedPanMode() {
                inPanMode = false;
                logger.debug("Exited pan mode");
        }

        public void enteredZoomMode() {
                logger.debug("Entered zoom mode");
                inZoomMode = true;
                // turn off the limit manager.
                plot.limitManager.setEnabled(false);
                plot.setPlotDisplayState(PlotDisplayState.USER_INTERACTION);
        }
        public void exitedZoomMode() {
                inZoomMode = false;
                logger.debug("Exited zoom mode");
        }


        public boolean isInZoomMode() {
                return inZoomMode;
        }


        public boolean isInPanMode() {
                return inPanMode;
        }


        public void panAction(PanDirection panningAction) {
                XYAxis xAxis = plot.plotView.getXAxis();
                XYAxis yAxis = plot.plotView.getYAxis();
                boolean timeChanged = false;
                if                  (plot.getAxisOrientationSetting()                    ==
AxisOrientationSetting.X_AXIS_AS_TIME) {
                        double nonTimeScalePanAmount = yAxis.getEnd() - yAxis.getStart();
                        double timeScalePanAmount = xAxis.getEnd() - xAxis.getStart();

                        timeScalePanAmount        =        (timeScalePanAmount/100)        *
PlotConstants.PANNING_TIME_AXIS_PERCENTAGE;
                        nonTimeScalePanAmount=          (nonTimeScalePanAmount/100)          *
PlotConstants.PANNING_TIME_AXIS_PERCENTAGE;

                        if (panningAction == PanDirection.PAN_HIGHER_Y_AXIS) {
                                yAxis.shift(nonTimeScalePanAmount);
                                pinNonTime();
                        } else if (panningAction == PanDirection.PAN_LOWER_Y_AXIS) {
                                yAxis.shift(-nonTimeScalePanAmount);
                                pinNonTime();
                        } else if (panningAction == PanDirection.PAN_LOWER_X_AXIS) {
                                xAxis.shift(-timeScalePanAmount);
                                pinTime();
                                plot.notifyObserversTimeChange();
                                timeChanged = true;
                        } else if (panningAction == PanDirection.PAN_HIGHER_X_AXIS) {
                                xAxis.shift(timeScalePanAmount);
                                pinTime();
                                plot.notifyObserversTimeChange();
                                timeChanged = true;
                        }
                } else {

                        double nonTimeScalePanAmount = xAxis.getEnd() - xAxis.getStart();
                        double timeScalePanAmount = yAxis.getEnd() - yAxis.getStart();

                        timeScalePanAmount        =        (timeScalePanAmount/100)        *
PlotConstants.PANNING_TIME_AXIS_PERCENTAGE;
                        nonTimeScalePanAmount=          (nonTimeScalePanAmount/100)          *
PlotConstants.PANNING_TIME_AXIS_PERCENTAGE;

                        if (panningAction == PanDirection.PAN_HIGHER_Y_AXIS) {
                                yAxis.shift(timeScalePanAmount);
                                pinTime();
                                plot.notifyObserversTimeChange();
                                timeChanged = true;
                        } else if (panningAction == PanDirection.PAN_LOWER_Y_AXIS) {
                                yAxis.shift(-timeScalePanAmount);
                                pinTime();
                                plot.notifyObserversTimeChange();
                                timeChanged = true;
                        } else if (panningAction == PanDirection.PAN_LOWER_X_AXIS) {
                                xAxis.shift(-nonTimeScalePanAmount);
                                pinNonTime();
                        } else if (panningAction == PanDirection.PAN_HIGHER_X_AXIS) {
                                xAxis.shift(nonTimeScalePanAmount);
                                pinNonTime();
                        }
                }
                plot.plotAbstraction.updateResetButtons();
                plot.refreshDisplay();
                if(timeChanged) {
                        plot.clearAllDataFromPlot();
                        plot.plotAbstraction.requestPlotData(plot.getCurrentTimeAxisMin(),
plot.getCurrentTimeAxisMax());
                }
        }


        private void pinTime() {
                plot.plotAbstraction.getTimeAxisUserPin().setPinned(true);
        }


        private void pinNonTime() {
                plot.getNonTimeAxisUserPin().setPinned(true);
        }


        private void markTimeZoomed() {
                Axis axis = plot.plotAbstraction.getTimeAxis();
                pinTime();
                axis.setZoomed(true);
        }

        private void markNonTimeZoomed() {
                Axis axis = plot.getNonTimeAxis();
                pinNonTime();
                axis.setZoomed(true);
        }

        public void zoomAction(ZoomDirection zoomAction) {
                XYAxis xAxis = plot.plotView.getXAxis();
                XYAxis yAxis = plot.plotView.getYAxis();
                boolean timeChanged = false;
                if                    (plot.getAxisOrientationSetting()                    ==
AxisOrientationSetting.X_AXIS_AS_TIME) {
                        double nonTimeScaleZoomAmount = yAxis.getEnd() - yAxis.getStart();
                        double timeScaleZoomAmount = xAxis.getEnd() - xAxis.getStart();

                        timeScaleZoomAmount        =        (timeScaleZoomAmount/100)        *
PlotConstants.ZOOMING_TIME_AXIS_PERCENTAGE;
                        nonTimeScaleZoomAmount=         (nonTimeScaleZoomAmount/100)         *
PlotConstants.ZOOMING_TIME_AXIS_PERCENTAGE;

                        if (zoomAction == ZoomDirection.ZOOM_IN_HIGH_Y_AXIS) {
                                yAxis.setEnd(yAxis.getEnd() - nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                      } else if (zoomAction == ZoomDirection.ZOOM_OUT_HIGH_Y_AXIS) {
                              yAxis.setEnd(yAxis.getEnd() + nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                      } else if (zoomAction == ZoomDirection.ZOOM_IN_CENTER_Y_AXIS) {
                              yAxis.setStart(yAxis.getStart() + nonTimeScaleZoomAmount);
                              yAxis.setEnd(yAxis.getEnd() - nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                      } else if (zoomAction == ZoomDirection.ZOOM_OUT_CENTER_Y_AXIS) {
                              yAxis.setStart(yAxis.getStart() - nonTimeScaleZoomAmount);
                              yAxis.setEnd(yAxis.getEnd() + nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                      } else if (zoomAction == ZoomDirection.ZOOM_IN_LOW_Y_AXIS) {
                              yAxis.setStart(yAxis.getStart() + nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                      } else if (zoomAction == ZoomDirection.ZOOM_OUT_LOW_Y_AXIS) {
                              yAxis.setStart(yAxis.getStart() - nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                      } else if (zoomAction == ZoomDirection.ZOOM_IN_LEFT_X_AXIS) {
                              xAxis.setStart(xAxis.getStart() + timeScaleZoomAmount);
                                markTimeZoomed();
                               plot.notifyObserversTimeChange();
                               timeChanged = true;
                      } else if (zoomAction == ZoomDirection.ZOOM_OUT_LEFT_X_AXIS) {
                                     xAxis.setStart(xAxis.getStart()                       -
timeScaleZoomAmount);
                                markTimeZoomed();
                                plot.notifyObserversTimeChange();
                                timeChanged = true;
                      } else if (zoomAction == ZoomDirection.ZOOM_IN_CENTER_X_AXIS) {
                              xAxis.setStart(xAxis.getStart() + timeScaleZoomAmount);
                              xAxis.setEnd(xAxis.getEnd() - timeScaleZoomAmount);
                                markTimeZoomed();
                                plot.notifyObserversTimeChange();
                                timeChanged = true;
                      } else if (zoomAction == ZoomDirection.ZOOM_OUT_CENTER_X_AXIS) {
                              xAxis.setStart(xAxis.getStart() - timeScaleZoomAmount);
                              xAxis.setEnd(xAxis.getEnd() + timeScaleZoomAmount);
                                markTimeZoomed();
                                plot.notifyObserversTimeChange();
                                timeChanged = true;
                      } else if (zoomAction == ZoomDirection.ZOOM_IN_RIGHT_X_AXIS) {
                              xAxis.setEnd(xAxis.getEnd() - timeScaleZoomAmount);
                                markTimeZoomed();
                                plot.notifyObserversTimeChange();
                                timeChanged = true;
                      } else if (zoomAction == ZoomDirection.ZOOM_OUT_RIGHT_X_AXIS) {
                              xAxis.setEnd(xAxis.getEnd() + timeScaleZoomAmount);
                                markTimeZoomed();
                                plot.notifyObserversTimeChange();
                                timeChanged = true;
                      }

               } else {
                      double nonTimeScaleZoomAmount = xAxis.getEnd() - xAxis.getStart();
                      double timeScaleZoomAmount = yAxis.getEnd() - yAxis.getStart();

                      timeScaleZoomAmount        =        (timeScaleZoomAmount/100)        *
PlotConstants.ZOOMING_TIME_AXIS_PERCENTAGE;
                      nonTimeScaleZoomAmount      =      (nonTimeScaleZoomAmount/100)      *
PlotConstants.ZOOMING_TIME_AXIS_PERCENTAGE;
```

```
                       if (zoomAction == ZoomDirection.ZOOM_IN_HIGH_Y_AXIS) {
                                      yAxis.setEnd(yAxis.getEnd() - timeScaleZoomAmount);
                                 markTimeZoomed();
                                 plot.notifyObserversTimeChange();
                                 timeChanged = true;
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_HIGH_Y_AXIS) {
                               yAxis.setEnd(yAxis.getEnd() + timeScaleZoomAmount);
                                 markTimeZoomed();
                                plot.notifyObserversTimeChange();
                                timeChanged = true;
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_CENTER_Y_AXIS) {
                                      yAxis.setStart(yAxis.getStart()                       +
timeScaleZoomAmount);
                                      yAxis.setEnd(yAxis.getEnd() - timeScaleZoomAmount);
                                 markTimeZoomed();
                                 plot.notifyObserversTimeChange();
                                 timeChanged = true;
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_CENTER_Y_AXIS) {
                               yAxis.setStart(yAxis.getStart() - timeScaleZoomAmount);
                               yAxis.setEnd(yAxis.getEnd() + timeScaleZoomAmount);
                                 markTimeZoomed();
                                 plot.notifyObserversTimeChange();
                                 timeChanged = true;
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_LOW_Y_AXIS) {
                               yAxis.setStart(yAxis.getStart() + timeScaleZoomAmount);
                                 markTimeZoomed();
                                plot.notifyObserversTimeChange();
                                timeChanged = true;
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_LOW_Y_AXIS) {
                               yAxis.setStart(yAxis.getStart() - timeScaleZoomAmount);
                                 markTimeZoomed();
                                 plot.notifyObserversTimeChange();
                                 timeChanged = true;
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_LEFT_X_AXIS) {
                               xAxis.setStart(xAxis.getStart() + nonTimeScaleZoomAmount);
                                 markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_LEFT_X_AXIS) {
                                      xAxis.setStart(xAxis.getStart()                       -
nonTimeScaleZoomAmount);
                                 markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_CENTER_X_AXIS) {
                                      xAxis.setStart(xAxis.getStart()                       +
nonTimeScaleZoomAmount);
                                      xAxis.setEnd(xAxis.getEnd() - nonTimeScaleZoomAmount);
                                 markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_CENTER_X_AXIS) {
                               xAxis.setStart(xAxis.getStart() - nonTimeScaleZoomAmount);
                               xAxis.setEnd(xAxis.getEnd() + nonTimeScaleZoomAmount);
                                 markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_RIGHT_X_AXIS) {
                                      xAxis.setEnd(xAxis.getEnd() - nonTimeScaleZoomAmount);
                                 markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_RIGHT_X_AXIS) {
                               xAxis.setEnd(xAxis.getEnd() + nonTimeScaleZoomAmount);
                                 markNonTimeZoomed();
                       }
               }
               plot.plotAbstraction.updateResetButtons();
               plot.refreshDisplay();
               if(timeChanged) {
                       plot.plotDataManager.resizeAndReloadPlotBuffer();
               }
```

```
        }
}
=======
package gov.nasa.arc.mct.fastplot.bridge;

import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.AxisOrientationSetting;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.PanDirection;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.PlotDisplayState;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.ZoomDirection;
import gov.nasa.arc.mct.fastplot.view.Axis;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import plotter.xy.XYAxis;

public class PanAndZoomManager {

        private final static Logger logger =
LoggerFactory.getLogger(PanAndZoomManager.class);

        private PlotterPlot plot;

        private boolean inZoomMode;

        private boolean inPanMode;

        public PanAndZoomManager(PlotterPlot quinnCurtisPlot) {
                plot = quinnCurtisPlot;
        }

        public void enteredPanMode() {
                logger.debug("Entering pan mode");
                inPanMode = true;
                plot.setPlotDisplayState(PlotDisplayState.USER_INTERACTION);
        }

        public void exitedPanMode() {
                inPanMode = false;
                logger.debug("Exited pan mode");
        }

        public void enteredZoomMode() {
                logger.debug("Entered zoom mode");
                inZoomMode = true;
                plot.setPlotDisplayState(PlotDisplayState.USER_INTERACTION);
        }

        public void exitedZoomMode() {
                inZoomMode = false;
                logger.debug("Exited zoom mode");
        }


        public boolean isInZoomMode() {
                return inZoomMode;
        }


        public boolean isInPanMode() {
                return inPanMode;
        }
        public void panAction(PanDirection panningAction) {
                XYAxis xAxis = plot.plotView.getXAxis();
                XYAxis yAxis = plot.plotView.getYAxis();
                if (plot.axisOrientation == AxisOrientationSetting.X_AXIS_AS_TIME) {
                        double nonTimeScalePanAmount = yAxis.getEnd() - yAxis.getStart();
                        double timeScalePanAmount = xAxis.getEnd() - xAxis.getStart();

                        timeScalePanAmount        =        (timeScalePanAmount/100)        *
PlotConstants.PANNING_TIME_AXIS_PERCENTAGE;
                        nonTimeScalePanAmount=         (nonTimeScalePanAmount/100)         *
PlotConstants.PANNING_TIME_AXIS_PERCENTAGE;

                        if (panningAction == PanDirection.PAN_HIGHER_Y_AXIS) {
                                yAxis.shift(nonTimeScalePanAmount);
                                pinNonTime();
                        } else if (panningAction == PanDirection.PAN_LOWER_Y_AXIS) {
                                yAxis.shift(-nonTimeScalePanAmount);
                                pinNonTime();
                        } else if (panningAction == PanDirection.PAN_LOWER_X_AXIS) {
                                xAxis.shift(-timeScalePanAmount);
                                pinTime();
                                plot.notifyObserversTimeChange();
                        } else if (panningAction == PanDirection.PAN_HIGHER_X_AXIS) {
                                xAxis.shift(timeScalePanAmount);
                                pinTime();
                                plot.notifyObserversTimeChange();
                        }
                } else {

                        double nonTimeScalePanAmount = xAxis.getEnd() - xAxis.getStart();
                        double timeScalePanAmount = yAxis.getEnd() - yAxis.getStart();

                        timeScalePanAmount        =        (timeScalePanAmount/100)        *
PlotConstants.PANNING_TIME_AXIS_PERCENTAGE;
                        nonTimeScalePanAmount=         (nonTimeScalePanAmount/100)         *
PlotConstants.PANNING_TIME_AXIS_PERCENTAGE;

                        if (panningAction == PanDirection.PAN_HIGHER_Y_AXIS) {
                                yAxis.shift(timeScalePanAmount);
                                pinTime();
                                plot.notifyObserversTimeChange();
                        } else if (panningAction == PanDirection.PAN_LOWER_Y_AXIS) {
                                yAxis.shift(-timeScalePanAmount);
                                pinTime();
                                plot.notifyObserversTimeChange();
                        } else if (panningAction == PanDirection.PAN_LOWER_X_AXIS) {
                                xAxis.shift(-nonTimeScalePanAmount);
                                pinNonTime();
                        } else if (panningAction == PanDirection.PAN_HIGHER_X_AXIS) {
                                xAxis.shift(nonTimeScalePanAmount);
                                pinNonTime();
                        }
                }
                plot.plotAbstraction.updateResetButtons();
                plot.refreshDisplay();
                //Always request data refresh
                plot.clearAllDataFromPlot();
                plot.limitManager.setModeUntranslated(false);
                plot.plotAbstraction.requestPlotData(plot.getCurrentTimeAxisMin(),
plot.getCurrentTimeAxisMax());
```

```
        }


        private void pinTime() {
                plot.plotAbstraction.getTimeAxisUserPin().setPinned(true);
        }


        private void pinNonTime() {
                plot.getNonTimeAxisUserPin().setPinned(true);
                if (plot.limitManager.isUntranslated()) {
                        plot.limitManager.setModeUntranslated(false);
                }
        }


        private void markTimeZoomed() {
                Axis axis = plot.plotAbstraction.getTimeAxis();
                pinTime();
                axis.setZoomed(true);
                if (plot.limitManager.isUntranslated()) {
                        plot.limitManager.setModeUntranslated(false);
                }
        }

        private void markNonTimeZoomed() {
                Axis axis = plot.getNonTimeAxis();
                pinNonTime();
                axis.setZoomed(true);
                if (plot.limitManager.isUntranslated()) {
                        plot.limitManager.setModeUntranslated(false);
                }
        }

        public void zoomAction(ZoomDirection zoomAction) {
                XYAxis xAxis = plot.plotView.getXAxis();
                XYAxis yAxis = plot.plotView.getYAxis();
                if (plot.axisOrientation == AxisOrientationSetting.X_AXIS_AS_TIME) {
                        double nonTimeScaleZoomAmount = yAxis.getEnd() - yAxis.getStart();
                        double timeScaleZoomAmount = xAxis.getEnd() - xAxis.getStart();

                        timeScaleZoomAmount        =        (timeScaleZoomAmount/100)        *
PlotConstants.ZOOMING_TIME_AXIS_PERCENTAGE;
                        nonTimeScaleZoomAmount=         (nonTimeScaleZoomAmount/100)         *
PlotConstants.ZOOMING_TIME_AXIS_PERCENTAGE;

                        if (zoomAction == ZoomDirection.ZOOM_IN_HIGH_Y_AXIS) {
                                yAxis.setEnd(yAxis.getEnd() - nonTimeScaleZoomAmount);
                                  markNonTimeZoomed();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_HIGH_Y_AXIS) {
                                yAxis.setEnd(yAxis.getEnd() + nonTimeScaleZoomAmount);
                                  markNonTimeZoomed();
                        } else if (zoomAction == ZoomDirection.ZOOM_IN_CENTER_Y_AXIS) {
                                yAxis.setStart(yAxis.getStart() + nonTimeScaleZoomAmount);
                                yAxis.setEnd(yAxis.getEnd() - nonTimeScaleZoomAmount);
                                  markNonTimeZoomed();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_CENTER_Y_AXIS) {
                                yAxis.setStart(yAxis.getStart() - nonTimeScaleZoomAmount);
                                yAxis.setEnd(yAxis.getEnd() + nonTimeScaleZoomAmount);
                                  markNonTimeZoomed();
                        } else if (zoomAction == ZoomDirection.ZOOM_IN_LOW_Y_AXIS) {
                                yAxis.setStart(yAxis.getStart() + nonTimeScaleZoomAmount);
                                  markNonTimeZoomed();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_LOW_Y_AXIS) {
                                yAxis.setStart(yAxis.getStart() - nonTimeScaleZoomAmount);
                                  markNonTimeZoomed();
                        } else if (zoomAction == ZoomDirection.ZOOM_IN_LEFT_X_AXIS) {
                                xAxis.setStart(xAxis.getStart() + timeScaleZoomAmount);
                                  markTimeZoomed();
                                 plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_LEFT_X_AXIS) {
                                       xAxis.setStart(xAxis.getStart()                      -
timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_IN_CENTER_X_AXIS) {
                                xAxis.setStart(xAxis.getStart() + timeScaleZoomAmount);
                                xAxis.setEnd(xAxis.getEnd() - timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_CENTER_X_AXIS) {
                                xAxis.setStart(xAxis.getStart() - timeScaleZoomAmount);
                                xAxis.setEnd(xAxis.getEnd() + timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_IN_RIGHT_X_AXIS) {
                                xAxis.setEnd(xAxis.getEnd() - timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_RIGHT_X_AXIS) {
                                xAxis.setEnd(xAxis.getEnd() + timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        }

                } else {
                        double nonTimeScaleZoomAmount = xAxis.getEnd() - xAxis.getStart();
                        double timeScaleZoomAmount = yAxis.getEnd() - yAxis.getStart();

                        timeScaleZoomAmount        =        (timeScaleZoomAmount/100)        *
PlotConstants.ZOOMING_TIME_AXIS_PERCENTAGE;
                        nonTimeScaleZoomAmount      =      (nonTimeScaleZoomAmount/100)      *
PlotConstants.ZOOMING_TIME_AXIS_PERCENTAGE;

                        if (zoomAction == ZoomDirection.ZOOM_IN_HIGH_Y_AXIS) {
                                       yAxis.setEnd(yAxis.getEnd() - timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_HIGH_Y_AXIS) {
                                yAxis.setEnd(yAxis.getEnd() + timeScaleZoomAmount);
                                  markTimeZoomed();
                                 plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_IN_CENTER_Y_AXIS) {
                                       yAxis.setStart(yAxis.getStart()                      +
timeScaleZoomAmount);
                                       yAxis.setEnd(yAxis.getEnd() - timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_CENTER_Y_AXIS) {
                                yAxis.setStart(yAxis.getStart() - timeScaleZoomAmount);
                                yAxis.setEnd(yAxis.getEnd() + timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
```

```
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_LOW_Y_AXIS) {
                              yAxis.setStart(yAxis.getStart() + timeScaleZoomAmount);
                                markTimeZoomed();
                               plot.notifyObserversTimeChange();
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_LOW_Y_AXIS) {
                              yAxis.setStart(yAxis.getStart() - timeScaleZoomAmount);
                                markTimeZoomed();
                                plot.notifyObserversTimeChange();
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_LEFT_X_AXIS) {
                              xAxis.setStart(xAxis.getStart() + nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_LEFT_X_AXIS) {
                                     xAxis.setStart(xAxis.getStart()                       -
nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_CENTER_X_AXIS) {
                                     xAxis.setStart(xAxis.getStart()                       +
nonTimeScaleZoomAmount);
                                     xAxis.setEnd(xAxis.getEnd() - nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_CENTER_X_AXIS) {
                              xAxis.setStart(xAxis.getStart() - nonTimeScaleZoomAmount);
                              xAxis.setEnd(xAxis.getEnd() + nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_RIGHT_X_AXIS) {
                                     xAxis.setEnd(xAxis.getEnd() - nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_RIGHT_X_AXIS) {
                              xAxis.setEnd(xAxis.getEnd() + nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                       }
               }
               plot.plotAbstraction.updateResetButtons();
               plot.refreshDisplay();
               //Always request data refresh
               plot.limitManager.setModeUntranslated(false);
               plot.plotDataManager.resizeAndReloadPlotBuffer();
        }
}
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
```

```
 * information.
 *******************************************************************************/

package gov.nasa.arc.mct.fastplot.bridge;

import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.AxisOrientationSetting;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.PanDirection;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.PlotDisplayState;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.ZoomDirection;
import gov.nasa.arc.mct.fastplot.view.Axis;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import plotter.xy.XYAxis;

public class PanAndZoomManager {

        private          final          static          Logger          logger          =
LoggerFactory.getLogger(PanAndZoomManager.class);

        private PlotterPlot plot;

        private boolean inZoomMode;

        private boolean inPanMode;

        public PanAndZoomManager(PlotterPlot quinnCurtisPlot) {
                plot = quinnCurtisPlot;
        }

        public void enteredPanMode() {
                logger.debug("Entering pan mode");
                inPanMode = true;
                plot.setPlotDisplayState(PlotDisplayState.USER_INTERACTION);
        }

        public void exitedPanMode() {
                inPanMode = false;
                logger.debug("Exited pan mode");
        }

        public void enteredZoomMode() {
                logger.debug("Entered zoom mode");
                inZoomMode = true;
                plot.setPlotDisplayState(PlotDisplayState.USER_INTERACTION);
        }

        public void exitedZoomMode() {
                inZoomMode = false;
                logger.debug("Exited zoom mode");
        }


        public boolean isInZoomMode() {
                return inZoomMode;
        }


        public boolean isInPanMode() {
                return inPanMode;
        }
        public void panAction(PanDirection panningAction) {
                XYAxis xAxis = plot.plotView.getXAxis();
                XYAxis yAxis = plot.plotView.getYAxis();
                if                    (plot.getAxisOrientationSetting()                    ==
AxisOrientationSetting.X_AXIS_AS_TIME) {
                        double nonTimeScalePanAmount = yAxis.getEnd() - yAxis.getStart();
                        double timeScalePanAmount = xAxis.getEnd() - xAxis.getStart();

                        timeScalePanAmount         =         (timeScalePanAmount/100)         *
PlotConstants.PANNING_TIME_AXIS_PERCENTAGE;
                        nonTimeScalePanAmount=          (nonTimeScalePanAmount/100)          *
PlotConstants.PANNING_TIME_AXIS_PERCENTAGE;

                        if (panningAction == PanDirection.PAN_HIGHER_Y_AXIS) {
                                yAxis.shift(nonTimeScalePanAmount);
                                pinNonTime();
                        } else if (panningAction == PanDirection.PAN_LOWER_Y_AXIS) {
                                yAxis.shift(-nonTimeScalePanAmount);
                                pinNonTime();
                        } else if (panningAction == PanDirection.PAN_LOWER_X_AXIS) {
                                xAxis.shift(-timeScalePanAmount);
                                pinTime();
                                plot.notifyObserversTimeChange();
                        } else if (panningAction == PanDirection.PAN_HIGHER_X_AXIS) {
                                xAxis.shift(timeScalePanAmount);
                                pinTime();
                                plot.notifyObserversTimeChange();
                        }
                } else {

                        double nonTimeScalePanAmount = xAxis.getEnd() - xAxis.getStart();
                        double timeScalePanAmount = yAxis.getEnd() - yAxis.getStart();

                        timeScalePanAmount         =         (timeScalePanAmount/100)         *
PlotConstants.PANNING_TIME_AXIS_PERCENTAGE;
                        nonTimeScalePanAmount=          (nonTimeScalePanAmount/100)          *
PlotConstants.PANNING_TIME_AXIS_PERCENTAGE;

                        if (panningAction == PanDirection.PAN_HIGHER_Y_AXIS) {
                                yAxis.shift(timeScalePanAmount);
                                pinTime();
                                plot.notifyObserversTimeChange();
                        } else if (panningAction == PanDirection.PAN_LOWER_Y_AXIS) {
                                yAxis.shift(-timeScalePanAmount);
                                pinTime();
                                plot.notifyObserversTimeChange();
                        } else if (panningAction == PanDirection.PAN_LOWER_X_AXIS) {
                                xAxis.shift(-nonTimeScalePanAmount);
                                pinNonTime();
                        } else if (panningAction == PanDirection.PAN_HIGHER_X_AXIS) {
                                xAxis.shift(nonTimeScalePanAmount);
                                pinNonTime();
                        }
                }
                plot.plotAbstraction.updateResetButtons();
                plot.refreshDisplay();
                //Always request data refresh
                plot.clearAllDataFromPlot();
                plot.limitManager.setModeUntranslated(false);
                plot.plotAbstraction.requestPlotData(plot.getCurrentTimeAxisMin(),
plot.getCurrentTimeAxisMax());

        }


        private void pinTime() {
                plot.plotAbstraction.getTimeAxisUserPin().setPinned(true);
        }


        private void pinNonTime() {
                plot.getNonTimeAxisUserPin().setPinned(true);
                if (plot.limitManager.isUntranslated()) {
                        plot.limitManager.setModeUntranslated(false);
                }
        }


        private void markTimeZoomed() {
                Axis axis = plot.plotAbstraction.getTimeAxis();
                pinTime();
                axis.setZoomed(true);
                if (plot.limitManager.isUntranslated()) {
                        plot.limitManager.setModeUntranslated(false);
                }
        }

        private void markNonTimeZoomed() {
                Axis axis = plot.getNonTimeAxis();
                pinNonTime();
                axis.setZoomed(true);
                if (plot.limitManager.isUntranslated()) {
                        plot.limitManager.setModeUntranslated(false);
                }
        }

        public void zoomAction(ZoomDirection zoomAction) {
                XYAxis xAxis = plot.plotView.getXAxis();
                XYAxis yAxis = plot.plotView.getYAxis();
                if                   (plot.getAxisOrientationSetting()                      ==
AxisOrientationSetting.X_AXIS_AS_TIME) {
                        double nonTimeScaleZoomAmount = yAxis.getEnd() - yAxis.getStart();
                        double timeScaleZoomAmount = xAxis.getEnd() - xAxis.getStart();

                        timeScaleZoomAmount        =        (timeScaleZoomAmount/100)        *
PlotConstants.ZOOMING_TIME_AXIS_PERCENTAGE;
                        nonTimeScaleZoomAmount=         (nonTimeScaleZoomAmount/100)         *
PlotConstants.ZOOMING_TIME_AXIS_PERCENTAGE;

                        if (zoomAction == ZoomDirection.ZOOM_IN_HIGH_Y_AXIS) {
                                yAxis.setEnd(yAxis.getEnd() - nonTimeScaleZoomAmount);
                                  markNonTimeZoomed();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_HIGH_Y_AXIS) {
                                yAxis.setEnd(yAxis.getEnd() + nonTimeScaleZoomAmount);
                                  markNonTimeZoomed();
                        } else if (zoomAction == ZoomDirection.ZOOM_IN_CENTER_Y_AXIS) {
                                yAxis.setStart(yAxis.getStart() + nonTimeScaleZoomAmount);
                                yAxis.setEnd(yAxis.getEnd() - nonTimeScaleZoomAmount);
                                  markNonTimeZoomed();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_CENTER_Y_AXIS) {
                                yAxis.setStart(yAxis.getStart() - nonTimeScaleZoomAmount);
                                yAxis.setEnd(yAxis.getEnd() + nonTimeScaleZoomAmount);
                                  markNonTimeZoomed();
                        } else if (zoomAction == ZoomDirection.ZOOM_IN_LOW_Y_AXIS) {
                                yAxis.setStart(yAxis.getStart() + nonTimeScaleZoomAmount);
                                  markNonTimeZoomed();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_LOW_Y_AXIS) {
                                yAxis.setStart(yAxis.getStart() - nonTimeScaleZoomAmount);
                                  markNonTimeZoomed();
                        } else if (zoomAction == ZoomDirection.ZOOM_IN_LEFT_X_AXIS) {
                                xAxis.setStart(xAxis.getStart() + timeScaleZoomAmount);
                                  markTimeZoomed();
                                 plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_LEFT_X_AXIS) {
                                        xAxis.setStart(xAxis.getStart()                      -
timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_IN_CENTER_X_AXIS) {
                                xAxis.setStart(xAxis.getStart() + timeScaleZoomAmount);
                                xAxis.setEnd(xAxis.getEnd() - timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_CENTER_X_AXIS) {
                                xAxis.setStart(xAxis.getStart() - timeScaleZoomAmount);
                                xAxis.setEnd(xAxis.getEnd() + timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_IN_RIGHT_X_AXIS) {
                                xAxis.setEnd(xAxis.getEnd() - timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_RIGHT_X_AXIS) {
                                xAxis.setEnd(xAxis.getEnd() + timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        }

                } else {
                        double nonTimeScaleZoomAmount = xAxis.getEnd() - xAxis.getStart();
                        double timeScaleZoomAmount = yAxis.getEnd() - yAxis.getStart();

                        timeScaleZoomAmount        =        (timeScaleZoomAmount/100)        *
PlotConstants.ZOOMING_TIME_AXIS_PERCENTAGE;
                        nonTimeScaleZoomAmount      =      (nonTimeScaleZoomAmount/100)      *
PlotConstants.ZOOMING_TIME_AXIS_PERCENTAGE;

                        if (zoomAction == ZoomDirection.ZOOM_IN_HIGH_Y_AXIS) {
                                        yAxis.setEnd(yAxis.getEnd() - timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_HIGH_Y_AXIS) {
                                yAxis.setEnd(yAxis.getEnd() + timeScaleZoomAmount);
                                  markTimeZoomed();
                                 plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_IN_CENTER_Y_AXIS) {
                                        yAxis.setStart(yAxis.getStart()                      +
timeScaleZoomAmount);
                                        yAxis.setEnd(yAxis.getEnd() - timeScaleZoomAmount);
                                  markTimeZoomed();
                                  plot.notifyObserversTimeChange();
                        } else if (zoomAction == ZoomDirection.ZOOM_OUT_CENTER_Y_AXIS) {
                                yAxis.setStart(yAxis.getStart() - timeScaleZoomAmount);
```

```
                              yAxis.setEnd(yAxis.getEnd() + timeScaleZoomAmount);
                                markTimeZoomed();
                                plot.notifyObserversTimeChange();
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_LOW_Y_AXIS) {
                              yAxis.setStart(yAxis.getStart() + timeScaleZoomAmount);
                                markTimeZoomed();
                               plot.notifyObserversTimeChange();
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_LOW_Y_AXIS) {
                              yAxis.setStart(yAxis.getStart() - timeScaleZoomAmount);
                                markTimeZoomed();
                                plot.notifyObserversTimeChange();
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_LEFT_X_AXIS) {
                              xAxis.setStart(xAxis.getStart() + nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_LEFT_X_AXIS) {
                                      xAxis.setStart(xAxis.getStart()                       -
nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_CENTER_X_AXIS) {
                                      xAxis.setStart(xAxis.getStart()                       +
nonTimeScaleZoomAmount);
                                      xAxis.setEnd(xAxis.getEnd() - nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_CENTER_X_AXIS) {
                              xAxis.setStart(xAxis.getStart() - nonTimeScaleZoomAmount);
                              xAxis.setEnd(xAxis.getEnd() + nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_IN_RIGHT_X_AXIS) {
                                      xAxis.setEnd(xAxis.getEnd() - nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                       } else if (zoomAction == ZoomDirection.ZOOM_OUT_RIGHT_X_AXIS) {
                              xAxis.setEnd(xAxis.getEnd() + nonTimeScaleZoomAmount);
                                markNonTimeZoomed();
                       }
               }
               plot.plotAbstraction.updateResetButtons();
               plot.refreshDisplay();
               //Always request data refresh
               plot.limitManager.setModeUntranslated(false);
               plot.plotDataManager.resizeAndReloadPlotBuffer();
        }
}
```

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/bridge/PlotCornerResetButtonManager.java

Chunk 17: (new code/class declaration, import declaration, package declaration)

```
package gov.nasa.arc.mct.fastplot.bridge;

import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.AxisOrientationSetting;
import gov.nasa.arc.mct.fastplot.view.Axis;

import java.util.List;

/**
 * Manages the corner reset buttons on the plot area.
 */
public class PlotCornerResetButtonManager {
        PlotterPlot plot;
        public PlotCornerResetButtonManager(PlotterPlot thePlot) {
                plot = thePlot;
        }

        /**
         * Notify manager that the action of unpausing and snapping to the current time has
         * been selected.
         */
        void informJumpToCurrentTimeSelected() {
                // unpause the plot.
                plot.qcPlotObjects.fastForwardTimeAxisToCurrentMCTTime(false);
                plot.notifyObserversTimeChange();
                plot.plotAbstraction.getTimeAxisUserPin().setPinned(false);
                plot.plotAbstraction.updateResetButtons();
                refreshPlotValues();
        }

        /**
         * Notify manager that the action of resetting the Y axis has been selected.
         */
        void informResetYAxisActionSelected() {
                // perform axis reset.
                resetY();
                if (plot.isTimeLabelEnabled) {
                  rescalePlotOnTimeAxis();
                }
                plot.plotAbstraction.updateResetButtons();
                plot.refreshDisplay();
        }

        /**
         * Notify manager that the action of resetting the X axis has been selected.
         */
        void informResetXAxisActionSelected() {
                // perform axis reset.
            resetX();
            if (plot.isTimeLabelEnabled) {
                  rescalePlotOnTimeAxis();
            }
                plot.plotAbstraction.updateResetButtons();
                plot.refreshDisplay();
        }

        /**
         * Notify manager that the action of resetting both the X and Y axis has been
selected.
         */
        void informResetXAndYActionSelected() {
                resetX();
                resetY();
                rescalePlotOnTimeAxis();
                plot.plotAbstraction.updateResetButtons();
                plot.refreshDisplay();
        }

        /**
         * Perform the reset of the x-axis by either fast forwarding to the current time
         * if time is on the x axis or resetting the non time min max if time is on the y
axis.
         */
        void resetX() {
                if                    (plot.getAxisOrientationSetting()                    ==
AxisOrientationSetting.X_AXIS_AS_TIME) {
                        resetTimeAxis();
                } else {
                        resetNonTimeAxis();
                }
        }

        /**
         * Perform the reset of the y-axis by either fast forwarding to the current time
         * if time is on the y axis or resetting the non time min max if time is on the x
axis.
         */
        void resetY() {
                if                    (plot.getAxisOrientationSetting()                    ==
AxisOrientationSetting.X_AXIS_AS_TIME) {
                        resetNonTimeAxis();
                } else {
                        resetTimeAxis();
                }
        }

        private void resetTimeAxis() {
                Axis axis = plot.plotAbstraction.getTimeAxis();
                axis.setZoomed(false);
                plot.qcPlotObjects.fastForwardTimeAxisToCurrentMCTTime(true);
                plot.notifyObserversTimeChange();
                plot.plotAbstraction.getTimeAxisUserPin().setPinned(false);
                refreshPlotValues();
        }

        private void refreshPlotValues() {
                plot.clearAllDataFromPlot();
                plot.plotAbstraction.requestPlotData(plot.getCurrentTimeAxisMin(),
plot.getCurrentTimeAxisMax());
        }

        private void resetNonTimeAxis() {
                Axis axis = plot.getNonTimeAxis();
                plot.getNonTimeAxisUserPin().setPinned(false);
                axis.setZoomed(false);
                plot.qcPlotObjects.resetNonTimeAxisToOriginalValues();
                plot.setNonTimeMinFixed(plot.isNonTimeMinFixedByPlotSettings());
                plot.setNonTimeMaxFixed(plot.isNonTimeMaxFixedByPlotSettings());
                if (!plot.limitManager.isEnabled()) {
                        plot.limitManager.setEnabled(true);
                }
        }

        /**
         *  Rescale  the  plot  to  match  the  current  x-axis  settings  on  the  plot  time
coordinates
         */
        private void rescalePlotOnTimeAxis() {
                if (plot.isPaused()) {
                        plot.plotAbstraction.updateResetButtons();
                }
        }


        public void updateButtons() {
                Axis timeAxis = plot.plotAbstraction.getTimeAxis();
                Axis nonTimeAxis = plot.getNonTimeAxis();
                Axis xAxis;
                Axis yAxis;
                if(plot.getAxisOrientationSetting() == AxisOrientationSetting.X_AXIS_AS_TIME)
{
                        xAxis = timeAxis;
                        yAxis = nonTimeAxis;
                } else {
                        xAxis = nonTimeAxis;
                        yAxis = timeAxis;
                }

                List<AbstractPlottingPackage> plots = plot.plotAbstraction.getSubPlots();
                // Only show the top right reset button on the top plot.
                if(plots.get(0) == plot) {
                        // This was changed to fix MCT-2613: [Plot] Top right corner button
appears briefly in jump and scrunch modes, between the time that the plot line hits the end
of the time axis and when the jump
                        // The problem was that the jump occurs based on the maximum time
plotted, which due to compression, is not the same as the current MCT time.
                        // As an easy fix, the button is always hidden when the time axis is
not pinned.
                        // Assuming that data should never appear off the right of a jump
plot, this works well enough.
                        // If that assumption breaks, the code should be modified to check
against the maximum plotted time instead of the current MCT time.
                        long now = plot.plotAbstraction.getCurrentMCTTime();
                        if(!timeAxis.isPinned()) {

        plot.localControlsManager.setJumpToCurrentTimeButtonVisible(false);
                        } else if(plot.getMaxTime() < now || plot.getMinTime() > now) {

        plot.localControlsManager.setJumpToCurrentTimeButtonAlarm(true);
                        } else {

        plot.localControlsManager.setJumpToCurrentTimeButtonAlarm(false);
                        }
                } else {
                        plot.localControlsManager.setJumpToCurrentTimeButtonVisible(false);
                }
                // Only show the time axis reset button on the bottom plot.
                boolean enableX = true;
                boolean enableY = true;
                if(plots.get(plots.size() - 1) != plot) {
                        if(plot.getAxisOrientationSetting() ==
AxisOrientationSetting.X_AXIS_AS_TIME) {
                                enableX = false;
                        } else {
                                enableY = false;
                        }
                }

                plot.localControlsManager.setXAxisCornerResetButtonVisible(enableX &&
!xAxis.isInDefaultState());
                plot.localControlsManager.setYAxisCornerResetButtonVisible(enableY &&
!yAxis.isInDefaultState());

        plot.localControlsManager.setXAndYAxisCornerResetButtonVisible(!xAxis.isInDefaultSta
te() && !yAxis.isInDefaultState());
        }
}
=======
package gov.nasa.arc.mct.fastplot.bridge;

import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.AxisOrientationSetting;
import gov.nasa.arc.mct.fastplot.view.Axis;

import java.util.List;

/**
 * Manages the corner reset buttons on the plot area.
 */
public class PlotCornerResetButtonManager {
        PlotterPlot plot;

        public PlotCornerResetButtonManager(PlotterPlot thePlot) {
                plot = thePlot;
        }

        /**
         * Notify manager that the action of unpausing and snapping to the current time has
         * been selected.
         */
        void informJumpToCurrentTimeSelected() {
                // unpause the plot.
                plot.qcPlotObjects.fastForwardTimeAxisToCurrentMCTTime(false);
                plot.notifyObserversTimeChange();
                plot.plotAbstraction.getTimeAxisUserPin().setPinned(false);
                if (!plot.limitManager.isUntranslated()) {
                        plot.limitManager.setModeUntranslated(true);
                }
                plot.plotAbstraction.updateResetButtons();
                refreshPlotValues();
        }

        /**
         * Notify manager that the action of resetting the Y axis has been selected.
         */
        void informResetYAxisActionSelected() {
                // perform axis reset.
                resetY();
                if (plot.isTimeLabelEnabled) {
                  rescalePlotOnTimeAxis();
                }
                plot.plotAbstraction.updateResetButtons();
                plot.refreshDisplay();
        }

        /**
         * Notify manager that the action of resetting the X axis has been selected.
         */
        void informResetXAxisActionSelected() {
                // perform axis reset.
            resetX();
            if (plot.isTimeLabelEnabled) {
                  rescalePlotOnTimeAxis();
            }
                plot.plotAbstraction.updateResetButtons();
                plot.refreshDisplay();
        }

        /**
         * Notify manager that the action of resetting both the X and Y axis has been
selected.
```

```
         */
        void informResetXAndYActionSelected() {
                resetX();
                resetY();
                rescalePlotOnTimeAxis();
                plot.plotAbstraction.updateResetButtons();
                plot.refreshDisplay();
        }

        /**
         * Perform the reset of the x-axis by either fast forwarding to the current time
         * if time is on the x axis or resetting the non time min max if time is on the y
axis.
         */
        void resetX() {
                if (plot.axisOrientation == AxisOrientationSetting.X_AXIS_AS_TIME) {
                        resetTimeAxis();
                } else {
                        resetNonTimeAxis();
                }
                if (!plot.limitManager.isUntranslated()) {
                        plot.limitManager.setModeUntranslated(true);
                }
        }

        /**
         * Perform the reset of the y-axis by either fast forwarding to the current time
         * if time is on the y axis or resetting the non time min max if time is on the x
axis.
         */
        void resetY() {
                if (plot.axisOrientation == AxisOrientationSetting.X_AXIS_AS_TIME) {
                        resetNonTimeAxis();
                } else {
                        resetTimeAxis();
                }
        }

        private void resetTimeAxis() {
                Axis axis = plot.plotAbstraction.getTimeAxis();
                axis.setZoomed(false);
                plot.qcPlotObjects.fastForwardTimeAxisToCurrentMCTTime(true);
                plot.notifyObserversTimeChange();
                plot.plotAbstraction.getTimeAxisUserPin().setPinned(false);
                refreshPlotValues();
        }

        private void refreshPlotValues() {
                plot.clearAllDataFromPlot();
                plot.plotAbstraction.requestPlotData(plot.getCurrentTimeAxisMin(),
plot.getCurrentTimeAxisMax());
        }

        private void resetNonTimeAxis() {
                Axis axis = plot.getNonTimeAxis();
                plot.getNonTimeAxisUserPin().setPinned(false);
                axis.setZoomed(false);
                plot.qcPlotObjects.resetNonTimeAxisToOriginalValues();
                plot.setNonTimeMinFixed(plot.isNonTimeMinFixedByPlotSettings());
                plot.setNonTimeMaxFixed(plot.isNonTimeMaxFixedByPlotSettings());
                plot.limitManager.setModeUntranslated(true);
                refreshPlotValues();
        }

        /**
         * Rescale the plot to match the current x-axis settings on the plot time
coordinates
         */
        private void rescalePlotOnTimeAxis() {
                if (plot.isPaused()) {
                        plot.plotAbstraction.updateResetButtons();
                }
        }


        public void updateButtons() {
                Axis timeAxis = plot.plotAbstraction.getTimeAxis();
                Axis nonTimeAxis = plot.getNonTimeAxis();
                Axis xAxis;
                Axis yAxis;
                if(plot.axisOrientation == AxisOrientationSetting.X_AXIS_AS_TIME) {
                        xAxis = timeAxis;
                        yAxis = nonTimeAxis;
                } else {
                        xAxis = nonTimeAxis;
                        yAxis = timeAxis;
                }

                List<AbstractPlottingPackage> plots = plot.plotAbstraction.getSubPlots();
                // Only show the top right reset button on the top plot.
                if(plots.get(0) == plot) {
                        // This was changed to fix MCT-2613: [Plot] Top right corner button
appears briefly in jump and scrunch modes, between the time that the plot line hits the end
of the time axis and when the jump
                        // The problem was that the jump occurs based on the maximum time
plotted, which due to compression, is not the same as the current MCT time.
                        // As an easy fix, the button is always hidden when the time axis is
not pinned.
                        // Assuming that data should never appear off the right of a jump
plot, this works well enough.
                        // If that assumption breaks, the code should be modified to check
against the maximum plotted time instead of the current MCT time.
                        long now = plot.plotAbstraction.getCurrentMCTTime();
                        if(!timeAxis.isPinned()) {

        plot.localControlsManager.setJumpToCurrentTimeButtonVisible(false);
                        }    else    if(plot.getCurrentTimeAxisMaxAsLong()    <    now    ||
plot.getCurrentTimeAxisMinAsLong() > now) {

        plot.localControlsManager.setJumpToCurrentTimeButtonAlarm(true);
                        } else {

        plot.localControlsManager.setJumpToCurrentTimeButtonAlarm(false);
                        }
                } else {
                        plot.localControlsManager.setJumpToCurrentTimeButtonVisible(false);
                }
                // Only show the time axis reset button on the bottom plot.
                boolean enableX = true;
                boolean enableY = true;
                if(plots.get(plots.size() - 1) != plot) {
                        if(plot.axisOrientation == AxisOrientationSetting.X_AXIS_AS_TIME) {
                                enableX = false;
                        } else {
                                enableY = false;
                        }
                }

                plot.localControlsManager.setXAxisCornerResetButtonVisible(enableX         &&
!xAxis.isInDefaultState());
                plot.localControlsManager.setYAxisCornerResetButtonVisible(enableY         &&
!yAxis.isInDefaultState());

        plot.localControlsManager.setXAndYAxisCornerResetButtonVisible(!xAxis.isInDefaultSta
te() && !yAxis.isInDefaultState());
        }
}
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
```

```
/*******************************************************************************
 * Mission Control Technologies, Copyright (c) 2009-2012, United States Government
 * as represented by the Administrator of the National Aeronautics and Space
 * Administration. All rights reserved.
 *
 * The MCT platform is licensed under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0.
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
 * WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
 * License for the specific language governing permissions and limitations under
 * the License.
 *
 * MCT includes source code licensed under additional open source licenses. See
 * the MCT Open Source Licenses file included with this distribution or the About
 * MCT Licenses dialog available at runtime from the MCT Help menu for additional
 * information.
 *******************************************************************************/

package gov.nasa.arc.mct.fastplot.bridge;

import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.AxisOrientationSetting;
import gov.nasa.arc.mct.fastplot.view.Axis;

import java.util.List;

/**
 * Manages the corner reset buttons on the plot area.
 */
public class PlotCornerResetButtonManager {
        PlotterPlot plot;

        public PlotCornerResetButtonManager(PlotterPlot thePlot) {
                plot = thePlot;
        }

        /**
         * Notify manager that the action of unpausing and snapping to the current time has
         * been selected.
         */
        void informJumpToCurrentTimeSelected() {
                // unpause the plot.
                plot.qcPlotObjects.fastForwardTimeAxisToCurrentMCTTime(false);
                plot.notifyObserversTimeChange();
                plot.plotAbstraction.getTimeAxisUserPin().setPinned(false);
                if (!plot.limitManager.isUntranslated()) {
                        plot.limitManager.setModeUntranslated(true);
                }
                plot.plotAbstraction.updateResetButtons();
                refreshPlotValues();
        }

        /**
         * Notify manager that the action of resetting the Y axis has been selected.
         */
        void informResetYAxisActionSelected() {
                // perform axis reset.
                resetY();
                if (plot.isTimeLabelEnabled) {
                  rescalePlotOnTimeAxis();
                }
                plot.plotAbstraction.updateResetButtons();
                plot.refreshDisplay();
        }

        /**
         * Notify manager that the action of resetting the X axis has been selected.
         */
        void informResetXAxisActionSelected() {
                // perform axis reset.
            resetX();
            if (plot.isTimeLabelEnabled) {
                   rescalePlotOnTimeAxis();
            }
                plot.plotAbstraction.updateResetButtons();
                plot.refreshDisplay();
        }

        /**
         *  Notify  manager  that  the  action  of  resetting  both  the  X  and  Y  axis  has  been
selected.
         */
        void informResetXAndYActionSelected() {
                resetX();
                resetY();
                rescalePlotOnTimeAxis();
                plot.plotAbstraction.updateResetButtons();
                plot.refreshDisplay();
        }

        /**
         * Perform the reset of the x-axis by either fast forwarding to the current time
         * if time is on the x axis or resetting the non time min max if time is on the y
axis.
         */
        void resetX() {
                if                    (plot.getAxisOrientationSetting()                    ==
AxisOrientationSetting.X_AXIS_AS_TIME) {
                        resetTimeAxis();
                } else {
                        resetNonTimeAxis();
                }
                if (!plot.limitManager.isUntranslated()) {
                        plot.limitManager.setModeUntranslated(true);
                }
        }

        /**
         * Perform the reset of the y-axis by either fast forwarding to the current time
         * if time is on the y axis or resetting the non time min max if time is on the x
axis.
         */
        void resetY() {
                if                    (plot.getAxisOrientationSetting()                    ==
AxisOrientationSetting.X_AXIS_AS_TIME) {
                        resetNonTimeAxis();
                } else {
                        resetTimeAxis();
                }
        }

        private void resetTimeAxis() {
                Axis axis = plot.plotAbstraction.getTimeAxis();
                axis.setZoomed(false);
                plot.qcPlotObjects.fastForwardTimeAxisToCurrentMCTTime(true);
                plot.notifyObserversTimeChange();
                plot.plotAbstraction.getTimeAxisUserPin().setPinned(false);
                refreshPlotValues();
        }

        private void refreshPlotValues() {
                plot.clearAllDataFromPlot();
                plot.plotAbstraction.requestPlotData(plot.getCurrentTimeAxisMin(),
plot.getCurrentTimeAxisMax());
        }

        private void resetNonTimeAxis() {
                Axis axis = plot.getNonTimeAxis();
                plot.getNonTimeAxisUserPin().setPinned(false);
                axis.setZoomed(false);
                plot.qcPlotObjects.resetNonTimeAxisToOriginalValues();
                plot.setNonTimeMinFixed(plot.isNonTimeMinFixedByPlotSettings());
                plot.setNonTimeMaxFixed(plot.isNonTimeMaxFixedByPlotSettings());
                plot.limitManager.setModeUntranslated(true);
                refreshPlotValues();
        }

        /**
         * Rescale the plot to match the current x-axis settings on the plot time
coordinates
         */
        private void rescalePlotOnTimeAxis() {
                if (plot.isPaused()) {
                        plot.plotAbstraction.updateResetButtons();
                }
        }


        public void updateButtons() {
                Axis timeAxis = plot.plotAbstraction.getTimeAxis();
                Axis nonTimeAxis = plot.getNonTimeAxis();
                Axis xAxis;
                Axis yAxis;
                if(plot.getAxisOrientationSetting() == AxisOrientationSetting.X_AXIS_AS_TIME)
{
                        xAxis = timeAxis;
                        yAxis = nonTimeAxis;
                } else {
                        xAxis = nonTimeAxis;
                        yAxis = timeAxis;
                }

                List<AbstractPlottingPackage> plots = plot.plotAbstraction.getSubPlots();
                // Only show the top right reset button on the top plot.
                if(plots.get(0) == plot) {
                        // This was changed to fix MCT-2613: [Plot] Top right corner button
appears briefly in jump and scrunch modes, between the time that the plot line hits the end
of the time axis and when the jump
                        // The problem was that the jump occurs based on the maximum time
plotted, which due to compression, is not the same as the current MCT time.
                        // As an easy fix, the button is always hidden when the time axis is
not pinned.
                        // Assuming that data should never appear off the right of a jump
plot, this works well enough.
                        // If that assumption breaks, the code should be modified to check
against the maximum plotted time instead of the current MCT time.
                        long now = plot.plotAbstraction.getCurrentMCTTime();
                        if(!timeAxis.isPinned()) {

        plot.localControlsManager.setJumpToCurrentTimeButtonVisible(false);
                        } else if(plot.getMaxTime() < now || plot.getMinTime() > now) {

        plot.localControlsManager.setJumpToCurrentTimeButtonAlarm(true);
                        } else {

        plot.localControlsManager.setJumpToCurrentTimeButtonAlarm(false);
                        }
                } else {
                        plot.localControlsManager.setJumpToCurrentTimeButtonVisible(false);
                }
                // Only show the time axis reset button on the bottom plot.
                boolean enableX = true;
                boolean enableY = true;
                if(plots.get(plots.size() - 1) != plot) {
                        if(plot.getAxisOrientationSetting()                                ==
AxisOrientationSetting.X_AXIS_AS_TIME) {
                                enableX = false;
                        } else {
                                enableY = false;
                        }
                }

                plot.localControlsManager.setXAxisCornerResetButtonVisible(enableX         &&
!xAxis.isInDefaultState());
                plot.localControlsManager.setYAxisCornerResetButtonVisible(enableY         &&
!yAxis.isInDefaultState());

        plot.localControlsManager.setXAndYAxisCornerResetButtonVisible(!xAxis.isInDefaultSta
te() && !yAxis.isInDefaultState());
        }
}
```

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/bridge/PlotLimitManager.java

Chunk 18: (new code/ if statement, method invocation, variable)

```
        /**
```

```
         * Inform limit manager of the most recently plotted time.
         * @param atTime time at which point was plotted
         */
        public void informPointPlottedAtTime(long atTime, double value) {
<<<<<<< HEAD
                boolean checkMax = plot.getNonTimeAxisSubsequentMaxSetting() ==
NonTimeAxisSubsequentBoundsSetting.FIXED
                                || plot.getNonTimeAxisSubsequentMaxSetting() ==
NonTimeAxisSubsequentBoundsSetting.SEMI_FIXED;
                if(checkMax && (value >= plot.getMinNonTime() ||
                                nonTimeValueWithin1PixelOfLimit(value,
plot.nonTimeAxisMaxPhysicalValue))) {
                        if (nonTimeMaxAlarm != LimitAlarmState.ALARM_OPENED_BY_USER &&
plot.isNonTimeMaxFixed()) {
=======
                boolean checkMax = plot.nonTimeAxisMaxSubsequentSetting ==
NonTimeAxisSubsequentBoundsSetting.FIXED
                                || plot.nonTimeAxisMaxSubsequentSetting ==
NonTimeAxisSubsequentBoundsSetting.SEMI_FIXED
                                || plot.getNonTimeAxis().isPinned()
                                || plot.plotAbstraction.getTimeAxisUserPin().isPinned()
                                || plot.plotAbstraction.getTimeAxis().isPinned()
                                || plot.plotAbstraction.getTimeAxis().isZoomed();
                if(checkMax &&
                                (value >= plot.getCurrentNonTimeAxisMax()
                                || nonTimeValueWithin1PixelOfLimit(value,
plot.nonTimeAxisMaxPhysicalValue))
                                && atTime >= plot.getCurrentTimeAxisMinAsLong() && atTime <=
plot.getCurrentTimeAxisMaxAsLong()) {
                        if (nonTimeMaxAlarm != LimitAlarmState.ALARM_OPENED_BY_USER ) {
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96

                                boolean wasOpen = nonTimeMaxAlarm ==
LimitAlarmState.ALARM_RAISED;
                                nonTimeMaxAlarm = LimitAlarmState.ALARM_RAISED;
                                maxAlarmMostRecentTime = atTime;
                                if(!wasOpen) {
                                        addMaxAlertButton();
<<<<<<< HEAD
                                        if(plot.getNonTimeAxisSubsequentMaxSetting() ==
NonTimeAxisSubsequentBoundsSetting.SEMI_FIXED) {
=======
                                        if(plot.nonTimeAxisMaxSubsequentSetting ==
NonTimeAxisSubsequentBoundsSetting.SEMI_FIXED &&
                                                        !plot.getNonTimeAxis().isPinned()) {
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
                                                processMaxAlertButtonPress();
                                        }
                                }
                        }
<<<<<<< HEAD
                }
                boolean checkMin = plot.getNonTimeAxisSubsequentMinSetting() ==
NonTimeAxisSubsequentBoundsSetting.FIXED
                                || plot.getNonTimeAxisSubsequentMinSetting() ==
NonTimeAxisSubsequentBoundsSetting.SEMI_FIXED;
                if(checkMin && (value <= plot.getMinNonTime() ||
                                nonTimeValueWithin1PixelOfLimit(value,
plot.nonTimeAxisMinPhysicalValue))) {
                        if (nonTimeMinAlarm != LimitAlarmState.ALARM_OPENED_BY_USER &&
plot.isNonTimeMinFixed()) {
=======
                }

                boolean      checkMin      =      plot.nonTimeAxisMinSubsequentSetting      ==
NonTimeAxisSubsequentBoundsSetting.FIXED
                              ||          plot.nonTimeAxisMinSubsequentSetting           ==
NonTimeAxisSubsequentBoundsSetting.SEMI_FIXED
                              || plot.getNonTimeAxis().isPinned()
                              || plot.plotAbstraction.getTimeAxisUserPin().isPinned()
                              || plot.plotAbstraction.getTimeAxis().isPinned()
                              || plot.plotAbstraction.getTimeAxis().isZoomed();
                if(checkMin && (value <= plot.getCurrentNonTimeAxisMin() ||
                              nonTimeValueWithin1PixelOfLimit(value,
plot.nonTimeAxisMinPhysicalValue)) &&
                              atTime  >=  plot.getCurrentTimeAxisMinAsLong()  &&  atTime  <=
plot.getCurrentTimeAxisMaxAsLong()) {
                        if (nonTimeMinAlarm != LimitAlarmState.ALARM_OPENED_BY_USER ) {
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96

                              boolean        wasOpen        =        nonTimeMinAlarm        ==
LimitAlarmState.ALARM_RAISED;
                              nonTimeMinAlarm = LimitAlarmState.ALARM_RAISED;
                              minAlarmMostRecentTime = atTime;
                              if(!wasOpen) {
                                      addMinAlertButton();
<<<<<<< HEAD
                                      if(plot.getNonTimeAxisSubsequentMinSetting()         ==
NonTimeAxisSubsequentBoundsSetting.SEMI_FIXED) {
=======
                                      if(plot.nonTimeAxisMinSubsequentSetting             ==
NonTimeAxisSubsequentBoundsSetting.SEMI_FIXED &&
                                                      !plot.getNonTimeAxis().isPinned()) {
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
                                              processMinAlertButtonPress();
                                      }
                              }
                      }
                }
```

```
        /**
         * Inform limit manager of the most recently plotted time.
         * @param atTime time at which point was plotted
         */
        public void informPointPlottedAtTime(long atTime, double value) {

                boolean     checkMax     =     plot.getNonTimeAxisSubsequentMinSetting()     ==
NonTimeAxisSubsequentBoundsSetting.FIXED
                              ||        plot.getNonTimeAxisSubsequentMaxSetting()         ==
NonTimeAxisSubsequentBoundsSetting.SEMI_FIXED
                              || plot.getNonTimeAxis().isPinned()
                              || plot.plotAbstraction.getTimeAxisUserPin().isPinned()
                              || plot.plotAbstraction.getTimeAxis().isPinned()
                              || plot.plotAbstraction.getTimeAxis().isZoomed();
                if(checkMax &&
                              (value >= plot.getMaxNonTime()
                              ||                      nonTimeValueWithin1PixelOfLimit(value,
plot.nonTimeAxisMaxPhysicalValue))
                              && atTime >= plot.getMinTime() && atTime <= plot.getMaxTime())
{
                        if (nonTimeMaxAlarm != LimitAlarmState.ALARM_OPENED_BY_USER ) {
                              boolean       wasOpen       =       nonTimeMaxAlarm       ==
LimitAlarmState.ALARM_RAISED;
                              nonTimeMaxAlarm = LimitAlarmState.ALARM_RAISED;
                              maxAlarmMostRecentTime = atTime;
                              if(!wasOpen) {
                                     addMaxAlertButton();

                                     if(plot.getNonTimeAxisSubsequentMaxSetting()        ==
NonTimeAxisSubsequentBoundsSetting.SEMI_FIXED &&
                                                   !plot.getNonTimeAxis().isPinned()) {

                                            processMaxAlertButtonPress();
                                     }
                              }
                       }
                }

                boolean    checkMin    =    plot.getNonTimeAxisSubsequentMinSetting()    ==
NonTimeAxisSubsequentBoundsSetting.FIXED
                              ||        plot.getNonTimeAxisSubsequentMaxSetting()        ==
NonTimeAxisSubsequentBoundsSetting.SEMI_FIXED
                              || plot.getNonTimeAxis().isPinned()
                              || plot.plotAbstraction.getTimeAxisUserPin().isPinned()
                              || plot.plotAbstraction.getTimeAxis().isPinned()
                              || plot.plotAbstraction.getTimeAxis().isZoomed();
                if(checkMin && (value <= plot.getMinNonTime() ||
                              nonTimeValueWithin1PixelOfLimit(value,
plot.nonTimeAxisMinPhysicalValue)) &&
                              atTime >= plot.getMinTime() && atTime <= plot.getMaxTime()) {
                       if (nonTimeMinAlarm != LimitAlarmState.ALARM_OPENED_BY_USER ) {

                              boolean       wasOpen       =       nonTimeMinAlarm       ==
LimitAlarmState.ALARM_RAISED;
                              nonTimeMinAlarm = LimitAlarmState.ALARM_RAISED;
                              minAlarmMostRecentTime = atTime;
                              if(!wasOpen) {
                                     addMinAlertButton();

                                     if(plot.getNonTimeAxisSubsequentMinSetting()        ==
NonTimeAxisSubsequentBoundsSetting.SEMI_FIXED &&
                                                   !plot.getNonTimeAxis().isPinned()) {
                                            processMinAlertButtonPress();
                                     }
                              }
                       }
                }

                // Check upper alarm still valid

        // Only check if an alarm a max alarm is raised and limit indicators showing.

                if (checkMax && nonTimeMaxAlarm != LimitAlarmState.NO_ALARM) {
                       if (plot.getMinTime() > maxAlarmMostRecentTime) {
                              // alarm has scrolled off.
                              nonTimeMaxAlarm = LimitAlarmState.NO_ALARM;
                              nonTimeMaxLimitButton.setVisible(false);
                       }
                }

                // Check lower alarm still valid
                // Only check if an alarm a max alarm is raised.
        if (checkMin && nonTimeMinAlarm != LimitAlarmState.NO_ALARM) {
```

```
        if (plot.getMinTime() > minAlarmMostRecentTime) {
           nonTimeMinAlarm = LimitAlarmState.NO_ALARM;
           nonTimeMinLimitButton.setVisible(false);
                   }
           }

    if (checkMin || checkMax) {
           plot.newPointPlotted(atTime, value);
    }

    }

    private void changeButtonIcon(JButton button, ImageIcon newIcon, String tooltip) {
           button.setIcon(newIcon);
           button.setToolTipText(tooltip);
    }

    private void setMaxAlarmIconToAlarmRaised() {
           changeButtonIcon(nonTimeMaxLimitButton, nonTimeMaxLimitAlarmRaisedIcon,
                          BUNDLE.getString("ShowAllData"));
    }

    private void setMaxAlarmIconToAlarmOpendedByUser() {
           changeButtonIcon(nonTimeMaxLimitButton, nonTimeMaxLimitAlarmOpenedByUserIcon,
                          BUNDLE.getString("HideOOBData"));
    }

    private void setMaxAlarmIconToAlarmClosedByUser() {
           changeButtonIcon(nonTimeMaxLimitButton,nonTimeMaxLimitAlarmClosedByUserIcon,
                          BUNDLE.getString("ShowAllDataAgain"));
    }

    private void setMinAlarmIconToAlarmRaised() {
           changeButtonIcon(nonTimeMinLimitButton, nonTimeMinLimitAlarmRaisedIcon,
                          BUNDLE.getString("ShowAllData"));
    }

    private void setMinAlarmIconToAlarmOpendedByUser() {
           changeButtonIcon(nonTimeMinLimitButton, nonTimeMinLimitAlarmOpenedByUserIcon,
                          BUNDLE.getString("HideOOBData"));
    }

    private void setMinAlarmIconToAlarmClosedByUser() {
           changeButtonIcon(nonTimeMinLimitButton, nonTimeMinLimitAlarmClosedByUserIcon,
                          BUNDLE.getString("ShowAllDataAgain"));
}
```

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/bridge/PlotterPlot.java

Chunk 19: (combination/annotation, method declaration, method invocation)

```
        void resetNonTimeMin() {
<<<<<<< HEAD
                adjustAxis(getInitialNonTimeMinSetting(), getMaxNonTime());
=======
                adjustAxis(getInitialNonTimeMinSetting(), getCurrentNonTimeAxisMax());
        }

        @Override
        public NonTimeAxisSubsequentBoundsSetting getNonTimeAxisSubsequentMaxSetting() {
                return nonTimeAxisMaxSubsequentSetting;
        }

        @Override
        public NonTimeAxisSubsequentBoundsSetting getNonTimeAxisSubsequentMinSetting() {
                return nonTimeAxisMinSubsequentSetting;
        }

        @Override
        public double getInitialNonTimeMaxSetting() {
                if (getNonTimeAxis().isInDefaultState()) {
                        return nonTimeVaribleAxisMaxValue;
                } else {
                        return limitManager.getCachedNonTimeMaxValue();
                }
        }

        @Override
        public double getNonTimeMaxPadding() {
                return scrollRescaleMarginNonTimeMax;
        }

        @Override
        public double getInitialNonTimeMinSetting() {
                if (getNonTimeAxis().isInDefaultState()) {
                return nonTimeVaribleAxisMinValue;
                } else {
                        return limitManager.getCachedNonTimeMinValue();
                }
        }

        @Override
        public double getNonTimeMinPadding() {
                return scrollRescaleMarginNonTimeMin;
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
        }
```

```
        void resetNonTimeMin() {
                adjustAxis(getInitialNonTimeMinSetting(), getMaxNonTime());
        }


        @Override
        public double getInitialNonTimeMaxSetting() {
                if (getNonTimeAxis().isInDefaultState()) {
                        return super.getMaxNonTime();
                } else {
                        return limitManager.getCachedNonTimeMaxValue();
                }
        }

        @Override
        public double getNonTimeMaxPadding() {
                return scrollRescaleMarginNonTimeMax;
        }

        @Override
        public double getInitialNonTimeMinSetting() {
                if (getNonTimeAxis().isInDefaultState()) {
                        return super.getMinNonTime();
                } else {
                        return limitManager.getCachedNonTimeMinValue();
                }
        }
```

fastPlotViews/src/test/java/gov/nasa/arc/mct/fastplot/bridge/TestLimitArrowIndicators.java

Chunk 20: (version 1/other)

```
                                   Color.white,
                                   "dd",
                                   Color.black,
                                   Color.white,
                                   1,
<<<<<<< HEAD
=======
                                   0.5,
                                   0.5,
                                   0.5,
                                   0.0,
                                   10.0,
                                   now.getTimeInMillis(),
                                   now.getTimeInMillis() + (5L * 60L * 1000L),
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
                                   false,
                                   true,
                                   true,
                                   testPlot,
                                   plotLabelingAlgorithm);
```

```
                                   Color.white,
                                   Color.white,
                                   Color.white,
                                   "dd",
                                   Color.black,
                                   Color.white,
                                   1,
                                   false,
                                   true,
                                   true,
                                   testPlot,
                                   plotLabelingAlgorithm);
```

Chunk 21: (version 1/other)

```
                                   "dd",
                                   Color.black,
<<<<<<< HEAD
                                   Color.white,
                                   1,
=======
                                   Color.white,
                                   1,
                                   0.5,
                                   0.5,
                                   0.5,
                                   0,
                                   10,
                                   now.getTimeInMillis(),
                                   now.getTimeInMillis() + (5L * 60L * 1000L),
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
                                   false,
                                   true,
                                   true,
```

```
                                   "dd",
                                   Color.black,
                                   Color.white,
                                   1,
                                   false,
                                   true,
                                   true,
```

Chunk 22: (version 1/other)

```
                                   Color.black,
                                   Color.white,
<<<<<<< HEAD
                                   1,
=======
                                   1,
                                   0.5,
                                   0.5,
                                   0.5,
                                   0,
                                   10,
                                   now.getTimeInMillis(),
                                   now.getTimeInMillis() + (5L * 60L * 1000L),
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
                                   false,
                                   true,
                                   true,
```

```
                                   Color.black,
                                   Color.white,
                                   1,
                                   false,
                                   true,
                                   true,
```

Chunk 23: (combination/method invocation)

```
<<<<<<< HEAD
          Assert.assertFalse(plot.isCompressionEnabled());

=======
          Assert.assertFalse(plot.isCompresionEnabled());
          now.add(Calendar.MINUTE, 1);
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
```

```
          Assert.assertFalse(plot.isCompressionEnabled());
          now.add(Calendar.MINUTE, 1);
```

Chunk 24: (version 1/other)

```
                                   "dd",
                                   Color.black,
                                   Color.white,
                                   1,
<<<<<<< HEAD
=======
                                   0.5,
                                   0.5,
                                   0.5,
                                   0,
                                   10,
                                   now.getTimeInMillis(),
                                   now.getTimeInMillis() + (5L * 60L * 1000L),
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
                                   false,
                                   true,
                                   true,
                                   testPlot,
```

```
                                   "dd",
                                   Color.black,
                                   Color.white,
                                   1,
                                   false,
                                   true,
                                   true,
                                   testPlot,
```

Chunk 25: (version 1/other)

```
                                                                 Color.black,
                                                                 Color.white,
                                                                 1,
<<<<<<< HEAD
=======
                                                                 0.5,
                                                                 0.5,
                                                                 0.5,
                                                                 0,
                                                                 10,
                                                                 currentTime,
                                                                 10L,
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
                                                                 false,
                                                                 true,
                                                                 true,
```

```
                                                         Color.black,
                                                                 Color.white,
                                                                 1,
                                                                 false,
                                                                 true,
                                                                 true,
```

Chunk 26: (version 1/other)

```
                                                                 "dd",
                                                                 Color.black,
                                                                 Color.white,
                                                                 1,
<<<<<<< HEAD
=======
                                                                 0.5,
                                                                 0.5,
                                                                 0.5,
                                                                 0,
                                                                 10,
                                                                 currentTime,
                                                                 10,
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
                                                                 false,
                                                                 true,
                                                                 true,
```

```
                                                                  "dd",
                                                                  Color.black,
                                                                  Color.white,
                                                                  1,
                                                                  false,
                                                                  true,
                                                                  true,
```

fastPlotViews/src/test/java/gov/nasa/arc/mct/fastplot/bridge/TestPanAndZoomManager.java

Chunk 27: (version 1/method invocation)

```
                 PlotAbstraction testPlotTimeX = new PlotView.Builder(PlotterPlot.class).
<<<<<<< HEAD
                                                  plotSettings(settings).
=======
                                                  axisOrientation(AxisOrientationSetting.X_AXIS_AS_TIME).
                                             nonTimeVaribleAxisMaxValue(100).
                                             nonTimeVaribleAxisMinValue(0).
                                             timeVariableAxisMinValue(now).
                                             timeVariableAxisMaxValue(now + 300000L).

nonTimeAxisMinSubsequentSetting(PlotConstants.NonTimeAxisSubsequentBoundsSetting.FIXED).

nonTimeAxisMaxSubsequentSetting(PlotConstants.NonTimeAxisSubsequentBoundsSetting.FIXED).
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
                                             build();
```

```
                 PlotAbstraction testPlotTimeX = new PlotView.Builder(PlotterPlot.class).
                                                  plotSettings(settings).
                                             build();
```

Chunk 28: (combination/ method invocation, variable)

```
                 settings.setMinNonTime(0);

<<<<<<< HEAD
                 PlotAbstraction testPlotTimeY = new PlotView.Builder(PlotterPlot.class).
                                                                    plotSettings(settings2).
                                                                    build();

=======
            PlotAbstraction testPlotTimeY = new PlotView.Builder(PlotterPlot.class).
        axisOrientation(AxisOrientationSetting.Y_AXIS_AS_TIME).
        nonTimeVaribleAxisMaxValue(100).
        nonTimeVaribleAxisMinValue(0).
        timeVariableAxisMinValue(now).
        timeVariableAxisMaxValue(now + 300000L).
        nonTimeAxisMinSubsequentSetting(PlotConstants.NonTimeAxisSubsequentBoundsSetting.FIXED).
        nonTimeAxisMaxSubsequentSetting(PlotConstants.NonTimeAxisSubsequentBoundsSetting.FIXED).
        build();
>>>>>>> 42cbc3f5b68e1b67340a44ddd6fca5daf1147a96
        plotTimeOnY = (PlotterPlot) testPlotTimeY.returnPlottingPackage();
```

```
                 settings.setMinNonTime(0);
                 settings.setMinTime(now);
                 settings.setMaxTime(now + 300000L);
                 settings.setNonTimeAxisSubsequentMinSetting(NonTimeAxisSubsequentBoundsSetting.FIXED);
                 settings.setNonTimeAxisSubsequentMaxSetting(NonTimeAxisSubsequentBoundsSetting.FIXED);

                 PlotAbstraction testPlotTimeY = new PlotView.Builder(PlotterPlot.class).
                                                                    plotSettings(settings2).
                                                                    build();

        plotTimeOnY = (PlotterPlot) testPlotTimeY.returnPlottingPackage();
```

Version: b39eefc494a0bebc2d7f984fc2caad98fbfbcf0f

Parents:

98ba0e68473743b825d2ebc69146dfc1bdce9322
ee36c4dcd4fb26c4fd2ec81bc7d2c2e180b38f2e

Merge base:

0cc9d801458f8daaca55ba149887e96b978729d5

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/bridge/PlotterPlot.java

Chunk 29: (combination/method declaration)

```
        @Override
        public void updateCompressionRatio() {
                plotDataManager.setupCompressionRatio();
        }
<<<<<<< HEAD

        public PlotLocalControlsManager getLocalControlsManager() {
                return localControlsManager;
        }

        public void setLocalControlsManager(PlotLocalControlsManager localControlsManager) {
                this.localControlsManager = localControlsManager;
=======

        public String getTimeSystemSetting() {
                return timeSystemSetting;
        }

        public String getTimeFormatSetting() {
                return timeFormatSetting;
        }

        public static NumberFormat getNumberFormatter(double value) {
                NumberFormat format = PlotConstants.DECIMAL_FORMAT;

                try {
                        if ( (value >= PlotConstants.MILLION_VALUES) || (value <=
PlotConstants.NEGATIVE_MILLION_VALUES) ) {
                                format = new
DecimalFormat(PlotConstants.SCIENTIFIC_NUMBER_FORMAT);
                        }
                } catch (NumberFormatException nfe) {
                        logger.error("NumberFormatException in very large numbers: {}", nfe);
                }
         return format;
>>>>>>> ee36c4dcd4fb26c4fd2ec81bc7d2c2e180b38f2e
        }
}
```

```
        @Override
        public void updateCompressionRatio() {
                plotDataManager.setupCompressionRatio();
        }

        public PlotLocalControlsManager getLocalControlsManager() {
                return localControlsManager;
        }

        public void setLocalControlsManager(PlotLocalControlsManager localControlsManager) {
                this.localControlsManager = localControlsManager;
        }

        public String getTimeSystemSetting() {
                return timeSystemSetting;
        }

        public String getTimeFormatSetting() {
                return timeFormatSetting;
        }

        public static NumberFormat getNumberFormatter(double value) {
                NumberFormat format = PlotConstants.DECIMAL_FORMAT;

                try {
                        if   (   (value   >=   PlotConstants.MILLION_VALUES)   ||   (value   <=
PlotConstants.NEGATIVE_MILLION_VALUES) ) {
                                format                          =                          new
DecimalFormat(PlotConstants.SCIENTIFIC_NUMBER_FORMAT);
                        }
                } catch (NumberFormatException nfe) {
                        logger.error("NumberFormatException in very large numbers: {}", nfe);
                }
        return format;
        }
}
```

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/view/PlotViewManifestation.java

Chunk 30: (concatenation/ import declaration)

```
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants;
<<<<<<< HEAD
import gov.nasa.arc.mct.fastplot.bridge.PlotterPlot;
=======
import gov.nasa.arc.mct.fastplot.bridge.PlotAbstraction.PlotSettings;
>>>>>>> ee36c4dcd4fb26c4fd2ec81bc7d2c2e180b38f2e
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.AxisOrientationSetting;
```

```
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants;
import gov.nasa.arc.mct.fastplot.bridge.PlotterPlot;
import gov.nasa.arc.mct.fastplot.bridge.PlotAbstraction.PlotSettings;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.AxisOrientationSetting;
```

Version: 11fac9cbd0be898f46e1b7e66dabfe3c0983580e

Parents:

ec2ba4b92febdc8ad5ed15f29cfb21316f7626b2
790f22af0041eb44189aeb623cdd8b856912b4a7

Merge base:

77d62d50f08cd5c2c06a3586e1739c8027612f03

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/view/PlotSettingsControlPanel.java

Chunk 31: (version 2/ method invocation)

```
               nonTimeMinAutoAdjustMode.addActionListener(buttonListener);
<<<<<<< HEAD
               nonTimeMinFixedMode.addActionListener(buttonListener);
               nonTimeMinSemiFixedMode.addActionListener(buttonListener);
               nonTimeMaxAutoAdjustMode.addActionListener(buttonListener);
               nonTimeMaxFixedMode.addActionListener(buttonListener);
               nonTimeMaxSemiFixedMode.addActionListener(buttonListener);
=======
               nonTimeMinFixedMode.addActionListener(buttonListener);
               nonTimeMinSemiFixedMode.addActionListener(buttonListener);
               nonTimeMaxAutoAdjustMode.addActionListener(buttonListener);
               nonTimeMaxFixedMode.addActionListener(buttonListener);
               nonTimeMaxSemiFixedMode.addActionListener(buttonListener);
>>>>>>> 790f22af0041eb44189aeb623cdd8b856912b4a7

               // Add listeners to the Time axis buttons
```

```
               nonTimeMinAutoAdjustMode.addActionListener(buttonListener);
               nonTimeMinFixedMode.addActionListener(buttonListener);
               nonTimeMinSemiFixedMode.addActionListener(buttonListener);
               nonTimeMaxAutoAdjustMode.addActionListener(buttonListener);
               nonTimeMaxFixedMode.addActionListener(buttonListener);
               nonTimeMaxSemiFixedMode.addActionListener(buttonListener);

               // Add listeners to the Time axis buttons
```

Version: ef2cdd4b69a60aabba8992fb81abfa3d6791b745

Parents:

661c0840da5fe3b0eecb5d4c33a1055944b130fc
b9ba630731c3c48e787c9cd06b62209bf6776ebb

Merge base:

032b52920457ee02fc4cded34a82978882aaa767

platform/src/main/java/gov/nasa/arc/mct/platform/PlatformImpl.java

Chunk 32: (combination/ commentary, method declaration, method invocation, return statement)

```
    @Override
    public AbstractComponent getUserDropboxes() {
<<<<<<< HEAD
        return userDropboxesId == null ? null :
getPersistenceProvider().getComponent(userDropboxesId);
=======
        return getPersistenceProvider().getComponent(userDropboxesId);
    }

    /**
     * Gets the OSGi FeedManager reference.
     * @return FeedManager reference.
     */
    public FeedManager getFeedManager() {
        OSGIRuntime osgiRuntime = EquinoxOSGIRuntimeImpl.getOSGIRuntime();
        return osgiRuntime.getService(FeedManager.class, null);
    }

    /**
     * Gets the OSGi FeedDataArchive reference.
     * @return FeedDataArchive reference.
     */
    public FeedDataArchive getFeedDataArchive() {
        OSGIRuntime osgiRuntime = EquinoxOSGIRuntimeImpl.getOSGIRuntime();
        return osgiRuntime.getService(FeedDataArchive.class, null);
>>>>>>> b9ba630731c3c48e787c9cd06b62209bf6776ebb
    }
}
```

```
@Override
    public AbstractComponent getUserDropboxes() {
        return userDropboxesId == null ? null :
getPersistenceProvider().getComponent(userDropboxesId);
    }

    /**
     * Gets the OSGi FeedManager reference.
     * @return FeedManager reference.
     */
    public FeedManager getFeedManager() {
        OSGIRuntime osgiRuntime = EquinoxOSGIRuntimeImpl.getOSGIRuntime();
        return osgiRuntime.getService(FeedManager.class, null);
    }

    /**
     * Gets the OSGi FeedDataArchive reference.
     * @return FeedDataArchive reference.
     */
    public FeedDataArchive getFeedDataArchive() {
        OSGIRuntime osgiRuntime = EquinoxOSGIRuntimeImpl.getOSGIRuntime();
        return osgiRuntime.getService(FeedDataArchive.class, null);
    }
}
```

Version: c39f38a4b573af41ac7ea3ddb216baf5691a95e2

Parents:

1a77daa6fa02a93e348b43b37bca94908d4344c4
2535516907b2ccbda1cab8ef4e40ea929e778e26

Merge base:

c07a2d235b353a36a66d6d0befefb495ce0b22b6

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/bridge/PlotDataManager.java

Chunk 33: (combination/commentary, if statement, method invocation)

```
               }
<<<<<<< HEAD
//             if (!plot.plotAbstraction.getTimeAxis().isPinned()) {
                       dataSeries.get(feed).updateRegressionLine();
//             }
=======
               if (!plot.plotAbstraction.getTimeAxis().isPinned()) {
                       dataSeries.get(feed).updateRegressionLine();
               }
>>>>>>> 2535516907b2ccbda1cab8ef4e40ea929e778e26
               for(Entry<Long, Double> point : points.entrySet()) {
```

```
               }
               dataSeries.get(feed).updateRegressionLine();
               for(Entry<Long, Double> point : points.entrySet()) {
```

Version: 5cc4a8a18aa5b7e1848061ad40e4889ee6eb1cb2

Parents:

3d41ee6180e43e3c2c430c1db843557c764aea94
c07a2d235b353a36a66d6d0befefb495ce0b22b6

Merge base:

8c4392fcaa7d50b2f1f8b916f824cb34e4e5fd23

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/bridge/LegendEntry.java

Chunk 34: (combinaton/commentary, if statement, method invocation, variable)

```
                updateLabelFont();

<<<<<<< HEAD
                linePlot.setForeground(originalPlotLineColor.brighter());

                if(originalPlotLineStroke == null) {
=======
                // Highlight this entry on the plot.
                originalPlotLineColor = linePlot.getForeground();
                originalPlotLineStroke = linePlot.getStroke();


                linePlot.setForeground(originalPlotLineColor.brighter().brighter());
                BasicStroke stroke = (BasicStroke) originalPlotLineStroke;
                if(stroke == null) {
>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6
                        linePlot.setStroke(new
BasicStroke(PlotConstants.SELECTED_LINE_THICKNESS));
```

```
                updateLabelFont();


                // Highlight this entry on the plot.
                originalPlotLineColor = linePlot.getForeground();
                originalPlotLineStroke = linePlot.getStroke();


                linePlot.setForeground(originalPlotLineColor.brighter().brighter());
                if(originalPlotLineStroke == null) {
                        linePlot.setStroke(new
BasicStroke(PlotConstants.SELECTED_LINE_THICKNESS));
```

Chunk 35: (combination/commentary, if statement)

```
                                        .getLineJoin(),                 stroke.getMiterLimit(),
stroke.getDashArray(), stroke.getDashPhase()));
<<<<<<< HEAD
                } //Otherwise, it's a stroke we can't change (ie EMPTY_STROKE)

=======
                }
                if (regressionLine != null) {
                        originalRegressionLineStroke = regressionLine.getStroke();

        regressionLine.setForeground(originalPlotLineColor.brighter().brighter());
```

```
                        stroke = (BasicStroke) regressionLine.getStroke();
                        //TODO synch with plot thickness feature changes
                        if(stroke == null) {
                                regressionLine.setStroke(new
BasicStroke(PlotConstants.SLOPE_LINE_WIDTH*2,
                                 BasicStroke.CAP_BUTT,
                                 BasicStroke.JOIN_MITER,
                                 10.0f, PlotConstants.dash1, 0.0f));
                        } else {
                                regressionLine.setStroke(new
BasicStroke(PlotConstants.SLOPE_LINE_WIDTH*2,
                                 BasicStroke.CAP_BUTT,
                                 BasicStroke.JOIN_MITER,
                                 10.0f, PlotConstants.dash1, 0.0f));
                        }
                }

>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6
                this.setToolTipText(currentToolTipTxt);

        }
```

```
                                      .getLineJoin(),                stroke.getMiterLimit(),
stroke.getDashArray(), stroke.getDashPhase()));

                } //Otherwise, it's a stroke we can't change (ie EMPTY_STROKE)

                if (regressionLine != null) {
                        originalRegressionLineStroke = regressionLine.getStroke();

        regressionLine.setForeground(originalPlotLineColor.brighter().brighter());
                        Stroke stroke = (BasicStroke) regressionLine.getStroke();
                        //TODO synch with plot thickness feature changes
                        if(stroke == null) {
                                regressionLine.setStroke(new
BasicStroke(PlotConstants.SLOPE_LINE_WIDTH*2,
                                 BasicStroke.CAP_BUTT,
                                 BasicStroke.JOIN_MITER,
                                 10.0f, PlotConstants.dash1, 0.0f));
                        } else {
                                regressionLine.setStroke(new
BasicStroke(PlotConstants.SLOPE_LINE_WIDTH*2,
                                 BasicStroke.CAP_BUTT,
                                 BasicStroke.JOIN_MITER,
                                 10.0f, PlotConstants.dash1, 0.0f));
                        }
                }


                this.setToolTipText(currentToolTipTxt);

        }
```

Chunk 36: (combination/class declaration, commentary, method declaration)

```
<<<<<<< HEAD

        public void setLineSettings(LineSettings settings) {
                lineSettings = settings;
                updateLinePlotFromSettings();
        }

        public LineSettings getLineSettings() {
                return lineSettings;
        }

        private void updateLinePlotFromSettings() {
                /* Color */
                int index = lineSettings.getColorIndex();
                Color c = PlotLineColorPalette.getColor(index);
                setForeground(c);

                /* Thickness */
                Stroke s = linePlot.getStroke();
                if (s == null || s instanceof BasicStroke) {
                        int t = lineSettings.getThickness();
                        linePlot.setStroke(t == 1 ? null : new BasicStroke(t));
                        originalPlotLineStroke = linePlot.getStroke();
                } // We only want to modify known strokes

                /* Marker */
                if (linePlot.getPointIcon() != null) {
                        Shape shape = null;
                        if (lineSettings.getUseCharacter()) {
                                Graphics g = (Graphics) getGraphics();
                                if (g != null && g instanceof Graphics2D) {
                                        FontRenderContext                 frc                   =
((Graphics2D)g).getFontRenderContext();
                                        shape                                                   =
PlotLineShapePalette.getShape(lineSettings.getCharacter(), frc);
                                }
                        } else {
                                int marker = lineSettings.getMarker();
                                shape = PlotLineShapePalette.getShape(marker);
                        }
                        if (shape != null) {
                                linePlot.setPointIcon(new PlotMarkerIcon(shape));

                                baseDisplayNameLabel.setIcon(new  PlotMarkerIcon(shape,  false,
12, 12));
                        }
                }

                linePlot.repaint();
                repaint();
        }

        private class ShapeIcon implements Icon {


                @Override
                public int getIconHeight() {
                        //return linePlot != null && linePlot.getPointFill() != null ? 12 :
0;
                        return linePlot != null && linePlot.getPointIcon() != null ?
                                        12 : 0;
                }

                @Override
                public int getIconWidth() {
                        return linePlot != null && linePlot.getPointIcon() != null ?
                                        12 : 0;         }

                @Override
                public void paintIcon(Component c, Graphics g, int x, int y) {
                        if (linePlot != null && linePlot.getPointIcon() != null)
                                linePlot.getPointIcon().paintIcon(c,g,x+6,y+6);
//                      if (linePlot != null) {
//                              if (g instanceof Graphics2D) {
//                                      Graphics2D g2d = (Graphics2D) g;
//                                      Shape s = linePlot.getPointFill();
//                                      if (s != null) {
//                                              g2d.setColor(c.getForeground());
//                                              g2d.translate(6, 6);
//                                              g2d.fill(AffineTransform.getScaleInstance(0.75,
0.75).createTransformedShape(s));
//                                              g2d.translate(-6, -6);
//                                      }
//                              }
//                      }
                }

=======

        /** Get whether a regression line is displayed or not.
         * @return regressionLine
         */
        public boolean hasRegressionLine() {
                return hasRegressionLine;
        }

        /** Set whether a regression line is displayed or not.
         * @param regressionLine boolean indicator
         */
        public void setHasRegressionLine(boolean regressionLine) {
                this.hasRegressionLine = regressionLine;
        }

        /** Get the number of regression points to use.
         * @return numberRegressionPoints the number of regression points to use
         */
        public int getNumberRegressionPoints() {
                return numberRegressionPoints;
        }

        /** Set the number of regression points to use.
         * @param numberRegressionPoints
         */
        public void setNumberRegressionPoints(int numberRegressionPoints) {
                this.numberRegressionPoints = numberRegressionPoints;
        }

        /** Get the regression line for this legend entry.
         * @return regressionLine a LinearXYPlotLine
         */
        public LinearXYPlotLine getRegressionLine() {
                return regressionLine;
        }

        /** Set the regression line for this legend entry.
         * @param regressionLine a LinearXYPlotLine
         */
        public void setRegressionLine(LinearXYPlotLine regressionLine) {
                this.regressionLine = regressionLine;
                if (regressionLine != null)
                        regressionLine.setForeground(foregroundColor);
>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6
        }
}
```

```
        public void setLineSettings(LineSettings settings) {
                lineSettings = settings;
                updateLinePlotFromSettings();
        }

        public LineSettings getLineSettings() {
                return lineSettings;
        }

        private void updateLinePlotFromSettings() {
                /* Color */
                int index = lineSettings.getColorIndex();
                Color c = PlotLineColorPalette.getColor(index);
                setForeground(c);

                /* Thickness */
                Stroke s = linePlot.getStroke();
                if (s == null || s instanceof BasicStroke) {
                        int t = lineSettings.getThickness();
                        linePlot.setStroke(t == 1 ? null : new BasicStroke(t));
                        originalPlotLineStroke = linePlot.getStroke();
                } // We only want to modify known strokes

                /* Marker */
                if (linePlot.getPointIcon() != null) {
                        Shape shape = null;
                        if (lineSettings.getUseCharacter()) {
                                Graphics g = (Graphics) getGraphics();
                                if (g != null && g instanceof Graphics2D) {
                                        FontRenderContext                 frc                    =
((Graphics2D)g).getFontRenderContext();
                                        shape                                                    =
PlotLineShapePalette.getShape(lineSettings.getCharacter(), frc);
                                }
                        } else {
                                int marker = lineSettings.getMarker();
                                shape = PlotLineShapePalette.getShape(marker);
                        }
                        if (shape != null) {
                                linePlot.setPointIcon(new PlotMarkerIcon(shape));

                                baseDisplayNameLabel.setIcon(new  PlotMarkerIcon(shape,  false,
12, 12));
                        }
                }

                linePlot.repaint();
                repaint();
        }

        private class ShapeIcon implements Icon {


                @Override
                public int getIconHeight() {
                        //return linePlot != null  && linePlot.getPointFill()  != null ? 12 :
0;
```

```
                        return linePlot != null && linePlot.getPointIcon() != null ?
                                        12 : 0;
                }

                @Override
                public int getIconWidth() {
                        return linePlot != null && linePlot.getPointIcon() != null ?
                                        12 : 0;         }

                @Override
                public void paintIcon(Component c, Graphics g, int x, int y) {
                        if (linePlot != null && linePlot.getPointIcon() != null)
                                linePlot.getPointIcon().paintIcon(c,g,x+6,y+6);
                }
        }


        /** Get whether a regression line is displayed or not.
         * @return regressionLine
         */
        public boolean hasRegressionLine() {
                return hasRegressionLine;
        }

        /** Set whether a regression line is displayed or not.
         * @param regressionLine boolean indicator
         */
        public void setHasRegressionLine(boolean regressionLine) {
                this.hasRegressionLine = regressionLine;
        }

        /** Get the number of regression points to use.
         * @return numberRegressionPoints the number of regression points to use
         */
        public int getNumberRegressionPoints() {
                return numberRegressionPoints;
        }

        /** Set the number of regression points to use.
         * @param numberRegressionPoints
         */
        public void setNumberRegressionPoints(int numberRegressionPoints) {
                this.numberRegressionPoints = numberRegressionPoints;
        }

        /** Get the regression line for this legend entry.
         * @return regressionLine a LinearXYPlotLine
         */
        public LinearXYPlotLine getRegressionLine() {
                return regressionLine;
        }

        /** Set the regression line for this legend entry.
         * @param regressionLine a LinearXYPlotLine
         */
        public void setRegressionLine(LinearXYPlotLine regressionLine) {
                this.regressionLine = regressionLine;
                if (regressionLine != null)
                        regressionLine.setForeground(foregroundColor);
        }
}
```

Developers are adding methods in parallel. The solution is the set of all methods.

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/bridge/PlotDataSeries.java

Chunk 37: (combination/import declaration)

```
package gov.nasa.arc.mct.fastplot.bridge;

<<<<<<< HEAD
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.AxisOrientationSetting;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.PlotLineConnectionType;

import java.awt.Color;
import java.awt.Polygon;
import java.awt.Shape;
import java.awt.Stroke;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import plotter.DoubleData;
import plotter.xy.CompressingXYDataset;
import plotter.xy.CompressingXYDataset.MinMaxChangeListener;
import plotter.xy.DefaultCompressor;
import plotter.xy.LinearXYPlotLine;
import plotter.xy.LinearXYPlotLine.LineMode;
import plotter.xy.XYDimension;
=======
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.AxisOrientationSetting;

import java.awt.BasicStroke;
import java.awt.Color;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import plotter.DoubleData;
import plotter.xy.CompressingXYDataset;
import plotter.xy.CompressingXYDataset.MinMaxChangeListener;
import plotter.xy.DefaultCompressor;
import plotter.xy.LinearXYPlotLine;
import plotter.xy.LinearXYPlotLine.LineMode;
import plotter.xy.XYDimension;
>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6

/**
```

```
package gov.nasa.arc.mct.fastplot.bridge;


import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.AxisOrientationSetting;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants.PlotLineConnectionType;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Polygon;
import java.awt.Shape;
import java.awt.Stroke;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import plotter.DoubleData;
import plotter.xy.CompressingXYDataset;
import plotter.xy.CompressingXYDataset.MinMaxChangeListener;
import plotter.xy.DefaultCompressor;
import plotter.xy.LinearXYPlotLine;
import plotter.xy.LinearXYPlotLine.LineMode;
import plotter.xy.XYDimension;


/**
```

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/bridge/PlotView.java

Chunk 38: (version 1/ commentary, method declaration)

```
                return colorAssignments;
        }

<<<<<<< HEAD
        /**
         * Get per-line settings currently in use for this stack of plots.
         * Each element of the returned list corresponds,
         * in order, to the sub-plots displayed, and maps subscription ID to a
         * LineSettings object describing how its plot line should be drawn.
         * @return a list of subscription->setting mappings for this plot
         */
        public List<Map<String, LineSettings>> getLineSettings() {
                List<Map<String,LineSettings>>        settingsAssignments        =        new
ArrayList<Map<String,LineSettings>>();
                for (int subPlotIndex = 0; subPlotIndex < subPlots.size(); subPlotIndex++) {
                        Map<String,    LineSettings>    settingsMap    =    new    HashMap<String,
LineSettings>();
                        settingsAssignments.add(settingsMap);
                        PlotterPlot plot = (PlotterPlot) subPlots.get(subPlotIndex);
                        for        (Entry<String,        PlotDataSeries>        entry        :
plot.plotDataManager.dataSeries.entrySet()) {
                                settingsMap.put(entry.getKey(),
entry.getValue().legendEntry.getLineSettings());
                        }
                }
                return settingsAssignments;
        }
=======
>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6


        /**
```

```
                return colorAssignments;
        }

        /**
         * Get per-line settings currently in use for this stack of plots.
         * Each element of the returned list corresponds,
         * in order, to the sub-plots displayed, and maps subscription ID to a
         * LineSettings object describing how its plot line should be drawn.
         * @return a list of subscription->setting mappings for this plot
         */
        public List<Map<String, LineSettings>> getLineSettings() {
                List<Map<String,LineSettings>>        settingsAssignments        =        new
ArrayList<Map<String,LineSettings>>();
                for (int subPlotIndex = 0; subPlotIndex < subPlots.size(); subPlotIndex++) {
                        Map<String,    LineSettings>    settingsMap    =    new    HashMap<String,
LineSettings>();
```

```
                        settingsAssignments.add(settingsMap);
                        PlotterPlot plot = (PlotterPlot) subPlots.get(subPlotIndex);
                        for        (Entry<String,        PlotDataSeries>        entry        :
plot.plotDataManager.dataSeries.entrySet()) {
                                settingsMap.put(entry.getKey(),
entry.getValue().legendEntry.getLineSettings());
                        }
                }
                return settingsAssignments;
        }


        /**
```

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/view/LegendEntryPopupMenuFactory.java

Chunk 39: (combination/import declaration)

```
package gov.nasa.arc.mct.fastplot.view;

<<<<<<< HEAD
import gov.nasa.arc.mct.fastplot.bridge.LegendEntry;
import gov.nasa.arc.mct.fastplot.bridge.PlotAbstraction.LineSettings;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants;
import gov.nasa.arc.mct.fastplot.bridge.PlotLineColorPalette;
import gov.nasa.arc.mct.fastplot.bridge.PlotLineShapePalette;
import gov.nasa.arc.mct.fastplot.bridge.PlotMarkerIcon;

import java.awt.Color;
import java.awt.Component;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.font.FontRenderContext;
import java.util.ResourceBundle;

import javax.swing.BorderFactory;
import javax.swing.Icon;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JTextField;
import javax.swing.SpringLayout;
import javax.swing.SwingUtilities;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.text.AbstractDocument;
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.Document;
import javax.swing.text.DocumentFilter;
=======
import gov.nasa.arc.mct.fastplot.bridge.LegendEntry;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants;
import gov.nasa.arc.mct.fastplot.bridge.PlotLineColorPalette;

import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusListener;
import java.awt.event.FocusEvent;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.ResourceBundle;

import javax.swing.AbstractButton;
import javax.swing.Icon;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JFormattedTextField;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JSpinner;
import javax.swing.JSpinner.NumberEditor;
import javax.swing.SpinnerModel;
import javax.swing.SpinnerNumberModel;
import javax.swing.SwingUtilities;
import javax.swing.border.EmptyBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.event.MenuKeyEvent;
import javax.swing.event.MenuKeyListener;
>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6

/**
```

```
package gov.nasa.arc.mct.fastplot.view;


import gov.nasa.arc.mct.fastplot.bridge.LegendEntry;
import gov.nasa.arc.mct.fastplot.bridge.PlotAbstraction.LineSettings;
import gov.nasa.arc.mct.fastplot.bridge.PlotConstants;
import gov.nasa.arc.mct.fastplot.bridge.PlotLineColorPalette;
import gov.nasa.arc.mct.fastplot.bridge.PlotLineShapePalette;
import gov.nasa.arc.mct.fastplot.bridge.PlotMarkerIcon;

import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.font.FontRenderContext;
import java.util.ResourceBundle;
import javax.swing.AbstractButton;
import javax.swing.BorderFactory;
import javax.swing.Icon;
import javax.swing.JButton;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JDialog;
import javax.swing.JFormattedTextField;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JSpinner;
import javax.swing.JSpinner.NumberEditor;
import javax.swing.JTextField;
import javax.swing.SpinnerModel;
import javax.swing.SpinnerNumberModel;
import javax.swing.SpringLayout;
import javax.swing.SwingUtilities;
import javax.swing.border.EmptyBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.event.MenuKeyEvent;
import javax.swing.event.MenuKeyListener;
import javax.swing.text.AbstractDocument;
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.Document;
import javax.swing.text.DocumentFilter;

/**
```

Chunk 40: (new code/ annotation, commentary, if statement, method declaration, method invocation, variable

```
<<<<<<< HEAD
                      if (name.isEmpty()) name = legendEntry.getFullBaseDisplayName();

                      if (!manifestation.isLocked()) {

                              final LineSettings settings = legendEntry.getLineSettings();

                              // Color submenu
                              String                      subMenuText                      =
String.format(BUNDLE.getString("SelectColor.label"), name);
                              JMenu subMenu = new JMenu(subMenuText);
                              Color currentColor = legendEntry.getForeground();
=======
                      if (name.isEmpty()) name = legendEntry.getFullBaseDisplayName();

                      String                         subMenuText1                        =
String.format(BUNDLE.getString("SelectColor.label"),
                                           name);
                      String                         subMenuText2                        =
String.format(BUNDLE.getString("RegressionPointsLabel"),
                    name);
                      final JMenu subMenu1 = new JMenu(subMenuText1);
                      final       JMenuItem       regressionLineCheckBox       =       new
JCheckBoxMenuItem(BUNDLE.getString("RegressionLineLabel"),false);
                      final JMenu regressionMenu = new JMenu(subMenuText2);


                      SpinnerModel             pointsModel             =             new
SpinnerNumberModel(legendEntry.getNumberRegressionPoints(), 2, 100, 1);
                      final JSpinner spinner = new JSpinner(pointsModel);
                      spinner.setPreferredSize(new Dimension(50, 20));
                      spinner.setBorder(new EmptyBorder(2,2,2,2));
                      spinner.addChangeListener(new ChangeListener() {

                              @Override
                              public void stateChanged(ChangeEvent e) {

       legendEntry.setNumberRegressionPoints(Integer.parseInt(((JSpinner)e.getSource()).get
Value().toString()));
                                      manifestation.setupRegressionLines();
                              }

                      });

                       final JFormattedTextField myTextField = ((NumberEditor) spinner
                                      .getEditor()).getTextField();

                      spinner.addKeyListener(new KeyListener() {

                              @Override
                              public void keyTyped(KeyEvent e) {
                                      if ( ! (e.getKeyChar() == KeyEvent.CHAR_UNDEFINED) &&
                                                      (e.getKeyCode()                     ==
KeyEvent.VK_UNDEFINED) &&
                                                      // Apparently, backspace has a key char
(although it should not)
                                                      (e.getKeyChar() == '0' ||
                                                       e.getKeyChar() == '1' ||
                                                       e.getKeyChar() == '2' ||
                                                       e.getKeyChar() == '3' ||
                                                       e.getKeyChar() == '4' ||
                                                       e.getKeyChar() == '5' ||
                                                       e.getKeyChar() == '6' ||
                                                       e.getKeyChar() == '7' ||
                                                       e.getKeyChar() == '8' ||
                                                       e.getKeyChar() == '9'
                                                                      ) &&
                                                      Integer.valueOf(myTextField.getValue()
+ String.valueOf(e.getKeyChar())).compareTo((Integer)
                                                                      ((SpinnerNumberModel)
spinner.getModel()).getMinimum()) > 0 &&
                                                      Integer.valueOf(myTextField.getValue()
+ String.valueOf(e.getKeyChar())).compareTo((Integer)
                                                                      ((SpinnerNumberModel)
spinner.getModel()).getMaximum()) < 0 ) {
                                              myTextField.setText(myTextField.getValue()    +
String.valueOf(e.getKeyChar()));

                                      }
                              }

                              @Override
                              public void keyPressed(KeyEvent e) {
                                      if (e.getKeyCode() == KeyEvent.VK_DELETE ) {
                                              ((NumberEditor)
spinner.getEditor()).getTextField().setText("");
                                      }
                                      myTextField.grabFocus();
                              }

                              @Override
                              public void keyReleased(KeyEvent e) {
                              }

                       });

                        myTextField.addFocusListener(new FocusListener()
                                   {
                               @Override
                               public void focusGained(FocusEvent e) {
                                       SwingUtilities.invokeLater(new Runnable() {
                                          public void run() {
                                             myTextField.selectAll();
                                          }
                                   });
                               }

                              @Override
                              public void focusLost(java.awt.event.FocusEvent e) {
                              }
                       });

                        final      NumberEditor      numberEditor      =      (NumberEditor)
spinner.getEditor();

                        numberEditor.addKeyListener(new KeyListener() {

                              @Override
                              public void keyTyped(KeyEvent e) {
                              }

                              @Override
                              public void keyPressed(KeyEvent e) {
                                      if (e.getKeyCode() == KeyEvent.VK_LEFT &&

        numberEditor.getTextField().getCaretPosition() == 0) {
                                              regressionMenu.setSelected(true);
                                      }
                              }

                              @Override
                              public void keyReleased(KeyEvent e) {
                              }
                       });

                        myTextField.addKeyListener(new KeyListener() {

                              @Override
                              public void keyTyped(KeyEvent e) {
                              }

                              @Override
                              public void keyPressed(KeyEvent e) {
                                      if (e.getKeyCode() == KeyEvent.VK_LEFT &&

        numberEditor.getTextField().getCaretPosition() == 0) {
```

```
                                                regressionMenu.setSelected(true);
                                                regressionMenu.grabFocus();
                                                ((JPopupMenu)
spinner.getParent()).setSelected(regressionMenu);
                                        }
                                }

                                @Override
                                public void keyReleased(KeyEvent e) {
                                }

                        });

                        regressionMenu.addMenuKeyListener(new MenuKeyListener() {

                                @Override
                                public void menuKeyTyped(MenuKeyEvent e) {
                                }

                                @Override
                                public void menuKeyPressed(MenuKeyEvent e) {
                                        if (e.getKeyCode() == KeyEvent.VK_RIGHT ) {
                                                spinner.setVisible(true);
                                                spinner.requestFocus();
                                                ((NumberEditor)
spinner.getEditor()).grabFocus();
                                        }
                                }

                                @Override
                                public void menuKeyReleased(MenuKeyEvent e) {
                                }

                        });


                        if (!manifestation.isLocked()) {
>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6
                                for        (int        i        =        0;        i        <
PlotConstants.MAX_NUMBER_OF_DATA_ITEMS_ON_A_PLOT; i++) {
```

```
                String name = legendEntry.getComputedBaseDisplayName();

                        if (name.isEmpty()) name = legendEntry.getFullBaseDisplayName();


                        String                          subMenuText2                          =
String.format(BUNDLE.getString("RegressionPointsLabel"),
                    name);

                        final       JMenuItem       regressionLineCheckBox       =       new
JCheckBoxMenuItem(BUNDLE.getString("RegressionLineLabel"),false);
                        final JMenu regressionMenu = new JMenu(subMenuText2);


                        SpinnerModel              pointsModel              =              new
SpinnerNumberModel(legendEntry.getNumberRegressionPoints(), 2, 100, 1);
                        final JSpinner spinner = new JSpinner(pointsModel);
                        spinner.setPreferredSize(new Dimension(50, 20));
                        spinner.setBorder(new EmptyBorder(2,2,2,2));
                        spinner.addChangeListener(new ChangeListener() {
                              @Override
                              public void stateChanged(ChangeEvent e) {

        legendEntry.setNumberRegressionPoints(Integer.parseInt(((JSpinner)e.getSource()).get
Value().toString()));
                                      manifestation.setupRegressionLines();
                              }

                       });

                        final JFormattedTextField myTextField = ((NumberEditor) spinner
                                      .getEditor()).getTextField();

                       spinner.addKeyListener(new KeyListener() {

                              @Override
                              public void keyTyped(KeyEvent e) {
                                      if ( ! (e.getKeyChar() == KeyEvent.CHAR_UNDEFINED) &&
                                                     (e.getKeyCode()                      ==
KeyEvent.VK_UNDEFINED) &&
                                                     // Apparently, backspace has a key char
(although it should not)
                                                     (e.getKeyChar() == '0' ||
                                                      e.getKeyChar() == '1' ||
                                                      e.getKeyChar() == '2' ||
                                                      e.getKeyChar() == '3' ||
                                                      e.getKeyChar() == '4' ||
                                                      e.getKeyChar() == '5' ||
                                                      e.getKeyChar() == '6' ||
                                                      e.getKeyChar() == '7' ||
                                                      e.getKeyChar() == '8' ||
                                                      e.getKeyChar() == '9'
                                                                    ) &&
                                                     Integer.valueOf(myTextField.getValue()
+ String.valueOf(e.getKeyChar())).compareTo((Integer)
                                                                    ((SpinnerNumberModel)
spinner.getModel()).getMinimum()) > 0 &&
                                                     Integer.valueOf(myTextField.getValue()
+ String.valueOf(e.getKeyChar())).compareTo((Integer)
                                                                    ((SpinnerNumberModel)
spinner.getModel()).getMaximum()) < 0 ) {
                                             myTextField.setText(myTextField.getValue()    +
String.valueOf(e.getKeyChar()));

                                      }
                              }

                              @Override
                              public void keyPressed(KeyEvent e) {
                                      if (e.getKeyCode() == KeyEvent.VK_DELETE ) {
                                             ((NumberEditor)
spinner.getEditor()).getTextField().setText("");
                                      }
                                      myTextField.grabFocus();
                              }

                              @Override
                              public void keyReleased(KeyEvent e) {
                              }

                       });
```

```
                        myTextField.addFocusListener(new FocusListener()
                                   {
                               @Override
                               public void focusGained(FocusEvent e) {
                                       SwingUtilities.invokeLater(new Runnable() {
                                          public void run() {
                                             myTextField.selectAll();
                                          }
                                   });
                               }

                              @Override
                              public void focusLost(java.awt.event.FocusEvent e) {
                              }
                       });

                        final NumberEditor numberEditor = (NumberEditor)
spinner.getEditor();

                        numberEditor.addKeyListener(new KeyListener() {

                              @Override
                              public void keyTyped(KeyEvent e) {
                              }

                              @Override
                              public void keyPressed(KeyEvent e) {
                                      if (e.getKeyCode() == KeyEvent.VK_LEFT &&

       numberEditor.getTextField().getCaretPosition() == 0) {
                                             regressionMenu.setSelected(true);
                                      }
                              }

                              @Override
                              public void keyReleased(KeyEvent e) {
                              }
                       });

                        myTextField.addKeyListener(new KeyListener() {

                              @Override
                              public void keyTyped(KeyEvent e) {
                              }

                              @Override
                              public void keyPressed(KeyEvent e) {
                                      if (e.getKeyCode() == KeyEvent.VK_LEFT &&

       numberEditor.getTextField().getCaretPosition() == 0) {
                                             regressionMenu.setSelected(true);
                                             regressionMenu.grabFocus();
                                             ((JPopupMenu)
spinner.getParent()).setSelected(regressionMenu);
                                      }
                              }

                              @Override
                              public void keyReleased(KeyEvent e) {
                              }
```

```
            });

            regressionMenu.addMenuKeyListener(new MenuKeyListener() {

                @Override
                public void menuKeyTyped(MenuKeyEvent e) {
                }

                @Override
                public void menuKeyPressed(MenuKeyEvent e) {
                    if (e.getKeyCode() == KeyEvent.VK_RIGHT ) {
                        spinner.setVisible(true);
                        spinner.requestFocus();
                        ((NumberEditor)
spinner.getEditor()).grabFocus();
                    }
                }

                @Override
                public void menuKeyReleased(MenuKeyEvent e) {
                }

            });


            if (!manifestation.isLocked()) {

                final LineSettings settings = legendEntry.getLineSettings();

                // Color submenu
                String                    subMenuText                    =
String.format(BUNDLE.getString("SelectColor.label"), name);
                JMenu subMenu = new JMenu(subMenuText);
                Color currentColor = legendEntry.getForeground();

                for        (int        i        =        0;        i        <
PlotConstants.MAX_NUMBER_OF_DATA_ITEMS_ON_A_PLOT; i++) {
```

Case 41: (new code/annotation, commentary, for statement, if statement, method declaration, method invocation)

```
                }
<<<<<<< HEAD
                add(subMenu);

                // Thickness submenu
                subMenuText                                                  =
String.format(BUNDLE.getString("SelectThickness.label"), name);
                subMenu = new JMenu(subMenuText);
                for (int i = 1; i <= PlotConstants.MAX_LINE_THICKNESS; i++) {
                    JMenuItem item = new JRadioButtonMenuItem("" + i,
                                (settings.getThickness() == i));
                    final int thickness = i;
                    item.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent e) {

                            settings.setThickness(thickness);
                            legendEntry.setLineSettings(settings);

    manifestation.persistPlotLineSettings();
```

```
                        }
                    });
                    subMenu.add(item);

                }
                add(subMenu);

                // Marker submenu
                if (manifestation.getPlot() != null &&

    manifestation.getPlot().getPlotLineDraw().drawMarkers()) {
                    subMenuText                                         =
String.format(BUNDLE.getString("SelectMarker.label"), name);
                    subMenu = new JMenu(subMenuText);
                    for       (int       i       =       0;       i       <
PlotConstants.MAX_NUMBER_OF_DATA_ITEMS_ON_A_PLOT; i++) {
                        JMenuItem item = new JRadioButtonMenuItem("",
                                new
PlotMarkerIcon(PlotLineShapePalette.getShape(i), false),
                                (settings.getMarker()  ==  i  &&
!settings.getUseCharacter()));

    item.setForeground(legendEntry.getForeground());
                        final int marker = i;
                        item.addActionListener(new ActionListener() {
                            @Override
                            public void actionPerformed(ActionEvent
e) {
                                settings.setMarker(marker);
                                settings.setUseCharacter(false);

    legendEntry.setLineSettings(settings);

    manifestation.persistPlotLineSettings();
                            }
                        });
                        subMenu.add(item);
                    }
                    JMenuItem            other            =            new
JRadioButtonMenuItem(BUNDLE.getString("SelectCharacter.label"),
                            settings.getUseCharacter());
                    if (!settings.getCharacter().isEmpty()) {
                        FontRenderContext     frc     =     ((Graphics2D)
manifestation.getGraphics()).getFontRenderContext();
                        other.setIcon(new PlotMarkerIcon(

    PlotLineShapePalette.getShape(settings.getCharacter(), frc),

    PlotLineColorPalette.getColor(settings.getColorIndex()),
                                false));
                    }
                    other.addActionListener( new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                            final   CharacterDialog   dialog   =   new
CharacterDialog();

    dialog.setInitialString(settings.getCharacter());
                            dialog.ok.addActionListener(        new
ActionListener() {
                                @Override
```

```
                                                                public                   void
actionPerformed(ActionEvent arg0) {

        settings.setCharacter(dialog.field.getText().trim());

        settings.setUseCharacter(true);

        legendEntry.setLineSettings(settings);

        manifestation.persistPlotLineSettings();
                                                                }

                                                        });
                                                        dialog.setVisible(true);
                                                }
                                        });
                                        subMenu.add(other);
                                        add(subMenu);
                                }

=======

                                add(subMenu1);
                                addSeparator();

                                regressionLineCheckBox.addActionListener(new  ActionListener()
{

                                        @Override
                                        public void actionPerformed(ActionEvent e) {
                                                AbstractButton         abstractButton         =
(AbstractButton) e.getSource();
                                                if (abstractButton.getModel().isSelected()) {
                                                        legendEntry.setHasRegressionLine(true);
                                                } else {

        legendEntry.setHasRegressionLine(false);
                                                }
                                                manifestation.setupRegressionLines();

                                        }

                                });
                                if (legendEntry.hasRegressionLine()) {
                                        regressionLineCheckBox.setSelected(true);
                                } else {
                                        regressionLineCheckBox.setSelected(false);
                                }
                                add(regressionLineCheckBox);
                                regressionMenu.add(spinner);
                                add(regressionMenu);
>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6
                        }

                }
```

```
                                }

                                add(subMenu);

                                // Thickness submenu
                                subMenuText                                          =
String.format(BUNDLE.getString("SelectThickness.label"), name);
                                subMenu = new JMenu(subMenuText);
                                for (int i = 1; i <= PlotConstants.MAX_LINE_THICKNESS; i++) {
                                        JMenuItem item = new JRadioButtonMenuItem("" + i,
                                                        (settings.getThickness() == i));
                                        final int thickness = i;
                                        item.addActionListener(new ActionListener() {
                                                @Override
                                                public void actionPerformed(ActionEvent e) {

                                                        settings.setThickness(thickness);
                                                        legendEntry.setLineSettings(settings);

        manifestation.persistPlotLineSettings();
                                                }
                                        });
                                        subMenu.add(item);

                                }
                                add(subMenu);

                                // Marker submenu
                                if (manifestation.getPlot() != null &&

        manifestation.getPlot().getPlotLineDraw().drawMarkers()) {
                                        subMenuText                                  =
String.format(BUNDLE.getString("SelectMarker.label"), name);
                                        subMenu = new JMenu(subMenuText);
                                        for       (int       i       =       0;       i       <
PlotConstants.MAX_NUMBER_OF_DATA_ITEMS_ON_A_PLOT; i++) {
                                                JMenuItem item = new JRadioButtonMenuItem("",
                                                                new
PlotMarkerIcon(PlotLineShapePalette.getShape(i), false),
                                                                (settings.getMarker()  ==  i  &&
!settings.getUseCharacter()));

        item.setForeground(legendEntry.getForeground());
                                                final int marker = i;
                                                item.addActionListener(new ActionListener() {
                                                        @Override
                                                        public  void  actionPerformed(ActionEvent
e) {
                                                                settings.setMarker(marker);
                                                                settings.setUseCharacter(false);

        legendEntry.setLineSettings(settings);

        manifestation.persistPlotLineSettings();
                                                        }
                                                });
                                                subMenu.add(item);
                                        }
                                        JMenuItem          other          =          new
JRadioButtonMenuItem(BUNDLE.getString("SelectCharacter.label"),
                                                        settings.getUseCharacter());
                                        if (!settings.getCharacter().isEmpty()) {
                                                FontRenderContext    frc    =    ((Graphics2D)
manifestation.getGraphics()).getFontRenderContext();
                                                other.setIcon(new PlotMarkerIcon(

        PlotLineShapePalette.getShape(settings.getCharacter(), frc),
        PlotLineColorPalette.getColor(settings.getColorIndex()),
                                                                false));
                                        }
                                        other.addActionListener( new ActionListener() {
                                                @Override
                                                public void actionPerformed(ActionEvent arg0) {
                                                        final   CharacterDialog   dialog   =   new
CharacterDialog();

        dialog.setInitialString(settings.getCharacter());
                                                        dialog.ok.addActionListener(         new
ActionListener() {
                                                                @Override
                                                                public                      void
actionPerformed(ActionEvent arg0) {

        settings.setCharacter(dialog.field.getText().trim());

        settings.setUseCharacter(true);

        legendEntry.setLineSettings(settings);

        manifestation.persistPlotLineSettings();
                                                                }

                                                        });
                                                        dialog.setVisible(true);
                                                }
                                        });
                                        subMenu.add(other);
                                        add(subMenu);
                                }

                                addSeparator();

                                regressionLineCheckBox.addActionListener(new  ActionListener()
{

                                        @Override
                                        public void actionPerformed(ActionEvent e) {
                                                AbstractButton        abstractButton        =
(AbstractButton) e.getSource();
                                                if (abstractButton.getModel().isSelected()) {
                                                        legendEntry.setHasRegressionLine(true);
                                                } else {

        legendEntry.setHasRegressionLine(false);
                                                }
                                                manifestation.setupRegressionLines();

                                        }

                                });
                                if (legendEntry.hasRegressionLine()) {
                                        regressionLineCheckBox.setSelected(true);
                                } else {
                                        regressionLineCheckBox.setSelected(false);
                                }
                                add(regressionLineCheckBox);
                                regressionMenu.add(spinner);
                                add(regressionMenu);
```

```
                        }

                }
```

The developers changed the artifact in parallel. The result is a merge of those conflicting areas, but it is difficult to solve without understand the software.

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/view/PlotPersistanceHandler.java

Case 42: (combination/ commentary, if statement, method declaration, method invocation, method signature, return statement, while statement)

```
                                }
                        }
                }

<<<<<<< HEAD
                return lineSettingAssignments;
        }

        private List<Map<String, Integer>> getColorAssignments() {
                String                         colorAssignmentString                          =
plotViewManifestation.getViewProperties().getProperty(PlotConstants.COLOR_ASSIGNMENTS,
String.class);
                List<Map<String,  Integer>>  colorAssignments  =  new  ArrayList<Map<String,
Integer>>();
                if (colorAssignmentString != null) {
                        StringTokenizer          allAssignmentTokens           =          new
StringTokenizer(colorAssignmentString, "\n");

                        while (allAssignmentTokens.hasMoreTokens()) {
                                StringTokenizer       colorAssignmentTokens       =       new
StringTokenizer(allAssignmentTokens.nextToken(), "\t");

                                Map<String,   Integer>   subPlotMap   =   new   HashMap<String,
Integer>();
                                colorAssignments.add(subPlotMap);
                                while (colorAssignmentTokens.hasMoreTokens()) {

                                        String dataSet   = colorAssignmentTokens.nextToken();
                                        int          colorIndex                              =
Integer.parseInt(colorAssignmentTokens.nextToken());

                                        subPlotMap.put(dataSet, colorIndex);
                                }
                        }
                }
                return colorAssignments;
        }
=======
                colorAssignments = new ArrayList<Map<String, Integer>>();
                while (allAssignmentTokens.hasMoreTokens()) {
                        StringTokenizer          colorAssignmentTokens          =          new
StringTokenizer(allAssignmentTokens.nextToken(), "\t");

                        Map<String, Integer> subPlotMap = new HashMap<String, Integer>();
                        colorAssignments.add(subPlotMap);
                        while (colorAssignmentTokens.hasMoreTokens()) {

                                String dataSet   = colorAssignmentTokens.nextToken();
                                int colorIndex =
Integer.parseInt(colorAssignmentTokens.nextToken());

                                subPlotMap.put(dataSet, colorIndex);
                        }
                }

                return colorAssignments;
        }

        /**
         * Retrieve persisted regression point assignments. Each element of the returned
list
         * corresponds, in order, to the sub-plots displayed, and maps subscription ID to
         * the number of regression points assigned and whether a regression line is
displayed.
         * The form of the values in the map is false|true:number of points
         * @return the persisted regression point assignments
         */
        public List<Map<String, String>> loadRegressionSettingsFromPersistence() {
                List<Map<String, String>> pointAssignments;

                String pointAssignmentString =
plotViewManifestation.getViewProperties().getProperty(PlotConstants.REGRESSION_LINE,
String.class);

                if (pointAssignmentString == null) return null;

                StringTokenizer allAssignmentTokens = new
StringTokenizer(pointAssignmentString, "\n");

                pointAssignments = new ArrayList<Map<String, String>>();
                while (allAssignmentTokens.hasMoreTokens()) {
                        StringTokenizer pointAssignmentTokens = new
StringTokenizer(allAssignmentTokens.nextToken(), "\t");

                        Map<String, String> subPlotMap = new HashMap<String, String>();
                        pointAssignments.add(subPlotMap);
                        while (pointAssignmentTokens.hasMoreTokens()) {

                                String dataSet   = pointAssignmentTokens.nextToken();
                                subPlotMap.put(dataSet, pointAssignmentTokens.nextToken());
                        }
                }

                return pointAssignments;
        }

        /**
         * Persist regression point assignments. Each element of the supplied list
corresponds,
         * in order, to the sub-plots displayed, and maps subscription ID to the number of
         * regression points assigned.
         * @param numberOfRegressionPoints the regression point assignments to persist.
         */
        public void persistRegressionSettings(List<Map<String, String>>
numberOfRegressionPoints) {
                /* Separate, because these are changed in a very different way from control
panel settings...
                 * But should these really be separate at this level? */
                ExtendedProperties viewProperties =
plotViewManifestation.getViewProperties();

                StringBuilder pointAssignmentBuilder = new StringBuilder();
                for (Map<String, String> subPlotMap : numberOfRegressionPoints) {
                        for (Entry<String,String> entry : subPlotMap.entrySet()) {
                                pointAssignmentBuilder.append(entry.getKey());
                                pointAssignmentBuilder.append('\t');
                                pointAssignmentBuilder.append(entry.getValue());
                                pointAssignmentBuilder.append('\t');
                        }
                        pointAssignmentBuilder.append('\n');
                }
                viewProperties.setProperty(PlotConstants.REGRESSION_LINE, "" +
                pointAssignmentBuilder.toString());

                if (plotViewManifestation.getManifestedComponent() != null) {
                        plotViewManifestation.getManifestedComponent().save();
                }
        }

>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6
```

```
/**
        * Retrieve persisted per-line plot settings (feed color assignments, line
thicknesses, etc).
        * Each element of the returned list corresponds, in order, to the sub-plots
displayed,
        * and maps subscription ID to a LineSettings object describing how the line is to
be displayed.
        * @return the persisted line settings
        */
        public List<Map<String, LineSettings>> loadLineSettingsFromPersistence() {
                List<Map<String, LineSettings>> lineSettingAssignments =
                        new ArrayList<Map<String, LineSettings>>();

                String lineSettings =
plotViewManifestation.getViewProperties().getProperty(PlotConstants.LINE_SETTINGS,
String.class);
                if (lineSettings != null) {
                        for (String plot : lineSettings.split("\n")) {
                                Map<String, LineSettings> settingsMap = new HashMap<String,
LineSettings>();

                                for (String line : plot.split("\t")) {
                                        LineSettings settings = new LineSettings();

                                        String[] tokens = line.split(" ");
                                        try {
                                                settings.setIdentifier   (tokens[0]);
                                                settings.setColorIndex
(Integer.parseInt(tokens[1]));
                                                settings.setThickness
(Integer.parseInt(tokens[2]));
                                                settings.setMarker
(Integer.parseInt(tokens[3]));
                                                settings.setCharacter    (tokens[4]);
                                                settings.setUseCharacter
(Boolean.parseBoolean(tokens[5]));
                                        } catch (Exception e) {
```

```
                        logger.error("Could not parse plot line
settings from persistence", e);
                    }

                    if (!settings.getIdentifier().isEmpty()) {
                        settingsMap.put(settings.getIdentifier(),
settings);
                    }
                }

                lineSettingAssignments.add(settingsMap);
            }
        }

        /* Merge in color assignments, if specified */
        List<Map<String, Integer>> colorAssignments = getColorAssignments();
        for (int i = 0; i < Math.min(colorAssignments.size(),
lineSettingAssignments.size()); i++) {
            Map<String, LineSettings> settingsMap =
lineSettingAssignments.get(i);
            for (Entry<String, Integer> e : colorAssignments.get(i).entrySet()) {
                if (!settingsMap.containsKey(e.getKey())) { // Only override
unspecified settings
                    LineSettings settings = new LineSettings();
                    settings.setIdentifier(e.getKey());
                    settings.setColorIndex(e.getValue());
                    settings.setMarker(e.getValue()); // Use same index
for markers by default
                    settingsMap.put(e.getKey(), settings);
                }
            }
        }

<<<<<<< HEAD
        return lineSettingAssignments;
    }

    private List<Map<String, Integer>> getColorAssignments() {
        String colorAssignmentString =
plotViewManifestation.getViewProperties().getProperty(PlotConstants.COLOR_ASSIGNMENTS,
String.class);
        List<Map<String, Integer>> colorAssignments = new ArrayList<Map<String,
Integer>>();
        if (colorAssignmentString != null) {
            StringTokenizer allAssignmentTokens = new
StringTokenizer(colorAssignmentString, "\n");

            while (allAssignmentTokens.hasMoreTokens()) {
                StringTokenizer colorAssignmentTokens = new
StringTokenizer(allAssignmentTokens.nextToken(), "\t");

                Map<String, Integer> subPlotMap = new HashMap<String,
Integer>();
                colorAssignments.add(subPlotMap);
                while (colorAssignmentTokens.hasMoreTokens()) {

                    String dataSet   = colorAssignmentTokens.nextToken();
                    int colorIndex =
Integer.parseInt(colorAssignmentTokens.nextToken());

                    subPlotMap.put(dataSet, colorIndex);
                }
```

```
                        }
                }
                return colorAssignments;
        }
=======
                colorAssignments = new ArrayList<Map<String, Integer>>();
                while (allAssignmentTokens.hasMoreTokens()) {
                        StringTokenizer colorAssignmentTokens = new
StringTokenizer(allAssignmentTokens.nextToken(), "\t");

                        Map<String, Integer> subPlotMap = new HashMap<String, Integer>();
                        colorAssignments.add(subPlotMap);
                        while (colorAssignmentTokens.hasMoreTokens()) {

                                String dataSet   = colorAssignmentTokens.nextToken();
                                int colorIndex =
Integer.parseInt(colorAssignmentTokens.nextToken());

                                subPlotMap.put(dataSet, colorIndex);
                        }
                }

                return colorAssignments;
        }

        /**
         * Retrieve persisted regression point assignments. Each element of the returned
list
         * corresponds, in order, to the sub-plots displayed, and maps subscription ID to
         * the number of regression points assigned and whether a regression line is
displayed.
         * The form of the values in the map is false|true:number of points
         * @return the persisted regression point assignments
         */
        public List<Map<String, String>> loadRegressionSettingsFromPersistence() {
                List<Map<String, String>> pointAssignments;

                String pointAssignmentString =
plotViewManifestation.getViewProperties().getProperty(PlotConstants.REGRESSION_LINE,
String.class);

                if (pointAssignmentString == null) return null;

                StringTokenizer allAssignmentTokens = new
StringTokenizer(pointAssignmentString, "\n");

                pointAssignments = new ArrayList<Map<String, String>>();
                while (allAssignmentTokens.hasMoreTokens()) {
                        StringTokenizer pointAssignmentTokens = new
StringTokenizer(allAssignmentTokens.nextToken(), "\t");

                        Map<String, String> subPlotMap = new HashMap<String, String>();
                        pointAssignments.add(subPlotMap);
                        while (pointAssignmentTokens.hasMoreTokens()) {

                                String dataSet   = pointAssignmentTokens.nextToken();
                                subPlotMap.put(dataSet, pointAssignmentTokens.nextToken());
                        }
                }

                return pointAssignments;
        }
        /**
         * Persist regression point assignments. Each element of the supplied list
corresponds,
         * in order, to the sub-plots displayed, and maps subscription ID to the number of
         * regression points assigned.
         * @param numberOfRegressionPoints the regression point assignments to persist.
         */
        public void persistRegressionSettings(List<Map<String, String>>
numberOfRegressionPoints) {
                /* Separate, because these are changed in a very different way from control
panel settings...
                 * But should these really be separate at this level? */

                ExtendedProperties viewProperties =
plotViewManifestation.getViewProperties();

                StringBuilder pointAssignmentBuilder = new StringBuilder();
                for (Map<String, String> subPlotMap : numberOfRegressionPoints) {
                        for (Entry<String,String> entry : subPlotMap.entrySet()) {
                                pointAssignmentBuilder.append(entry.getKey());
                                pointAssignmentBuilder.append('\t');
                                pointAssignmentBuilder.append(entry.getValue());
                                pointAssignmentBuilder.append('\t');
                        }
                        pointAssignmentBuilder.append('\n');
                }
                viewProperties.setProperty(PlotConstants.REGRESSION_LINE, "" +
                pointAssignmentBuilder.toString());

                if (plotViewManifestation.getManifestedComponent() != null) {
                        plotViewManifestation.getManifestedComponent().save();
                }
        }

>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6
```

fastPlotViews/src/main/java/gov/nasa/arc/mct/fastplot/view/PlotViewManifestation.java

Chunk 43: (combination/commenrary, method declaration, method invocation)

```
        public void persistPlotLineSettings() {
                if (thePlot != null)
<<<<<<< HEAD

        plotPersistanceHandler.persistLineSettings(thePlot.getLineSettings());
=======

        plotPersistanceHandler.persistColorSettings(thePlot.getColorAssignments());
        }

        /**
         * Pull regression point settings from persistence and apply them to the plot.
         */
        public void setupRegressionLines() {
                if (thePlot != null)

        plotPersistanceHandler.persistRegressionSettings(thePlot.getRegressionPoints());
>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6
        }
        public void persistPlotLineSettings() {
                if (thePlot != null)


        plotPersistanceHandler.persistLineSettings(thePlot.getLineSettings());

        }

        /**
         * Pull regression point settings from persistence and apply them to the plot.
         */
        public void setupRegressionLines() {
                if (thePlot != null)

        plotPersistanceHandler.persistRegressionSettings(thePlot.getRegressionPoints());
        }
```

Actually the second method is not in conflict. However, the first one can be inferred based on the name of methods.

Chunk 44: (combination/ method invocation)

```
        private void generatePlot() {
                plotDataAssigner.informFeedProvidersHaveChanged();
                createPlotAndAddItToPanel();
                plotDataAssigner.assignFeedsToSubPlots();
                enforceBackgroundColor(plotFrameBackground);
                thePlot.addPopupMenus();
<<<<<<< HEAD

        thePlot.setLineSettings(plotPersistanceHandler.loadLineSettingsFromPersistence());
=======

        thePlot.setRegressionPointAssignments(plotPersistanceHandler.loadRegressionSettingsF
romPersistence());

        thePlot.setColorAssignments(plotPersistanceHandler.loadColorSettingsFromPersistence(
));

>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6
        }
```

```
        private void generatePlot() {
                plotDataAssigner.informFeedProvidersHaveChanged();
                createPlotAndAddItToPanel();
                plotDataAssigner.assignFeedsToSubPlots();
                enforceBackgroundColor(plotFrameBackground);
                thePlot.addPopupMenus();

        thePlot.setLineSettings(plotPersistanceHandler.loadLineSettingsFromPersistence());

        thePlot.setRegressionPointAssignments(plotPersistanceHandler.loadRegressionSettingsF
romPersistence());

        }
```

It is difficult to solve automatically.

fastPlotViews/src/test/java/gov/nasa/arc/mct/fastplot/bridge/TestLegendEntryPopup.java

Chunk 45: Combination/Commentary, Method body, Method call, Method signature

```


        Mockito.when(mockLegendEntry.getComputedBaseDisplayName()).thenReturn("test");
                Mockito.when(mockLegendEntry.getFullBaseDisplayName()).thenReturn("test");
<<<<<<< HEAD
                Mockito.when(mockLegendEntry.getLineSettings()).thenReturn(new
LineSettings());
=======
                Mockito.when(mockLegendEntry.getNumberRegressionPoints()).thenReturn(15);
>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6

        }
```

```
        Mockito.when(mockLegendEntry.getComputedBaseDisplayName()).thenReturn("test");
                Mockito.when(mockLegendEntry.getFullBaseDisplayName()).thenReturn("test");
                Mockito.when(mockLegendEntry.getLineSettings()).thenReturn(new
LineSettings());
                Mockito.when(mockLegendEntry.getNumberRegressionPoints()).thenReturn(15);



        }
```

It is difficult to solve automatically. They are editing a methods body.

fastPlotViews/src/test/java/gov/nasa/arc/mct/fastplot/view/TestPlotPersistanceHandler.java

Chunk 46: (new code/ method invocation, variable)

```
        @Test
        public void testMigrateFixed() {
                Mockito.when(manifestation.getViewProperties()).thenReturn(new
ExtendedProperties());

                PlotPersistanceHandler h = new PlotPersistanceHandler(manifestation);
                h.persistPlotSettings(AxisOrientationSetting.X_AXIS_AS_TIME,
XAxisMaximumLocationSetting.MAXIMUM_AT_RIGHT,
                                YAxisMaximumLocationSetting.MAXIMUM_AT_TOP,
TimeAxisSubsequentBoundsSetting.SCRUNCH, NonTimeAxisSubsequentBoundsSetting.FIXED,
<<<<<<< HEAD
                                NonTimeAxisSubsequentBoundsSetting.FIXED,    0.0,    1.0,    new
GregorianCalendar(), new GregorianCalendar(), 0.0, 0.0, 0.0, true, false,
                                PlotConstants.DEFAULT_PLOT_LINE_DRAW,
                                PlotLineConnectionType.STEP_X_THEN_Y);

        manifestation.getViewProperties().setProperty(PlotConstants.TIME_AXIS_SUBSEQUENT_SET
TING, "FIXED");
                PlotSettings settings = h.loadPlotSettingsFromPersistance();
=======
                                NonTimeAxisSubsequentBoundsSetting.FIXED,    0.0,    1.0,    new
GregorianCalendar(), new GregorianCalendar(), 0.0, 0.0, 0.0, true, false);

        manifestation.getViewProperties().setProperty(PlotConstants.TIME_AXIS_SUBSEQUENT_SET
TING, "FIXED");
                manifestation.getViewProperties().setProperty(PlotConstants.REGRESSION_LINE,
"isp:123456\tfalse|20\t");
                PlotSettings settings = h.loadPlotSettingsFromPersistance();
                List<Map<String,              String>>              regSettings              =
h.loadRegressionSettingsFromPersistence();
>>>>>>> c07a2d235b353a36a66d6d0befefb495ce0b22b6
```

```
                Assert.assertEquals(settings.timeAxisSubsequent,
TimeAxisSubsequentBoundsSetting.JUMP);
                Assert.assertTrue(settings.pinTimeAxis);
                Assert.assertEquals(regSettings.iterator().next().get("isp:123456"),
"false|20");
        }
```

```
        @Test
        public void testMigrateFixed() {
                Mockito.when(manifestation.getViewProperties()).thenReturn(new
ExtendedProperties());

                PlotPersistanceHandler h = new PlotPersistanceHandler(manifestation);
                h.persistPlotSettings(AxisOrientationSetting.X_AXIS_AS_TIME,
XAxisMaximumLocationSetting.MAXIMUM_AT_RIGHT,
                                YAxisMaximumLocationSetting.MAXIMUM_AT_TOP,
TimeAxisSubsequentBoundsSetting.SCRUNCH, NonTimeAxisSubsequentBoundsSetting.FIXED,
                                NonTimeAxisSubsequentBoundsSetting.FIXED,   0.0,   1.0,   new
GregorianCalendar(), new GregorianCalendar(), 0.0, 0.0, 0.0, true, false,
                                PlotConstants.DEFAULT_PLOT_LINE_DRAW,
                                PlotLineConnectionType.STEP_X_THEN_Y);

        manifestation.getViewProperties().setProperty(PlotConstants.TIME_AXIS_SUBSEQUENT_SET
TING, "FIXED");
                PlotSettings settings = h.loadPlotSettingsFromPersistance();
                List<Map<String,              String>>              regSettings              =
h.loadRegressionSettingsFromPersistence();

                Assert.assertEquals(settings.timeAxisSubsequent,
TimeAxisSubsequentBoundsSetting.JUMP);
                Assert.assertTrue(settings.pinTimeAxis);
                Assert.assertEquals(regSettings.iterator().next().get("isp:123456"),
"false|20");
        }
```