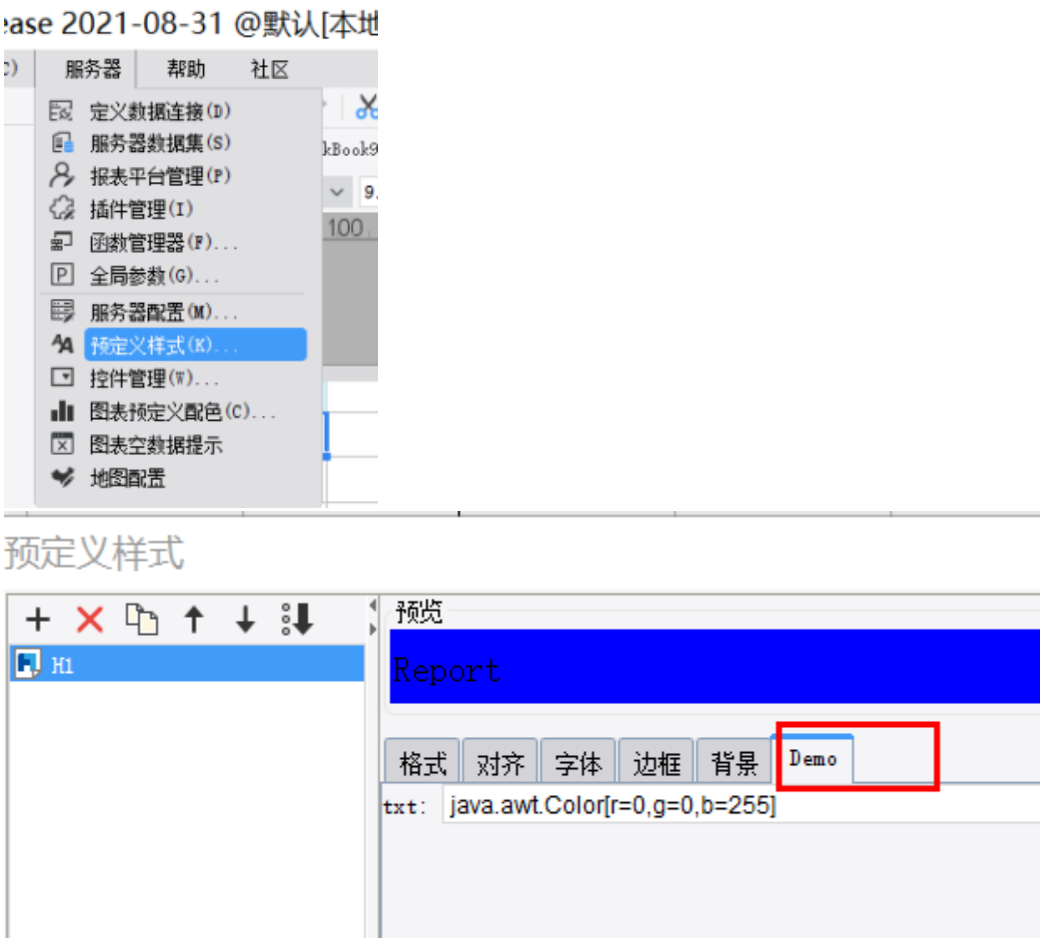
com.fr.design.fun.MultiStyleUIConfigProvider

MultiStyleUIConfigProviderStyleUIConfigProvider



MultiStyleUIConfigProvider.java

```
package com.fr.design.fun;

import com.fr.common.annotations.Open;
import com.fr.stable.fun.mark.Mutable;

import java.util.List;

/**
 * Created by kerry on 2019-11-11
 */
@Open
public interface MultiStyleUIConfigProvider extends Mutable {
    String XML_TAG = "MultiStyleUIConfigProvider";

    int CURRENT_LEVEL = 1;

    /**
     * list
     *
     * @return list
     */
    List<StyleUIConfigProvider> getConfigList();
}
```

StyleUIConfigProvider.java

```
package com.fr.design.fun;

import com.fr.base.Style;
import com.fr.common.annotations.Open;
import com.fr.stable.fun.mark.Mutable;

import javax.swing.JComponent;
import javax.swing.event.ChangeListener;

/**
 * Created by kerry on 2019-11-11
 */
@Open
public interface StyleUIConfigProvider extends Mutable {
    String XML_TAG = "CustomStyleUIConfigProvider";

    int CURRENT_LEVEL = 1;

    /**
     * @return
     */
    String configName();

    /**
     * @param changeListener listener
     * @return component
     */
    JComponent uiComponent(ChangeListener changeListener);

    /**
     * @return
     */
    Style updateConfig();

    /**
     * @param style
     */
    void populateConfig(Style style);
}
```

FR	10.0		

plugin.xml

```
<extra-designer>
        <!--  -->
        <MultiStyleUIConfigProvider class="your class name"/>
</extra-designer>
```

StylePane.java

```
package com.fr.design.style;

import com.fr.base.CellBorderStyle;
import com.fr.base.NameStyle;
import com.fr.base.ScreenResolution;
import com.fr.base.Style;
import com.fr.base.core.StyleUtils;
import com.fr.design.ExtraDesignClassManager;
import com.fr.design.beans.BasicBeanPane;
import com.fr.design.dialog.FineJOptionPane;
import com.fr.design.fun.MultiStyleUIConfigProvider;
import com.fr.design.fun.StyleUIConfigProvider;
import com.fr.design.gui.frpane.UITabbedPane;
import com.fr.design.layout.FRGUIPaneFactory;
import com.fr.design.mainframe.ElementCasePane;
import com.fr.design.style.background.BackgroundPane;
import com.fr.design.utils.gui.GUICoreUtils;
import com.fr.grid.selection.CellSelection;
import com.fr.grid.selection.FloatSelection;
import com.fr.grid.selection.Selection;
import com.fr.log.FineLoggerFactory;
import com.fr.plugin.solution.sandbox.collection.PluginSandboxCollections;
import com.fr.report.cell.CellElement;
import com.fr.report.cell.DefaultTemplateCellElement;
import com.fr.report.cell.FloatElement;
import com.fr.report.cell.TemplateCellElement;
import com.fr.report.elementcase.ElementCase;
import com.fr.report.elementcase.TemplateElementCase;

import javax.swing.JComponent;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.util.List;
import java.util.Set;

/**
 * Style Pane.
 */
public class StylePane extends BasicBeanPane<Style> implements ChangeListener {
    private static final int BORDER_ARRAY_LENGTH = 4;
    private static final int ALIGNMENT_INDEX = 1;
    private static final int FONT_INDEX = 2;
    private static final int BORDER_INDEX = 3;
    private static final int BACKGROUND_INDEX = 4;
    private static final int NEXT_TAB_INDEX = 5;
        private ElementCasePane reportPane;
        protected Style editing;
        private NameStyle globalStyle;
        private FormatPane formatPane = null;
        private AlignmentPane alignmentPane = null;
        private FRFontPane frFontPane = null;
        private BorderPane borderPane = null;
        private BackgroundPane backgroundPane = null;
        private static List<StyleUIConfigProvider> configList = PluginSandboxCollections.newSandboxList();
        private PreivewArea previewArea;
        private JPanel previewPane;

        static {
                Set<MultiStyleUIConfigProvider> preferenceConfigProviders = ExtraDesignClassManager.
```

```
getInstance().getArray(MultiStyleUIConfigProvider.XML_TAG);
                for (MultiStyleUIConfigProvider provider : preferenceConfigProviders) {
                        configList.addAll(provider.getConfigList());
                }
        }

        /**
         * Constructor
         */
        public StylePane() {
                this.setLayout(FRGUIPaneFactory.createBorderLayout());

                previewPane = FRGUIPaneFactory.createBorderLayout_S_Pane();
                this.add(previewPane, BorderLayout.NORTH);
                previewPane.setBorder(GUICoreUtils.createTitledBorder(com.fr.design.i18n.Toolkit.i18nText("Fine-
Design_Basic_Preview"), null));

                previewArea = new PreivewArea();
                previewPane.add(previewArea, BorderLayout.CENTER);

                JPanel previewControlPane = FRGUIPaneFactory.createNColumnGridInnerContainer_S_Pane(1);
                previewPane.add(previewControlPane, BorderLayout.EAST);

                UITabbedPane mainTabbedPane = new UITabbedPane();
                this.add(mainTabbedPane, BorderLayout.CENTER);

                mainTabbedPane.addTab(com.fr.design.i18n.Toolkit.i18nText("Fine-Design_Chart_Format"), this.
getFormatPane());
                mainTabbedPane.addTab(com.fr.design.i18n.Toolkit.i18nText("Fine-Design_Report_Alignment"),
FRGUIPaneFactory.createY_AXISBoxInnerContainer_L_Pane());
                mainTabbedPane.addTab(com.fr.design.i18n.Toolkit.i18nText("Fine-Design_Basic_Sytle_FRFont"),
FRGUIPaneFactory.createY_AXISBoxInnerContainer_L_Pane());
                mainTabbedPane.addTab(com.fr.design.i18n.Toolkit.i18nText("Fine-Design_Report_Border"),
FRGUIPaneFactory.createY_AXISBoxInnerContainer_L_Pane());
                mainTabbedPane.addTab(com.fr.design.i18n.Toolkit.i18nText("Fine-Design_Basic_Background"),
FRGUIPaneFactory.createY_AXISBoxInnerContainer_L_Pane());

                for (StyleUIConfigProvider config : configList) {
                        mainTabbedPane.addTab(config.configName(), FRGUIPaneFactory.
createY_AXISBoxInnerContainer_L_Pane());
                }
                mainTabbedPane.addChangeListener(tabChangeActionListener);
                this.setPreferredSize(new Dimension(450, 480));
        }

        @Override
        protected String title4PopupWindow() {
                return com.fr.design.i18n.Toolkit.i18nText("Fine-Design_Basic_Style");
        }

        public void stateChanged(ChangeEvent e) {
                this.updatePreviewArea();
        }

        protected FormatPane getFormatPane() {
                if (this.formatPane == null) {
                        this.formatPane = new FormatPane();
                        if (this.editing != null) {
                                this.formatPane.populate(this.editing.getFormat());
                        }
                }

                return this.formatPane;
        }

        private AlignmentPane getAlignmentPane() {
                if (this.alignmentPane == null) {
                        this.alignmentPane = new AlignmentPane();
                        this.alignmentPane.addChangeListener(this);
                        if (this.editing != null) {
                                this.alignmentPane.populate(this.editing);
```

```
                        }
                }

                return this.alignmentPane;
        }

        private FRFontPane getFRFontPane() {
                if (this.frFontPane == null) {
                        this.frFontPane = new FRFontPane();
                        this.frFontPane.addChangeListener(this);
                        if (this.editing != null) {
                                this.frFontPane.populate(this.editing.getFRFont());
                        }
                }

                return this.frFontPane;
        }

        class FRFontListSelectionListener implements ListSelectionListener {

                public void valueChanged(ListSelectionEvent e) {
                        updatePreviewArea();
                }
        }

        private BorderPane getBorderPane() {
                if (this.borderPane == null) {
                        this.borderPane = new BorderPane();
                        this.borderPane.addChangeListener(this);

                        // p:borderPaneStyle
                        // reportPane,StyleoldStyle
                        if (this.reportPane != null) {
                                Object[] fourObjectArray = BorderUtils.createCellBorderObject(reportPane);

                                if (fourObjectArray != null && fourObjectArray.length == BORDER_ARRAY_LENGTH) {
                                        this.borderPane.populate((CellBorderStyle) fourObjectArray[0],
((Boolean) fourObjectArray[1]).booleanValue(),
                                                        ((Integer) fourObjectArray[2]).intValue(), (Color)
fourObjectArray[3]);
                                }
                        } else {
                                if (this.editing != null) {
                                        this.borderPane.populate(this.editing);
                                }
                        }
                }

                return this.borderPane;
        }

        private BackgroundPane getBackgroundPane() {
                if (this.backgroundPane == null) {
                        this.backgroundPane = new BackgroundPane();
                        this.backgroundPane.addChangeListener(this);
                        if (this.editing != null) {
                                this.backgroundPane.populate(this.editing.getBackground());
                        }
                }

                return this.backgroundPane;
        }

        public NameStyle getGlobalStyle() {
                return this.globalStyle;
        }

        public void setGlobalStyle(NameStyle globalStyle) {
                this.globalStyle = globalStyle;
        }
```

```
        /**
         * Populate
         */
        public void populate(ElementCasePane reportPane) {
                // p:ReportPane, BorderPane.
                this.reportPane = reportPane;
                this.populateBean(this.analyzeCurrentStyle(reportPane));
                updatePreviewArea();
        }

        /**
         * Style
         */
        private Style analyzeCurrentStyle(ElementCasePane reportPane) {
                Style style = null;

                // p:CellElementStyle.
                Selection sel = reportPane.getSelection();
                if (sel instanceof FloatSelection) {
                        // got simple cell element from column and row.
                        ElementCase report = reportPane.getEditingElementCase();

                        FloatElement selectedFloatElement = report.getFloatElement(((FloatSelection) sel).
getSelectedFloatName());
                        style = selectedFloatElement.getStyle();
                } else {
                        CellSelection cs = (CellSelection) sel;

                        // got simple cell element from column and row.
                        ElementCase report = reportPane.getEditingElementCase();
                        CellElement editCellElement = report.getCellElement(cs.getColumn(), cs.getRow());

                        if (editCellElement != null && editCellElement.getStyle() != null) {// editCellElement

// cellstyle.
                                try {
                                        style = (Style) editCellElement.getStyle().clone();
                                } catch (CloneNotSupportedException e) {
                    FineLoggerFactory.getLogger().error(e.getMessage(), e);
                                }
                        }
                }

                if (style == null) {
                        // peter:Style.
                        style = Style.DEFAULT_STYLE;
                }

                return style;
        }

        /**
         *
         */
        public boolean updateGlobalStyle(ElementCasePane reportPane) {
                updatePreviewArea();
                NameStyle nameStyle = this.getGlobalStyle();

                return applyStyle(reportPane, new NameStyleG(nameStyle));
        }

        /**
         * Update. Return whether style changed.
         */
        public boolean update(ElementCasePane reportPane) {
                Style oldStyle = analyzeCurrentStyle(reportPane);

                return applyStyle(reportPane, new NormalStyleG(oldStyle, this.updateBean()));
        }

        private static abstract class GStyle {
```

```
                protected abstract Style getStyle2Apply(Style styleOfEl);
        }

        private static class NameStyleG extends GStyle {
                NameStyle nameStyle;

                NameStyleG(NameStyle nameStyle) {
                        this.nameStyle = nameStyle;
                }

                @Override
                protected Style getStyle2Apply(Style styleOfEl) {
                        return nameStyle;
                }
        }

        private static class NormalStyleG extends GStyle {
                Style styleOfReportPane;
                Style newStyle;

                NormalStyleG(Style styleOfReportPane, Style newStyle) {
                        this.styleOfReportPane = styleOfReportPane;
                        this.newStyle = newStyle;
                }

                @Override
                protected Style getStyle2Apply(Style styleOfEl) {
                        return StyleUtils.applyCellStyle(styleOfReportPane, newStyle, styleOfEl);
                }
        }

        private boolean applyStyle(ElementCasePane reportPane, GStyle gstyle) {
                TemplateElementCase report = reportPane.getEditingElementCase();
                Selection sel = reportPane.getSelection();
                if (sel instanceof FloatSelection) {
                        FloatElement floatElement = report.getFloatElement(((FloatSelection) sel).
getSelectedFloatName());

                        // Apply style.
                        floatElement.setStyle(gstyle.getStyle2Apply(floatElement.getStyle()));
                } else {
                        CellSelection cs = (CellSelection) sel;
                        // Got editCellElement.
                        TemplateCellElement editCellElement;

                        //
                        for (int j = 0; j < cs.getRowSpan(); j++) {
                                for (int i = 0; i < cs.getColumnSpan(); i++) {
                                        int column = i + cs.getColumn();
                                        int row = j + cs.getRow();

                                        editCellElement = report.getTemplateCellElement(column, row);
                                        if (editCellElement == null) {
                                                editCellElement = new DefaultTemplateCellElement(column, row);
                                                report.addCellElement(editCellElement);
                                        }

                                        // Apply cellstyle.
                                        editCellElement.setStyle(gstyle.getStyle2Apply(editCellElement.
getStyle()));
                                }
                        }
                }

                // p:borderpaneupdate borderpane
                if (this.borderPane != null) {
                        BorderUtils.update(reportPane, this.borderPane.update());
                }
                reportPane.repaint();
                return true;
        }
```

```
/**
 * Populate Style
 */
@Override
public void populateBean(Style ob) {
        this.editing = ob == null ? Style.getInstance() : ob;

        if (this.formatPane != null) {
                this.formatPane.populate(editing.getFormat());
        }
        if (this.alignmentPane != null) {
                this.alignmentPane.populate(editing);
        }
        if (this.frFontPane != null) {
                this.frFontPane.populate(editing.getFRFont());
        }
        if (this.borderPane != null) {
                this.borderPane.populate(editing);
        }
        if (this.backgroundPane != null) {
                this.backgroundPane.populate(editing.getBackground());
        }
        for(StyleUIConfigProvider tabConfig : configList){
                tabConfig.populateConfig(this.editing);
        }
        updatePreviewArea();
}

/**
 * Update Style
 */
@Override
public Style updateBean() {
        // need the check the valid of CellAlignment pane.
        try {
                if (this.alignmentPane != null) {
                        this.alignmentPane.checkValid();
                }
        } catch (Exception exp) {
                FineJOptionPane.showMessageDialog(this, exp.getMessage());
                return editing;
        }

        Style style = editing;
        if (this.formatPane != null) {
                style = style.deriveFormat(this.formatPane.update());
        }
        if (this.alignmentPane != null) {
                style = this.alignmentPane.update(style);
        }
        if (this.frFontPane != null) {
                style = style.deriveFRFont(this.frFontPane.update());
        }
        if (this.borderPane != null) {
                style = this.borderPane.update(style);
        }
        if (this.backgroundPane != null) {
                style = style.deriveBackground(this.backgroundPane.update());
        }
        for(StyleUIConfigProvider tabConfig : configList){
                style = tabConfig.updateConfig();
        }

        return style;
}

protected ChangeListener tabChangeActionListener = new ChangeListener() {

        public void stateChanged(ChangeEvent evt) {
                Object tabObj = evt.getSource();
```

```
                        if (tabObj == null || !(tabObj instanceof JTabbedPane)) {
                                return;
                        }
                        JTabbedPane tabbedPane = (JTabbedPane) tabObj;

                        int selectedIndex = tabbedPane.getSelectedIndex();
                        if (tabbedPane.getComponentAt(selectedIndex).getClass() == JPanel.class) {// JPanel,
                                if (selectedIndex == ALIGNMENT_INDEX) {
                                        tabbedPane.setComponentAt(selectedIndex, StylePane.this.
getAlignmentPane());
                                } else if (selectedIndex == FONT_INDEX) {
                                        tabbedPane.setComponentAt(selectedIndex, StylePane.this.
getFRFontPane());
                                } else if (selectedIndex == BORDER_INDEX) {
                                        tabbedPane.setComponentAt(selectedIndex, StylePane.this.
getBorderPane());
                                } else if (selectedIndex == BACKGROUND_INDEX) {
                                        tabbedPane.setComponentAt(selectedIndex, StylePane.this.
getBackgroundPane());
                                } else if (configList.size() + NEXT_TAB_INDEX > selectedIndex && configList.get
(selectedIndex - NEXT_TAB_INDEX) != null) {
                                        tabbedPane.setComponentAt(selectedIndex, configList.get(selectedIndex -
NEXT_TAB_INDEX).uiComponent(StylePane.this));
                                        configList.get(selectedIndex - NEXT_TAB_INDEX).populateConfig(StylePane.
this.editing);
                                }
                        }
                        updatePreviewArea();
                }
        };

        public void updatePreviewArea() {
                if (editing != null) {
                        previewArea.setStyle(this.updateBean());
                }
        }

        public PreivewArea getPreviewArea() {
                return this.previewArea;
        }

        /**
         * Style
         *
         * @author richer
         */
        public static class PreivewArea extends JComponent {

                private String paintText = "Report";
                private Style style = Style.DEFAULT_STYLE;

                public PreivewArea() {
                        setPreferredSize(new Dimension(40, 40));
                }

                public void setStyle(Style style) {
                        this.style = style;
                        repaint();
                }

                @Override
                public void paintComponent(Graphics g) {
                        Graphics2D g2d = (Graphics2D) g;
                        int resolution = ScreenResolution.getScreenResolution();

                        if (style == Style.DEFAULT_STYLE) {
                                // style,"Report"
                                Style.paintContent(g2d, paintText, style, getWidth() - 3, getHeight() - 3,
resolution);
                                return;
                        }
```

```
                        Style.paintBackground(g2d, style, getWidth() - 3, getHeight() - 3);

                        Style.paintContent(g2d, paintText, style, getWidth() - 3, getHeight() - 3, resolution);

                        Style.paintBorder(g2d, style, getWidth() - 3, getHeight() - 3);
                }

                @Override
                public Dimension getMinimumSize() {
                        return getPreferredSize();
                }
        }

}
```

StyleUIConfigProviderMultiStyleUIConfigProvider

demo-multi-style-ui-config-provider